aws

Developer Guide

Amazon Simple Email Service

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

Amazon Simple Email Service: Developer Guide

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

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

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

Table of Contents

What is Amazon SES? 1
Benefits 1
Related services 1
Pricing 2
Regions 2
SES regions and endpoints 3
Sandbox removal and sending limit increases 4
Verification of email addresses and domains 4
Easy DKIM 4
Account-level suppression list 4
Feedback notifications 5
SMTP credentials 5
Feedback endpoints used for custom MAIL FROM domains 5
Sending authorization 6
Email receiving 6
Quotas 6
Email sending quotas 7
Email receiving quotas 11
Mail Manager quotas 12
General quotas 13
Types of credentials 14
How Amazon SES works 18
After a sender sends an email request to SES 19
After Amazon SES sends an email 20
Email format 22
Understanding deliverability 26
Email best practices 32
Working with AWS SDKs 38
Agent setup guide 39
Before you begin 39
Step 1: Connect your agent to AWS 39
Claude Code 40
Codex 40
Cursor 40

Kiro ... 40
Alternative: Automatic setup with the AWS Command Line Interface ... 41
Alternative: Connect directly with OAuth 2.1 ... 41
Another agent or custom framework ... 41
Step 2: Use the Amazon SES skill ... 42
Keep your setup current ... 42
Getting started ... 43
Setting up ... 43
Sign up for an AWS account ... 43
Set up your SES account ... 43
Download an AWS SDK (For using the SES APIs) ... 44
Migrating to Amazon SES ... 44
Step 1. Verify your domain ... 44
Step 2. Request production access ... 45
Step 3. Configure domain authentication systems ... 45
Step 4. Generate your SMTP credentials ... 45
Step 5. Connect to an SMTP endpoint ... 45
Next steps ... 45
Request production access ... 46
Sending limits ... 50
Increasing your sending quotas ... 51
Automatically increased sending quotas ... 52
User requested increased sending quotas ... 52
Monitoring your sending quotas ... 53
Monitoring your sending quotas using the Amazon SES console ... 54
Monitoring your sending quotas using the Amazon SES API ... 55
Sending quota errors ... 55
Reaching sending limits with the Amazon SES API ... 55
Reaching sending limits with SMTP ... 56
Set up email sending ... 57
Using the SMTP interface ... 57
Requirements to send email over SMTP ... 58
Methods to send email over SMTP ... 58
Email information to provide ... 59
Obtaining SMTP credentials ... 59
Using Mail Manager ... 68

Connecting to an SMTP endpoint ... 71
Using software packages to send email ... 72
Sending emails programmatically ... 74
Integrating with your existing email server ... 84
Testing your connection to the Amazon SES SMTP interface ... 97
Using the API ... 106
Sending formatted email ... 107
Sending raw email ... 108
Using templates to send email ... 119
Sending email using an AWS SDK ... 143
Content encodings ... 162
Configuration overrides ... 163
What are configuration overrides? ... 163
How tracking overrides work ... 164
IAM permissions for tracking overrides ... 166
Using tracking overrides ... 170
Effect of tracking overrides on events and metrics ... 171
Best practices and considerations ... 171
Supported security protocols ... 172
Email sender to Amazon SES ... 172
Amazon SES to receiver ... 173
End-to-end encryption ... 173
Supported header fields ... 174
Email attachments ... 177
How attachments work ... 177
Attachment object structure ... 178
Best practices ... 182
Unsupported attachment types ... 182
Email receiving ... 184
Email receiving concepts & use cases ... 185
Recipient-based control using receipt rules ... 185
IP-based control using IP address filters ... 187
Email-receiving process ... 188
Use cases & restrictions ... 189
Email authentication and malware detection ... 192
Setting up email receiving ... 193

Verifying your domain ... 194
Publishing an MX record ... 194
Giving permission ... 197
Email receiving console walkthroughs ... 206
Creating receipt rules ... 206
Create IP filters ... 246
Email receiving metrics ... 247
Tenants ... 251
What is tenant management? ... 251
How it works ... 251
Resource association ... 252
Reputation monitoring ... 252
Setting up ... 253
Creating tenants ... 253
Assigning resources ... 253
Reputation policies ... 254
Sending email ... 255
SendEmail API ... 255
SMTP ... 256
Tenant status ... 257
Status and metrics ... 257
Pause/Unpause ... 258
Reputation findings ... 258
View findings ... 259
Understanding findings ... 260
Resolving issues ... 260
Monitoring ... 261
CloudWatch metrics ... 261
EventBridge notifications ... 261
Trust & Safety ... 264
Best practices ... 265
Suppression lists ... 265
Limitations ... 266
Pricing ... 266
Verified identities ... 268
Creating & verifying identities ... 268

Creating a domain identity ... 271
Verifying a domain identity ... 275
Creating an email address identity ... 281
Verifying an email address identity ... 282
Create & verify an identity and assign a default configuration set at the same time (API) ... 283
Using custom verification email templates ... 284
Managing identities ... 296
View identities using the console ... 297
Delete an identity using the console ... 298
Edit an identity using the console ... 298
Edit an identity to use a default configuration set using the SES API ... 299
Retrieve the default configuration set used by the identity using the SES API ... 300
Override the current default configuration set used by the identity using the SES API ... 301
Configuring identities ... 301
Email authentication methods ... 302
Setting up event notifications ... 357
Using identity authorization ... 396
Using sending authorization ... 412
Sending test emails with the simulator ... 443
Using the mailbox simulator from the console ... 443
Using the mailbox simulator manually ... 445
Configuration sets ... 449
Create configuration sets ... 450
Create a configuration set ... 450
Create a configuration set (AWS CLI) ... 455
Manage configuration sets ... 456
View, edit, & delete configuration set (console) ... 457
List configuration sets (AWS CLI) ... 457
Get configuration set details (AWS CLI) ... 457
Delete a configuration set (AWS CLI) ... 458
Stop sending email from a configuration set (AWS CLI) ... 458
Understanding default configuration sets ... 458
Create event destinations ... 460
Assign IP pools ... 464
Configure custom open and click domains ... 465
Specify configuration sets in email ... 474

View and export reputation metrics ... 474
Enabling the export of reputation metrics ... 474
Disabling the export of reputation metrics ... 475
Global endpoints ... 476
What are Global endpoints? ... 476
How Global endpoints work ... 476
Setting up Global endpoints ... 477
Prerequisites ... 477
Creating a Global endpoint ... 477
Global endpoint states ... 478
Preparing the secondary region ... 478
(1) Duplicate configuration sets ... 479
(2) Duplicate verified domain identities ... 480
(3) Duplicate production limits ... 481
Using Global endpoints ... 482
Integrating with your application ... 482
Monitoring and metrics ... 483
Best practices and considerations ... 484
Pricing ... 485
Dedicated IP addresses ... 486
Ease of setup ... 488
Reputation management ... 488
Predictability of sending patterns ... 489
Volume of outbound email ... 489
Additional costs ... 490
Control over sender reputation ... 490
Ability to isolate sender reputation ... 490
Known, unchanging IP addresses ... 490
Standard ... 491
Request & relinquish ... 491
Warming up ... 493
Creating pools ... 497
Dedicated IP addresses (managed) ... 499
Benefits and features ... 500
Importance of warmup ... 501
FAQS you should know ... 502

Creating a managed IP pool ... 503
Viewing pool sending and capacity ... 506
Deleting a managed IP pool ... 508
Bring your own IP addresses ... 509
Requirements ... 509
Considerations ... 510
Using your own IP addresses with Amazon SES ... 510
Virtual Deliverability Manager ... 512
SES deliverability ... 512
Getting started ... 513
Dashboard ... 517
Advisor ... 537
Settings ... 543
Global deliverability ... 546
Getting started ... 547
Campaign analytics ... 548
Inbox placement rates ... 549
Inbox placement tests ... 550
Blocklists ... 552
Email Validation ... 554
Email Validation API ... 555
API validation (console) ... 556
API validation (AWS CLI) ... 556
Auto Validation ... 558
Auto Validation (console) ... 559
Configuration set overrides ... 560
Auto Validation (AWS CLI) ... 561
Email Validation Dashboard ... 563
Mail Manager ... 565
Getting started ... 566
Getting started ... 567
Ingress endpoints ... 568
Configuring ingress endpoints ... 568
Credentials rotation ... 573
TLS policy ... 574
mTLS authentication ... 576

Creating an ingress endpoint (console) ... 577
Traffic policies & policy statements ... 579
Creating traffic policies & policy statements (console) ... 581
Policy statement conditions ... 582
Rule sets & rules ... 583
Creating rule sets & rules (console) ... 584
Rule conditions & actions ... 586
SMTP relay ... 590
Creating an SMTP relay (console) ... 592
Setting up Google Workspaces ... 595
Setting up Microsoft Office 365 ... 596
Address Lists ... 600
What are Address Lists? ... 600
How Address Lists work ... 600
Setting up Address Lists ... 600
Using Address Lists ... 605
Best Practices ... 484
Email archiving ... 607
Using email archiving (console) ... 608
Email Add Ons ... 613
Subscribing to Add Ons (console) ... 613
Permission policies ... 615
Ingress endpoint policies ... 615
SMTP relay policies ... 617
Email archiving policies ... 619
Rule action policies ... 623
Logging ... 631
Setting up log delivery ... 631
Interpreting the logs ... 634
Lists and subscriptions ... 641
Global suppression list ... 643
Global suppression list considerations ... 643
Using the account-level suppression list ... 644
Account-level suppression list considerations ... 645
Enabling the account-level suppression list ... 647
Enabling your account-level suppression list for a configuration set ... 648

Adding individual email addresses to your account-level suppression list 650
Adding email addresses in bulk to your account-level suppression list 651
Viewing a list of addresses that are on your account-level suppression list 654
Removing individual email addresses from your account-level suppression list 657
Removing email addresses in bulk from your account-level suppression list 658
Viewing a list of import jobs for the account 662
Getting information about an import job for the account 663
Disabling the account-level suppression list 665
Tenant-level suppression lists 665
How tenant-level suppression works 666
Suppression precedence 667
Automatic suppression recording 668
Identifying tenant suppression bounces 668
Managing entries 669
Considerations and limitations 669
Related resources 670
Configuring tenant-level suppression 670
Managing tenant suppression entries 678
Using configuration set-level suppression 685
Enabling configuration set-level suppression 688
Using list management 689
List management overview 689
Configuring list management 690
List management walkthrough with examples 695
Using subscription management 697
Subscription management overview 698
Unsubscribe header considerations 699
Adding an unsubscribe footer link 700
Pricing plans 701
Choose and manage your plan 701
Choosing a plan 701
Changing or canceling your plan 702
Monitoring sending activity 703
Monitoring using the console 709
Account dashboard 710
Reputation metrics 711

SMTP settings ... 712
Using the console to monitor metrics ... 712
Monitor using the API ... 714
Calling the GetSendStatistics API operation using the AWS CLI ... 714
Calling the GetSendStatistics operation programmatically ... 715
Monitor email sending using event publishing ... 718
How event publishing works with configuration sets and message tags ... 718
Fine-grained feedback for email campaigns ... 720
How to use event publishing ... 721
Event publishing terminology ... 721
Setting up event publishing ... 723
Working with event data ... 738
Monitoring sender reputation ... 812
Using reputation metrics ... 812
Reputation metrics messages ... 814
General Status Messages ... 815
Bounce Rate Notification ... 816
Complaint Rate Notification ... 818
Anti-Spam Organization Notification ... 819
Listbombing Notification ... 820
Direct Feedback Notification ... 822
Domain Blocklist Notification ... 823
Internal Review Notification ... 824
Mailbox Provider Notification ... 826
Recipient Feedback Notification ... 827
Related Account Notification ... 829
Spamtrap Notification ... 829
Vulnerable Site Notification ... 831
Compromised Credentials Notification ... 832
Other Notification ... 833
Creating alarms using CloudWatch ... 833
SNDS metrics for dedicated IPs ... 835
Troubleshooting questions ... 837
Automatically pausing email sending ... 838
For your entire account ... 838
For a configuration set ... 845

Monitoring using EventBridge 855
SES events 855
Events schema reference 857
Virtual Deliverability Manager advisor status schema 858
SES email sending status schema 859
Using EventBridge 862
Specify a sample event in EventBridge 862
Event patterns for SES events 863
Additional EventBridge resources 865
Code examples 866
Amazon SES 868
Basics 870
Scenarios 1001
Amazon SES API v2 1049
Basics 1050
Scenarios 1131
Security 1189
Data protection 1190
Data at rest encryption 1191
Encryption in transit 1203
Deleting personal data 1203
Identity and access management 1210
Creating IAM Policies for Access to SES 1211
Example IAM Policies for SES 1214
AWS managed policies 1220
Using service-linked roles 1222
Logging and monitoring 1225
Logging API calls 1226
Compliance validation 1236
Resilience 1237
Infrastructure security in SES 1237
VPC endpoints 1238
Walkthrough example of setting up SES in Amazon VPC 1239
Troubleshooting 1243
General issues 1244
Changes that I make are not immediately visible 1244

Verification problems ... 1245
Domain verification problems ... 1245
Checking domain verification settings ... 1246
Email verification problems ... 1248
DKIM problems ... 1248
Delivery problems ... 1251
Problems with received emails ... 1252
Notification problems ... 1253
Email sending errors ... 1254
Increasing throughput ... 1257
SMTP issues ... 1258
SMTP response codes ... 1260
FAQs ... 1267
Managed DIPS FAQs ... 1267
Managed DIPS FAQ Q1 ... 1267
Managed DIPS FAQ Q2 ... 1268
Managed DIPS FAQ Q3 ... 1268
Managed DIPS FAQ Q4 ... 1268
Managed DIPS FAQ Q5 ... 1268
Managed DIPS FAQ Q6 ... 1269
Sending review process FAQs ... 1269
Account Under Review ... 1270
Sending Pauses ... 1273
Bounces ... 1277
Complaints ... 1280
Spamtraps ... 1287
Manual investigations ... 1289
DNS Blackhole List (DNSBL) FAQs ... 1291
DNSBL FAQ Q1 ... 1291
DNSBL FAQ Q2 ... 1291
DNSBL FAQ Q3 ... 1292
DNSBL FAQ Q4 ... 1292
DNSBL FAQ Q5 ... 1292
DNSBL FAQ Q6 ... 1293
Email metrics FAQs ... 1295
General ... 1295

Open Tracking 1296
Click Tracking 1298
Quick Find Index 1301
How-tos & concepts 1301
FAQs on supported protocols 1308
What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ? 1308
How do I get started with the Smithy RPC v2 CBOR protocol? 1308
What are the risks of enabling the Smithy RPC v2 CBOR protocol? 1309
What if my solution does not support Smithy RPC v2 CBOR? 1309
What AWS Regions support the Smithy RPC v2 CBOR protocol? 1309
What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR? 1309
Will the AWS JSON protocol be deprecated? 1309
Where can I find more information about Smithy RPC v2 CBOR? 1310

What is Amazon SES?

Amazon Simple Email Service (SES) is an email platform that provides an easy, cost-effective way for you to send and receive email using your own email addresses and domains.

For example, you can send marketing emails such as special offers, transactional emails such as order confirmations, and other types of correspondence such as newsletters. When you use Amazon SES to receive mail, you can develop software solutions such as email autoresponders, email unsubscribe systems, and applications that generate customer support tickets from incoming emails.

For more information about topics related to Amazon SES, see the AWS Messaging and Targeting Blog.

Benefits

Building a large-scale email solution is often a complex and costly challenge for a business. You must deal with infrastructure challenges such as email server management, network configuration, and IP address reputation. Additionally, many third-party email solutions require contract and price negotiations, as well as significant up-front costs. Amazon SES eliminates these challenges and enables you to benefit from the years of experience and sophisticated email infrastructure Amazon.com has built to serve its own large-scale customer base.

Related services

Amazon SES integrates seamlessly with other AWS products. For example, you can:

- Add email-sending capabilities to any application.
- You can send email from Amazon EC2 by using an AWS SDK, by using the Amazon SES SMTP interface, or by making calls directly to the Amazon SES API.
- Use AWS Elastic Beanstalk to create an email-enabled application such as a program that uses Amazon SES to send a newsletter to customers.
- Set up Amazon Simple Notification Service (Amazon SNS) to notify you of your emails that bounced, produced a complaint, or were successfully delivered to the recipient's mail server. When you use Amazon SES to receive emails, your email content can be published to Amazon SNS topics.

- Use the AWS Management Console to set up Easy DKIM, which is a way to authenticate your emails. Although you can use Easy DKIM with any DNS provider, it is especially easy to set up when you manage your domain with Route 53.
- Control user access to your email sending by using AWS Identity and Access Management (IAM).
- Store emails you receive in Amazon Simple Storage Service (Amazon S3).
- Take action on your received emails by triggering AWS Lambda functions.
- Use AWS Key Management Service (AWS KMS) to optionally encrypt the mail you receive in your Amazon S3 bucket.
- Use AWS CloudTrail to log Amazon SES API calls that you make using the console or the Amazon SES API.
- Publish your email sending events to Amazon CloudWatch or Amazon Data Firehose. If you publish your email sending events to Firehose, you can access them in Amazon Redshift, Amazon OpenSearch Service, or Amazon S3.

Pricing

With Amazon SES, you pay based on the volume of emails sent and received. For more information, see Amazon SES pricing.

Regions and Amazon SES

SES is available in several AWS Regions around the world. In each region, AWS maintains multiple Availability Zones. These Availability Zones are physically isolated from each other, but are united by private, low-latency, high-throughput, and highly redundant network connections. These Availability Zones enable us to provide very high levels of availability and redundancy, while also minimizing latency.

For a list of all of the SES regional endpoints, see Amazon Simple Email Service endpoints and quotas in the *AWS General Reference*. To learn more about the number of Availability Zones that are available in each region, see AWS Global Infrastructure.

This section contains information that you need to know if you plan to use SES in multiple AWS Regions. It discusses the following subjects:

- SES regions and endpoints

- Sandbox removal and sending limit increases
- Verification of email addresses and domains
- Easy DKIM
- Account-level suppression list
- Feedback notifications
- SMTP credentials
- Sending authorization
- Feedback endpoints used for custom MAIL FROM domains
- Email receiving
- Setting up (MX) records

For general information about AWS Regions, see AWS service endpoints in the *AWS General Reference.*

SES regions and endpoints

When you use SES to send email, you connect to a URL that provides an endpoint for the SES API or SMTP interface. The *AWS General Reference* contains a complete list of endpoints that you use to send and receive email through SES. For more information, see Amazon Simple Email Service endpoints and quotas in the AWS General Reference—specific sections are referenced below:

- **API endpoints** – When you send email through SES, you can use the URLs listed in this table to make HTTPS requests to the SES API.
- **SMTP endpoints** – You can use the URLs listed in this table to send email when using the SMTP interface.
- **Email Receiving endpoints** – If you've configured SES to receive email that's sent to your domain, you can use the inbound SMTP endpoint URLs listed in this table when you set up the mail exchanger (MX) records in the DNS settings for your domain.

> **Note**
>
> The inbound SMTP URLs aren't IMAP server addresses. In other words, you can't use them to receive email by using an application such as Outlook. For a service that provides an IMAP server for incoming email, see Amazon WorkMail.

Sandbox removal and sending limit increases

The sandbox status for your account can differ between AWS Regions. In other words, if your account has been removed from the sandbox in the US West (Oregon) region, it might still be in the sandbox in the US East (N. Virginia) region, unless you've also had it removed from the sandbox in that region.

Sending limits can also be different depending on the AWS Region. For example, if your account is able to send 10 messages per second in the Europe (Ireland) region, you might be able to send more or fewer messages in other regions.

When you submit a request to have your account removed from the sandbox, or when you submit a request to have your account's sending quotas increased, be sure to choose all of the AWS Regions that your request applies to. You can submit several requests in a single Support Center case.

Verification of email addresses and domains

Before you can send email using SES, you have to verify that you own the email address or domain that you plan to send from. The verification status of email addresses and domains also differs across AWS Regions. For example, if you verify a domain in the US West (Oregon) region, you can't use that domain to send email in the US East (N. Virginia) region until you complete the verification process again for that region. For more information about verifying email addresses and domains, see Verified identities in Amazon SES.

Easy DKIM

You have to perform the Easy DKIM setup process for each AWS Region where you want to use Easy DKIM. That is, in each region, you have to use the SES console or the SES API to generate CNAME records. Next, you have to add all of the CNAME records to the DNS configuration for your domain. For more information about setting up Easy DKIM, see Easy DKIM in Amazon SES.

SES uses various DKIM hosted zones that might differ per AWS Region and email identity. Always use the `SigningHostedZone` value returned by the CreateEmailIdentity or GetEmailIdentity operation to construct your CNAME records. For more information, see Managing Easy DKIM and BYODKIM.

Account-level suppression list

Your SES account-level suppression list applies to your AWS account only in the current AWS Region. You can manually add or remove, individually or in bulk, addresses from your account-

level suppression list by using the SES API v2 or console. For more information about using your account-level suppression list, see Using the Amazon SES account-level suppression list.

Feedback notifications

There are two important points to note about setting up feedback notifications in multiple AWS Regions:

- Verified identity settings, such as whether you receive feedback by email or through SNS, only apply to the region that you set them in. For example, if you verify *user@example.com* in the US West (Oregon) and US East (N. Virginia) regions and you want to receive bounced emails via SNS notifications, you have to use the SES API or the SES console to set up SNS feedback notifications for *user@example.com* in both regions.
- SNS topics that you use for feedback forwarding have to be in the same region where you use SES.

For more information about monitoring your sending activity through feedback notifications, see Setting up event notifications for Amazon SES.

SMTP credentials

The credentials that you use to send email through the SES SMTP interface are unique to each AWS Region. If you use the SES SMTP interface to send email in more than one region, you have to generate a set of SMTP credentials for each region.

> **Note**
>
> If you created your SMTP credentials before January 10, 2019, your SMTP credentials were created using an older version of the AWS Signature. For security purposes, you should delete credentials that you created before this date, and replace them with newer credentials. You can delete older credentials by using the IAM console.

Feedback endpoints used for custom MAIL FROM domains

If you use a custom MAIL FROM domain, SES requires you to publish an MX record so that your domain can receive the bounce and complaint notifications that email providers send you. You can

use the same custom MAIL FROM domain for verified identities in different AWS Regions because the bounce and complaint notifications are sent to a region specific feedback endpoint.

When you configure a custom MAIL FROM domain, SES automatically specifies the correct feedback endpoint for the region where the custom MAIL FROM is being configured. This endpoint is provided in the MX record's value field for you to publish (add) to your domain's DNS configuration.

The custom MAIL FROM setup process is described in Using a custom MAIL FROM domain. For reference, the feedback endpoints that SES uses for the different AWS Regions is listed in the Feedback endpoints table in the AWS General Reference.

Sending authorization

Delegate senders can only send emails from the AWS Region where the identity owner's identity is verified. The sending authorization policy that gives permission to the delegate sender must be attached to the identity in that region. For more information about sending authorization, see Using sending authorization with Amazon SES.

Email receiving

With the exception of Amazon S3 buckets, all of the AWS resources that you use for receiving email with SES have to be in the same AWS Region as the SES endpoint. For example, if you use SES in the US West (Oregon) region, then any SNS topics, KMS keys, and Lambda functions that you use also have to be in the US West (Oregon) region. Similarly, to receive email with SES within a region, you have to create an active receipt rule set in that region. Email receiving concepts and setup process are explained in Email receiving with Amazon SES.

The Email Receiving endpoints table in the AWS General Reference lists the email receiving endpoints for all of the AWS Regions where SES supports email receiving.

Service quotas in Amazon SES

The following sections list and describe the quotas that apply to Amazon SES resources and operations. Some quotas can be increased, while others can't. To determine whether you can request an increase for a quota, refer to the **Adjustable** column.

> **Note**
>
> SES quotas are for each AWS Region that you use in your AWS account.

Email sending quotas

The following quotas apply to sending email through SES.

Sending quotas

Quotas are based on the number of recipients, rather than on the number of messages.

Resource	Default Quota	Adjustable
Number of emails that can be sent per 24-hour period	If your account is in the sandbox, you can send up to 200 emails per 24-hour period. If your account is out of the sandbox, this number varies based on your specific use case.	Yes
Number of emails that can be sent per second (*sending rate*)	If your account is in the sandbox, you can send 1 email per second. If your account is out of the sandbox, this rate varies based on your specific use case.	Yes

Message quotas

Resource	Default Quota	Adjustable
Using the SES v1 API - Maximum message size (including attachments)	10 MB per message (after base64 encoding).	No *(For workloads with message sizes in excess of 10MB, consider migrating to the SES v2 API.)*
Using the SES v2 API or SMTP - Maximum message size (including attachments)	40 MB per message (after base64 encoding).	No

> **Note**
>
> Messages larger than 10MB are subject to bandwidth throttling, and depending on your sending rate, you may be throttled to as low as 40MB/s. For example, you could send a 40MB message at the rate of 1 message per second, or two 20MB messages per second.

Sender and recipient quotas

Resource	Default Quota	Adjustable
Maximum number of tenants	10,000	Yes
Maximum number of recipients per message	50 recipients per message. **Note** A recipient is any "To", "CC", or "BCC" address.	No.
Maximum number of identities that you can verify	10,000 identities per AWS Region.	Please contact your AWS Account Manager to discuss your use case.

Resource	Default Quota	Adjustable
	Note An *identity* is a domain or email address that you use to send email through SES.	
Maximum number of dedicated IP pools (inclusive of both managed and standard IP pools)	50	No

Quotas related to event publishing

Resource	Default Quota	Adjustable
Maximum number of configuration sets	10,000	No
Maximum length of configuration set name	Configuration set names can contain up to 64 alphanumeric characters. They can also contain hyphens (-) and underscores (_). Names can't contain spaces, accented characters, or any other special characters.	No
Maximum number of event destinations per configuration set	10	No

Resource	Default Quota	Adjustable
Maximum number of dimensions per CloudWatch event destination	10	No

Email template quotas

Resource	Default Quota	Adjustable
Maximum number of email templates in each AWS Region	20,000	No
Maximum template size	500 KB	No
Maximum number of replacement values in each template	Unlimited	N/A
Maximum number of recipients for each templated email	50 destinations. A *destination* is any email address on the "To", "CC", or "BCC" lines. **Note** The number of destinations you can contact in a single call to the API may be limited by your account's maximum sending rate.	No

Email receiving quotas

The following table lists the quotas associated with receiving email through SES.

Resource	Default Quota	Adjustable
Maximum number of rules per receipt rule set	200	No
Maximum number of actions per receipt rule	10	No
Maximum number of recipients per receipt rule	500	No
Maximum number of receipt rule sets per AWS account	40	No
Maximum number of IP address filters per AWS account	100	No
Maximum email size (including headers) that can be stored in an Amazon S3 bucket	40 MB	No
Maximum email size (including headers) that can be published using an Amazon SNS notification	150 KB	No
Maximum email headers size that can be published using an Amazon SNS notification	10 KB	No
Maximum email headers size that can be published using an AWS Lambda function	50 KB	No

Mail Manager quotas

The following table lists the quotas associated with Mail Manager.

Resource	Default Quota	Adjustable
Maximum number of open ingress endpoints	10	No
Maximum number of authorized ingress endpoints	50	No
Maximum number of mutual TLS ingress endpoints	10	No
Maximum number of recipients per message	100	No
Maximum email size (including headers)	40 MB	No
Maximum number of traffic policy statements	20	No
Maximum number of traffic policy statement conditions	10	No
Maximum number of traffic policies per region	100	No
Maximum number of SMTP relays	40	No
Maximum number of Address Lists per region	100	No
Maximum number of addresses per Address List	100,000	No
Maximum number of rule sets	40	No

Resource	Default Quota	Adjustable
Maximum number of rules per rule set	40	No
Maximum number of conditions per rule	10	No
Maximum number of actions per rule	10	No
Maximum number of relay or send actions per rule set	10	No
Maximum number of *active* archives	10	No
Maximum number of archive search results	1000	No
Maximum number of exported archive search results	250,000	No
Maximum number of running search requests in parallel	1	No
Maximum number of running export requests in parallel	1	No
Maximum number of retention changes for archive per week	1	No

General quotas

The following table lists quotas that apply to both sending and receiving email through SES.

SES API sending quotas

Resource	Default Quota	Adjustable
Rate at which you can call Amazon SES API actions	All actions (except for `SendEmail` , `SendRawEmail` , and `SendTemplatedEmail`) are throttled at one request per second.	No
MIME parts	500	No

SES miscellaneous quotas

Resource	Default Quota	Adjustable
Maximum number of concurrent import jobs	20	No
Maximum number of concurrent export jobs	20	No

Types of Amazon SES credentials

To interact with Amazon Simple Email Service (Amazon SES), you use security credentials to verify who you are and whether you have permission to interact with Amazon SES. There are different types of credentials, and the credentials you use depend on what you want to do. For example, you use AWS access keys when you send an email using the Amazon SES API, and SMTP credentials when you send an email using the Amazon SES SMTP interface.

The following table lists the types of credentials you might use with Amazon SES, depending on what you are doing.

If you want to access the...	Use these credentials	What the credentials consist of	How to get the credentials
Amazon SES API (You might access the Amazon SES API directly, or indirectly through an AWS SDK, the AWS Command Line Interface, or the AWS Tools for Windows PowerShell.)	AWS access keys	Access key ID and secret access key	See Access Keys in the *AWS General Reference*. **Note** For security best practice, use AWS Identity and Access Management (IAM) user access keys instead of AWS account access keys. Your AWS account credentials grant full access to all your AWS resources, so you should store them in a safe place and instead use IAM user credentials for day-to-day interaction with AWS. For more information, see Root Account Credentials vs. IAM User Credentials in the *AWS General Reference*.
Amazon SES SMTP interface	SMTP credentials	User name and password	See Obtaining Amazon SES SMTP credentials.

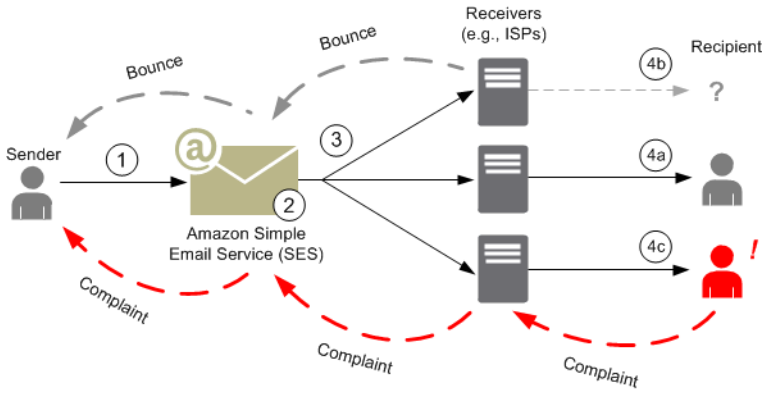
If you want to access the...	Use these credentials	What the credentials consist of	How to get the credentials
			Note Although your Amazon SES SMTP credentials are different than your AWS access keys and IAM user access keys, Amazon SES SMTP credentials are actually a type of IAM credentials. An IAM user can create Amazon SES SMTP credentials, but the root account owner must ensure that the IAM user's policy gives them permission to access the following IAM actions: "iam:ListUsers", "iam:CreateUser", "iam:CreateAccessKey", and "iam:PutUserPolicy".

If you want to access the...	Use these credentials	What the credentials consist of	How to get the credentials
Amazon SES console	IAM user name and password OR Email address and password	IAM user name and password OR Email address and password	See IAM User Name and Password and Email Address and Password of the *AWS General Reference*. **Note** For security best practice, use an IAM user name and password instead of an email address and password. The email address and password combination are for your AWS account, so you should store them in a safe place instead of using them for day-to-day interaction with AWS. For more information, see Root Account Credentials vs. IAM User Credentials in the *AWS General Reference*.

For more information about different types of AWS security credentials (except for SMTP credentials, which are used only for Amazon SES), see AWS Security Credentials in the *AWS General Reference*.

How email sending works in Amazon SES

This topic describes what happens when you send an email with SES, and the various outcomes that can occur after the email is sent. The following figure is a high-level overview of the sending process:



1. A client application, acting as an email sender, makes a request to SES to send email to one or more recipients.
2. If the request is valid, SES accepts the email.
3. SES sends the message over the Internet to the recipient's receiver. Once the message is passed to SES, it is usually sent immediately, with the first delivery attempt normally occurring within milliseconds.
4. At this point, there are different possibilities. For example:
    a. The ISP successfully delivers the message to the recipient's inbox.
    b. The recipient's email address does not exist, so the ISP sends a bounce notification to SES. SES then forwards the notification to the sender.
    c. The recipient receives the message but considers it to be spam and registers a complaint with the ISP. The ISP, which has a feedback loop set up with SES, sends the complaint to SES, which then forwards it to the sender.

The following sections review the individual possible outcomes after a sender sends an email request to SES and after SES sends an email message to the recipient.

After a sender sends an email request to SES

When the sender makes a request to SES to send an email, the call may succeed or fail. The following sections describe what happens in each case.

Successful sending request

If the request to SES succeeds, SES returns a success response to the sender. This message includes the *message ID*, a string of characters that uniquely identifies the request. You can use the message ID to identify the sent email or to track problems encountered during sending (you must store your own mapping between an identifier and the SES message ID that SES passes back to you when it accepts the email). SES then assembles an email message based on the request parameters, scans the message for questionable content and viruses and then sends it out over the Internet using Simple Mail Transfer Protocol (SMTP). Your message is usually sent immediately; the first delivery attempt typically occurs within milliseconds.

> **Note**
>
> If SES accepts the sender's request and then determines that the message contains a virus, SES stops processing the message and doesn't attempt to deliver it to the recipient's mail server.

Failed sending request

If the sender's email-sending request to SES fails, SES responds to the sender with an error and drops the email. The request could fail for several reasons. For example, the request may not be formatted properly or the email address may not have been verified by the sender.

> **Note**
>
> On rare occasions, SES may accept an email for delivery even though the send request returns an error to the caller. If your application retries the request, a second email with a different message ID may be sent, and you may receive event notifications for the message associated with the failed call. To reliably correlate event notifications with your sending logic, use message tags rather than relying solely on the message ID.

The method through which you can determine if the request has failed depends on how you call SES. The following are examples of how errors and exceptions are returned:

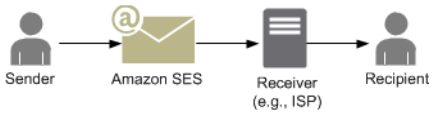
- If you are calling SES through the Query (HTTPS) API (`SendEmail` or `SendRawEmail`), the actions will return an error. For more information, see the Amazon Simple Email Service API Reference.
- If you are using an AWS SDK for a programming language that uses exceptions, the call to SES will throw a *MessageRejectedException*. (The name of the exception may vary slightly depending on the SDK.)
- If you are using the SMTP interface, then the sender receives an SMTP response code, but how the error is conveyed depends on the sender's client. Some clients may display an error code; others may not.

For information about errors that can occur when you send an email with SES, see Amazon SES email sending errors.

After Amazon SES sends an email

If the sender's request to SES succeeds, then SES sends the email and one of the following outcomes occurs:

- **Successful delivery and the recipient does not object to the email –** The email is accepted by the ISP, and the ISP delivers the email to the recipient. A successful delivery is shown in the following figure.



- **Hard bounce –** The email is rejected by the ISP because of a persistent condition or rejected by SES because the email address is on the SES suppression list. An email address is on the SES suppression list if it has recently caused a hard bounce for any SES customer. A hard bounce with an ISP can occur because the recipient's address is invalid. A hard bounce notification is sent from the ISP back to SES, which notifies the sender through email or through Amazon Simple Notification Service (Amazon SNS), depending on the sender's setup. SES notifies the sender of suppression list bounces by the same means. The path of a hard bounce from an ISP is shown in the following figure.

- **Soft bounce –** The ISP cannot deliver the email to the recipient because of a temporary condition, such as the ISP is too busy to handle the request or the recipient's mailbox is full. A soft bounce can also occur if the domain does not exist. The ISP sends a soft bounce notification back to SES, or, in the case of a nonexistent domain, SES cannot find an email server for the domain. In either case, SES retries the email for an extended period of time. If SES cannot deliver the email in that time period, it sends you a bounce notification through email or through Amazon SNS. If SES can deliver the email to the recipient during a retry, the delivery is successful. A soft bounce is shown in the following figure. In this case, SES retries sending the email, and the ISP is eventually able to deliver it to the recipient.

- **Complaint –** The email is accepted by the ISP and delivered to the recipient, but the recipient considers the email to be spam and clicks a button such as "Mark as spam" in his or her email client. If SES has a feedback loop set up with the ISP, then a complaint notification is sent to SES, which forwards the complaint notification to the sender. Most ISPs do not provide the email address of the recipient who submitted the complaint, so the complaint notification from SES provides the sender a list of recipients who might have sent the complaint, based on the recipients of the original message and the ISP from which SES received the complaint. The path of a complaint is shown in the following figure.

- **Auto response –** The email is accepted by the ISP, and the ISP delivers it to the recipient. The ISP then sends an automatic response such as an out-of-the-office (OOTO) message to SES. SES forwards the auto response notification to the sender. An auto response is shown in the following figure.

Make sure that your SES-enabled program does not retry sending messages that generate an auto response.

> **Tip**
>
> You can use the SES mailbox simulator to test a successful delivery, bounce, complaint, OOTO, or what happens when an address is on the suppression list. For more information, see Using the mailbox simulator manually.

Email format in Amazon SES

When a client makes a request to Amazon SES, Amazon SES constructs an email message compliant with the Internet Message Format specification (RFC 5322). An email consists of a *header*, a *body*, and an *envelope*, as described below.

- **Header—**Contains routing instructions and information about the message. Examples are the sender's address, the recipient's address, the subject, and the date. The header is analogous to the information at the top of a postal letter, though it can contain many other types of information, such as the format of the message.
- **Body—**Contains the text of the message itself.
- **Envelope—**Contains the actual routing information that is communicated between the email client and the mail server during the SMTP session. This email envelope information is analogous to the information on a postal envelope. The routing information of the email envelope is usually the same as the routing information in the email header, but not always. For example, when you send a blind carbon copy (BCC), the actual recipient address (derived from the envelope) is not the same as the "To" address that is displayed in the recipient's email client, which is derived from the header.

The following is a simple example of an email. The header is followed by a blank line and then the body of the email. The envelope isn't shown because it is communicated between the client and the mail server during the SMTP session, rather than a part of the email itself.

```
Received: from abc.smtp-out.amazonses.com (123.45.67.89) by in.example.com
 (87.65.43.210); Fri, 17 Dec 2010 14:26:22
From: "Andrew" <andrew@example.com>;
```

```
To: "Bob" <bob@example.com>
Date: Fri, 17 Dec 2010 14:26:21 -0800
Subject: Hello
Message-ID: <61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>
Accept-Language: en-US
Content-Language: en-US
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: quoted-printable
MIME-Version: 1.0

Hello, I hope you are having a good day.

-Andrew
```

The following sections review email headers and bodies and identify the information that you need to provide when you use Amazon SES.

Email header

There is one header per email message. Each line of the header contains a field followed by a colon followed by a field body. When you read an email in an email client, the email client typically displays the values of the following header fields:

- **To—**The email addresses of the message's recipients.
- **CC—**The email addresses of the message's carbon copy recipients.
- **From—**The email address from which the email is sent.
- **Subject—**A summary of the message topic.
- **Date—**The time and date the email is sent.

There are many additional header fields that provide routing information and describe the content of the message. Email clients typically do not display these fields to the user. For a full list of the header fields that Amazon SES accepts, see Amazon SES header fields. When you use Amazon SES, you particularly need to understand the difference between "From," "Reply-To," and "Return-Path" header fields. As noted previously, the "From" address is the email address of the message sender, whereas "Reply-To" and "Return-Path" are as follows:

- **Reply-To—**The email address to which replies will be sent. By default, replies are sent to the original sender's email address.

- **Return-Path—**The email address to which message bounces and complaints should be sent. "Return-Path" is sometimes called "envelope from," "envelope sender," or "MAIL FROM."

> **Note**
>
> When you use Amazon SES, we recommend that you always set the "Return-Path" parameter so that you can be aware of bounces and take corrective action if they occur.

To easily match a bounced message with its intended recipient, you can use Variable Envelope Return Path (VERP). With VERP, you set a different "Return-Path" for each recipient, so that if the message bounces back, you automatically know which recipient it bounced from, rather than having to open the bounce message and parse it.

Email body

The email body contains the text of the message. The body can be sent in the following formats:

- **HTML—**If the recipient's email client can interpret HTML, the body can include formatted text and hyperlinks
- **Plain text—**If the recipient's email client is text-based, the body must not contain any nonprintable characters.
- **Both HTML and plain text—**When you use both formats to send the same content in a single message, the recipient's email client decides which to display, based upon its capabilities.

If you are sending an email message to a large number of recipients, then it makes sense to send it in both HTML and text. Some recipients will have HTML-enabled email clients, so that they can click embedded hyperlinks in the message. Recipients using text-based email clients will need you to include URLs that they can copy and open using a web browser.

Email information you need to provide to Amazon SES

When you send an email with Amazon SES, the email information you need to provide depends on how you call Amazon SES. You can provide a minimal amount of information and have Amazon SES take care of all of the formatting for you. Or, if you want to do something more advanced like send an attachment, you can provide the raw message yourself. The following sections review what you need to provide when you send an email by using the Amazon SES API, the Amazon SES SMTP interface, or the Amazon SES console.

Amazon SES API

If you call the Amazon SES API directly, you call either the `SendEmail` or the `SendRawEmail` API. The amount of information you need to provide depends on which API you call.

- The `SendEmail API` requires you to provide only a source address, destination address, message subject, and a message body. You can optionally provide "Reply-To" addresses. When you call this API, Amazon SES automatically assembles a properly formatted multi-part Multipurpose Internet Mail Extensions (MIME) email message optimized for display by email client software. For more information, see Sending formatted email using the Amazon SES API.
- The `SendRawEmail` API provides you the flexibility to format and send your own raw email message by specifying headers, MIME parts, and content types. `SendRawEmail` is typically used by advanced users. You need to provide the body of the message and all header fields that are specified as required in the Internet Message Format specification (RFC 5322). For more information, see Sending raw email using the Amazon SES API v2.

If you use an AWS SDK to call the Amazon SES API, you provide the information listed above to the corresponding functions (for example, `SendEmail` and `SendRawEmail` for Java).

For more information about sending email using the Amazon SES API, see Using the Amazon SES API to send email.

Amazon SES SMTP interface

When you access Amazon SES through the SMTP interface, your SMTP client application assembles the message, so the information you need to provide depends on the application you are using. At a minimum, the SMTP exchange between a client and a server requires a source address, a destination address, and message data.

For more information about sending email using the Amazon SES SMTP interface, see Using the Amazon SES SMTP interface to send email.

Amazon SES console

When you send an email by using the Amazon SES console, the amount of information you need to provide depends on whether you choose to send a formatted or raw email.

- To send a formatted email, you need to provide a source address, a destination address, a message subject, and a message body. Amazon SES automatically assembles a properly

formatted multi-part MIME email message optimized for display by email client software. You can also specify a reply-to and a return path field.

- To send a raw email, you provide the source address, a destination address, and the message content, which must contain the body of the message and all header fields that are specified as required in the Internet Message Format specification (RFC 5322).

Understanding email deliverability in Amazon SES

You want your recipients to read your emails, find them valuable, and not label them as spam. In other words, you want to maximize email *deliverability*—the percentage of your emails that arrive in your recipients' inboxes. This topic reviews email deliverability concepts that you should be familiar with when you use Amazon SES.

To maximize email deliverability, you need to understand email delivery issues, proactively take steps to prevent them, stay informed of the status of the emails that you send, and then improve your email-sending program, if necessary, to further increase the likelihood of successful deliveries. The following sections review the concepts behind these steps and how Amazon SES helps you through the process.

Understand email delivery issues

In most cases, your messages are delivered successfully to recipients who expect them. In some cases, however, a delivery might fail, or a recipient might not want to receive the mail that you are sending. Bounces, complaints, and the suppression list are related to these delivery issues and are described in the following sections.

Bounce

If your recipient's receiver (for example, an email provider) fails to deliver your message to the recipient, the receiver bounces the message back to Amazon SES. Amazon SES then notifies you of the bounced email through email or through Amazon Simple Notification Service (Amazon SNS), depending on how you have your system set up. For more information, see Setting up event notifications for Amazon SES.

There are *hard bounces* and *soft bounces*, as follows:

- **Hard bounce –** A persistent email delivery failure. For example, the mailbox does not exist. Amazon SES does not retry hard bounces, with the exception of DNS lookup failures. We strongly recommend that you do not make repeated delivery attempts to email addresses that hard bounce.
- **Soft bounce –** A temporary email delivery failure. For example, the mailbox is full, there are too many connections (also called *throttling*), or the connection times out. Amazon SES retries soft bounces multiple times. If the email still cannot be delivered, then Amazon SES stops retrying it.

Amazon SES notifies you of hard bounces and soft bounces that will no longer be retried. However, only hard bounces count toward your bounce rate and the bounce metric that you retrieve using the Amazon SES console or the `GetSendStatistics` API.

Bounces can also be *synchronous* or *asynchronous*. A synchronous bounce occurs while the email servers of the sender and receiver are actively communicating. An asynchronous bounce occurs when a receiver initially accepts an email message for delivery and then subsequently fails to deliver it to the recipient.

Complaint

Most email client programs provide a button labeled "Mark as Spam," or similar, which moves the message to a spam folder, and forwards it to the email provider. Additionally, most email providers maintain an abuse address (e.g., abuse@example.net), where users can forward unwanted email messages and request that the email provider take action to prevent them. In both of these cases, the recipient is making a complaint. If the email provider concludes that you are a spammer, and Amazon SES has a feedback loop set up with the email provider, then the email provider will send the complaint back to Amazon SES. When Amazon SES receives such a complaint, it forwards the complaint to you either by email or by using an Amazon SNS notification, depending on how you have your system set up. For more information, see Setting up event notifications for Amazon SES. We recommend that you do not make repeated delivery attempts to email addresses that generate complaints.

Global suppression list

The Amazon SES *global suppression list*, owned and managed by SES to protect the reputation of addresses in the SES shared IP pool, contains recipient email addresses that have recently caused a hard bounce for any SES customer. If you try to send an email through SES to an address that is on the suppression list, the call to SES succeeds, but SES treats the email as a hard bounce instead of attempting to send it. Like any hard bounce, suppression list bounces count towards your sending

quota and your bounce rate. An email address can remain on the suppression list for up to 14 days. If you're sure that the email address that you're trying to send to is valid, you can override the global suppression list by making sure the address isn't listed in your account-level suppression list and SES will still attempt delivery, but if it bounces, the bounce will affect your own reputation, but no one else will get bounces because they can't send to that email address if they aren't using their own account-level suppression list. To understand more about the account-level suppression list, see Using the Amazon SES account-level suppression list.

Be proactive

One of the biggest issues with email on the Internet is unsolicited bulk email (spam). Email providers take extensive measures to prevent their customers from receiving spam. Amazon SES also takes steps to decrease the likelihood that email providers consider your email to be spam. Amazon SES uses verification, authentication, sending quotas, and content filtering. Amazon SES also maintains a trusted reputation with email providers and requires you to send high-quality email. Amazon SES does some of those things for you automatically (for example, content filtering); in other cases, it provides the tools (such as authentication), or guides you in the right direction (sending quotas). The following sections provide more information about each concept.

Verification

Unfortunately, it's possible for a spammer to falsify an email header and spoof the originating email address so that it appears as though the email originated from a different source. To maintain trust between email providers and Amazon SES, Amazon SES needs to ensure that its senders are who they say they are. You are therefore required to verify all email addresses from which you send emails through Amazon SES to protect your sending identity. You can verify email addresses by using the Amazon SES console or by using the Amazon SES API. You can also verify entire domains. For more information, see Creating an email address identity and Creating a domain identity.

If your account is still in the Amazon SES sandbox, you also need to verify all recipient addresses except for addresses provided by the Amazon SES mailbox simulator. For information about getting out of the sandbox, see Request production access (Moving out of the Amazon SES sandbox). For more information about the mailbox simulator, see Using the mailbox simulator manually.

Authentication

Authentication is another way that you can indicate to email providers that you are who you say you are. When you authenticate an email, you provide evidence that you are the owner

of the account and that your emails have not been modified in transit. In some cases, email providers refuse to forward email that is not authenticated. Amazon SES supports two methods of authentication: Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM). For more information, see Configuring identities in Amazon SES.

Sending quotas

If an email provider detects sudden, unexpected spikes in the volume or rate of your emails, the email provider might suspect you are a spammer and block your emails. Therefore, every Amazon SES account has a set of sending quotas. These quotas restrict the number of emails that you can send in a 24-hour period, and the number that you can send per second. These sending quotas help protect your trustworthiness with email providers.

In most cases, if you're a brand-new user, Amazon SES lets you send a small amount of email each day. If the mail that you send is acceptable to email providers, we automatically increase this quota. Your sending quotas steadily increase over time so that you can send larger quantities of email at faster rates. You can also create an SES Sending Limits Increase case to request additional quota increases.

For more information about sending quotas and how to increase them, see Managing your Amazon SES sending limits.

Content filtering

Many email providers use content filtering to determine if incoming emails are spam. Content filters look for questionable content and block the email if the email fits the profile of spam. Amazon SES uses content filters also. When your application sends a request to Amazon SES, Amazon SES assembles an email message on your behalf and then scans the message header and body to determine if they contain content that email providers might consider spam. If your messages look like spam to the content filters that Amazon SES uses, your reputation with Amazon SES will be negatively affected.

Amazon SES also scans all messages for viruses. If a message contains a virus, Amazon SES doesn't attempt to deliver the message to the recipient's mail server.

Reputation

When it comes to email sending, *reputation*—a measure of confidence that an IP address, email address, or sending domain is not the source of spam—is important. Amazon SES maintains a strong reputation with email providers so that they deliver your email to your recipients' inboxes.

Similarly, you need to maintain a trusted reputation with Amazon SES. You build your reputation with Amazon SES by sending high-quality content. When you send high-quality content, your reputation becomes more trusted over time and Amazon SES increases your sending quotas. Excessive bounces and complaints negatively impact your reputation and can cause Amazon SES to reduce the sending quotas for your account, or terminate your Amazon SES account.

One way to help maintain your reputation is to use the mailbox simulator when you test your system, instead of sending to email addresses that you have created yourself. Emails to the mailbox simulator do not count toward your bounce and complaint metrics. For more information about the mailbox simulator, see Using the mailbox simulator manually.

High-quality email

High-quality email is email that recipients find valuable and want to receive. Value means different things to different recipients and can come in the form of offers, order confirmations, receipts, newsletters, etc. Ultimately, your deliverability rests on the quality of the emails that you send because email providers block emails that they consider to be low quality.

Stay informed

Whether your deliveries fail, your recipients complain about your emails, or Amazon SES successfully delivers an email to a recipient's mail server, Amazon SES helps you to track down the issue by providing notifications and by enabling you to easily monitor your usage statistics.

Notifications

When an email bounces, the email provider notifies Amazon SES, and Amazon SES notifies you. Amazon SES notifies you of hard bounces and soft bounces that Amazon SES will no longer retry. Many email providers also forward complaints, and Amazon SES sets up complaint feedback loops with the major email providers so you don't have to. Amazon SES can notify you of bounces, complaints, and successful deliveries in two ways: you can set your account up to receive notifications through Amazon SNS, or you can receive notifications by email (bounces and complaints only). For more information, see Setting up event notifications for Amazon SES.

Usage statistics

Amazon SES provides usage statistics so that you can view your failed deliveries to determine and resolve the root causes. You can view your usage statistics by using the Amazon SES console or by calling the Amazon SES API. You can view how many deliveries, bounces, complaints, and virus-

infected rejected emails you have, and you can also view your sending quotas to ensure that you stay within them.

Improve your email-sending program

If you are getting large numbers of bounces and complaints, it's time to reassess your email-sending strategy. Remember that excessive bounces, complaints, and attempts to send low-quality email constitute abuse and put your AWS account at risk of termination. Ultimately, you need to be sure that you use Amazon SES to send high-quality emails and to only send emails to recipients who want to receive them.

At-least-once delivery

Amazon SES stores copies of your messages on multiple servers for redundancy and high availability. On rare occasions, one of the servers that stores a copy of a message might be unavailable when you receive or delete a message.

If this occurs, the copy of the message isn't deleted on that unavailable server, and you might get that message copy again when you receive messages. Design your applications to be idempotent (they should not be affected adversely when processing the same message more than once).

Best practices for sending email using Amazon SES

The way you manage email communications with your customers is referred to as your *email program*. There are several factors that can lead to the success or failure of your email program; these factors may seem confusing or mysterious at first. However, by understanding how email is delivered, and by following certain best practices, you can increase the chances of your email successfully reaching your customers' inboxes.

Topics

- Email program success metrics
- Maintaining a positive sender reputation

Email program success metrics

There are several metrics that help measure the success of your email program.

This section provides information about the following metrics:

- Bounces

- Complaints
- Message quality

Bounces

A *bounce* occurs when an email cannot be delivered to the intended recipient. There are two types of bounces: *hard bounces* and *soft bounces*. A hard bounce occurs when the email cannot be delivered because of a persistent issue, such as when an email address doesn't exist. A soft bounce occurs when a temporary issue prevents the delivery of an email. Soft bounces can occur when a recipient's inbox is full, or when the receiving server is temporarily unavailable. Amazon SES handles soft bounces by attempting to re-deliver soft bounced emails for a certain period of time.

It's essential that you monitor the number of hard bounces in your email program, and that you remove hard-bouncing email addresses from your recipient lists. When email receivers detect a high rate of hard bounces, they assume that you don't know your recipients well. As a result, a high hard bounce rate can negatively impact the deliverability of your email messages.

The following guidelines can help you avoid bounces and improve your sender reputation:

- Try to keep your hard bounce rate below 5%. The fewer hard bounces in your email program, the more likely ISPs will see your messages as legitimate and valuable. This rate should be considered a reasonable and attainable goal, but isn't a universal rule across all ISPs.
- Never rent or buy email lists. These lists may contain large numbers of invalid addresses, which could cause your hard bounce rates to increase dramatically. Furthermore, these lists could contain spam traps—email addresses specifically used to catch illegitimate senders. If your messages land in a spam trap, your delivery rates and sender reputation could be irrevocably damaged.
- Keep your list up to date. If you haven't emailed your recipients in a long time, try to validate your customers' statuses through some other means (such as website login activity or purchase history).
- If you don't have a method of verifying your customers' statuses, consider sending a *win-back* email. A typical win-back email mentions that you haven't heard from the customer in a while, and encourages the customer to confirm that they still want to receive your email. After sending a win-back email, purge all of the recipients who did not respond from your lists.

When you receive bounces, it's vital that you respond to them appropriately by observing the following rules:

- If an email address hard bounces, immediately remove that address from your lists. Do not attempt to re-send messages to hard-bouncing addresses. Repeated hard bounces add up, and ultimately harm your reputation with the recipient's ISP.
- Make sure that the address you use to receive bounce notifications is able to receive email. For more information about setting up bounce and complaint notifications, see Setting up event notifications for Amazon SES.
- If your inbound email comes to you from an ISP, instead of through your own internal servers, an influx of bounce notifications can land in your spam folder or be dropped completely. Ideally, you should not use a hosted email address to receive bounces. If you must, however, then check the spam folder often, and don't mark the bounce messages as spam. In Amazon SES, you can specify the address that bounce notifications are sent to.
- Usually, a bounce provides the address of the mailbox refusing delivery. However, if you need more granular data to map a recipient address to a particular email campaign, include an X-header with a value you can trace back to your internal tracking system. For more information, see Amazon SES header fields.

Complaints

A complaint occurs when an email recipient clicks the "Mark as Spam" (or equivalent) button in their web-based email client. If you accumulate a large number of these complaints, the ISP assumes that you are sending spam. This has a negative impact on your deliverability rate and sender reputation. Some, but not all, ISPs will notify you when a complaint is reported; this is known as a *feedback loop*. Amazon SES automatically forwards complaints from ISPs that offer feedback loops to you.

The following guidelines can help you avoid complaints and improve your sender reputation:

- Try to keep your complaint rate below 0.1%. The fewer complaints in your email program, the more likely ISPs will see your messages as legitimate and valuable. This rate should be considered a reasonable and attainable goal, but isn't a universal rule across all ISPs.
- If a customer complains about a marketing email, you should immediately stop sending that customer marketing emails. However, if your email program also includes other types of emails (such as notification or transactional emails), it may be acceptable to continue to send those types of messages to the recipient who issued the complaint.

- As with hard bounces, if you have a list that you haven't sent email to in a while, ensure that your recipients understand why they're receiving your messages. We recommend that you send a welcome message reminding them of who you are and why you're contacting them.

When you receive complaints, it's vital that you respond to them appropriately by observing the following rules:

- Make sure that the address you use to receive complaint notifications is able to receive email. For more information about setting up bounce and complaint notifications, see Setting up event notifications for Amazon SES.
- Make sure that your complaint notifications aren't being marked as spam by your ISP or mail system.
- Complaint notifications usually contain the body of the email; this is different from bounce notifications, which only include the email headers. However, in complaint notifications, the email address of the individual who issued the complaint is removed. Use custom X-headers or special identifiers embedded in the email body so that you can identify the email address that issued the complaint. This technique makes it easier to identify addresses that complained so that you can remove them from your recipient lists.

Message quality

Email receivers use *content filters* to detect certain attributes in your messages to identify whether your message is legitimate. These content filters automatically review the content of your messages to identify common traits of unwanted to malicious messages. Amazon SES uses content filtering technologies to help detect and block messages that contain malware before they are sent.

If an email receiver's content filters determine that your message contains the characteristics of spam or malicious email, your message will most likely be flagged and diverted from recipients' inboxes.

Remember the following when designing your email:

- Modern content filters are intelligent, continuously adapting and changing. They don't rely on a predefined set of rules. Third-party services such as ReturnPath or Litmus can help identify content in your email that may trigger content filters.

- If your email contains links, check the URLs for those links against DNS-based Blackhole Lists (DNSBLs), such as those found at URIBL.com and SURBL.org.
- Avoid using link shorteners. Malicious senders may use link shorteners to hide the actual destination of a link. When ISPs notice that link shortening services—even the most reputable ones—are being used for nefarious purposes, they may deny access to those services altogether. If your email contains a link to a link shortening service that has been added to a deny list, it won't reach your customers' inboxes, and the success of your email campaign suffers.
- Test every link in your email to ensure that it points to the intended page.
- Make sure your website includes Privacy Policy and Terms of Use documents, and that these documents are up to date. It's a good practice to link to these documents from each email you send. Providing links to these documents demonstrates that you have nothing to hide from your customers, which can help build a relationship of trust.
- If you plan to send high-frequency content (such as "daily deals" messages), ensure that the content of your email is different with each deployment. When you send messages with high frequency, you must ensure that those messages are timely and relevant, rather than repetitive and annoying.

Maintaining a positive sender reputation

In Amazon SES, sender reputation refers to the credibility and trustworthiness of the email sender as perceived by email providers and spam filters. It is a measure of how likely your emails are to be considered legitimate and delivered successfully to the recipients' inboxes.

The following sections introduce the core email sending principals you must pay attention to in order to ensure that your email communications reach your intended audience while maintaining a good sender reputation.

Domain and "From" address considerations

- Think carefully about the addresses you send email from. The "From" address is one of the first pieces of information your recipients see, and therefore can leave a lasting first impression. Additionally, some ISPs associate your reputation with your "From" address.
- Consider using subdomains for different types of communications. For example, assume you are sending email from the domain *example.com*, and you plan to send both marketing and transactional messages. Rather than sending all of your messages from *example.com*, send your marketing messages from a subdomain such as *marketing.example.com*, and your transactional

messages from a subdomain such as *orders.example.com*. Unique subdomains develop their own reputations. Using subdomains reduces the risk of damage to your reputation if, for example, your marketing communications land in a spam trap or trigger a content filter.

- If you plan to send a large number of messages, don't send those messages from an ISP-based address such as *sender@hotmail.com*. If an ISP notices a large volume of messages coming from *sender@hotmail.com*, that email is treated differently than an email that comes from an outbound email sending domain that you own.
- Work with your domain registrar to ensure that the WHOIS information for your domain is accurate. Maintaining an honest and up-to-date WHOIS record demonstrates that you value transparency, and allows users to quickly identify whether or not your domain is legitimate.
- Avoid using a *no-reply* address, such as *no-reply@example.com*, as your "From" or "Reply-to" address. Using a *no-reply@* email address sends your recipients a clear message: that you aren't offering them a way to contact you, and that you're not interested in their feedback.

Authentication

- Authenticate your domain with SPF and SenderID. These authentication methods confirm to email recipients that each email you send is actually from the domain it claims to be from.
- Sign your outbound mail with DKIM. This step confirms to recipients that the content has not been changed in transit between sender and receiver.
- You can test your authentication settings for both SPF and DKIM by sending an email to an ISP-based email address that you own, such as a personal Gmail or Hotmail account, and then viewing the message's headers. The headers indicate whether your attempts to authenticate and sign the message were successful.

Building and maintaining your lists

- Implement a double opt-in strategy. When users sign up to receive email from you, send them a message with a confirmation link, and do not start sending them email until they confirm their address by clicking that link. A double opt-in strategy helps reduce the number of hard bounces resulting from typographical errors.
- When collecting email addresses with a web-based form, perform minimal validation on those addresses upon submission. For example, ensure that the addresses you collect are well-formed (that is, they are in the format *recipient@example.com*), and that they refer to domains with valid MX records.

- Use caution when allowing user-defined input to be passed to Amazon SES unchecked. Forums registrations and form submissions present unique risks because the content is completely user-generated, and spammers can fill out forms with their own content. It's your responsibility to ensure that you only send email with high-quality content.
- It is highly unlikely that a standard alias (such as *postmaster@*, *abuse@*, or *noc@*) will ever sign up for your email intentionally. Ensure that you are only sending messages to real people who actually want to receive them. This rule is especially true for standard aliases, which are customarily reserved for email watchdogs. These aliases can be maliciously added to your list as a form of sabotage, in order to damage your reputation.

Compliance

- Be aware of the email marketing and anti-spam laws and regulations in the countries and regions you send email to. You're responsible for ensuring that the email you send complies with these laws. This guide doesn't cover these laws, so it's important that you research them. For a list of laws, see Email Spam Legislation by Country on Wikipedia.
- Always consult an attorney to obtain legal advice.

Using Amazon SES with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

For examples specific to Amazon SES, see Code examples for Amazon SES using AWS SDKs.

Agent setup guide

Amazon SES publishes the `amazon-ses` skill to help AI coding agents set up and troubleshoot email sending. The AWS MCP Server can provide the current skill when your request needs it.

The current `amazon-ses` skill does not include Mail Manager or email receiving.

Topics

- Before you begin
- Step 1: Connect your agent to AWS
- Step 2: Use the Amazon SES skill
- Keep your setup current

Before you begin

Prerequisites depend on the setup method that you choose.

- For the AWS Command Line Interface wizard, an `aws-core` plugin, or the Kiro local-proxy configuration on this page, install uv, which includes `uvx`. These methods use `uvx` to run the local proxy for the AWS MCP Server. A direct OAuth connection does not require `uv`.
- This step applies only when you install the skill locally for Kiro. Install Node.js, which provides the `npx` command.
- For a local-proxy or IAM SigV4 connection, configure AWS credentials locally before the agent calls AWS APIs. Documentation search and skill discovery do not require credentials. A direct OAuth connection uses browser sign-in and does not require local AWS credentials for the MCP connection. Local scripts that call AWS APIs still require credentials.
- For a direct OAuth connection, the IAM identity that signs in must allow `signin:AuthorizeOAuth2Access` and `signin:CreateOAuth2Token`.

Step 1: Connect your agent to AWS

Follow the instructions for your agent. When connected, your agent can access the `amazon-ses` skill when needed.

Claude Code

Run these commands in Claude Code:

```
/plugin install aws-core@claude-plugins-official
/reload-plugins
```

> **Note**
>
> If Claude Code reports that the plugin was not found, run `/plugin marketplace update claude-plugins-official`, and then retry the installation.

Codex

Add the Agent Toolkit marketplace from your terminal:

```
codex plugin marketplace add aws/agent-toolkit-for-aws
```

Launch Codex, run `/plugins`, and install `aws-core`. Start a new conversation before you continue to Step 2.

Cursor

In Cursor, choose *Settings*, *Plugins*, *Team Marketplaces*, *Add Marketplace*, and *Import from Repo*. Enter `aws/agent-toolkit-for-aws`. Then open the Plugins panel and install `aws-core`.

Kiro

Kiro uses separate MCP and local skill configurations. Add the AWS MCP Server to `.kiro/settings/mcp.json`. Replace `us-west-2` with the default AWS Region for your operations.

```
{
  "mcpServers": {
    "aws": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws-cli@latest",
```

```
        "https://aws-mcp.us-east-1.api.aws/mcp",
        "--metadata", "AWS_REGION=us-west-2"
      ]
    }
  }
}
```

From your project root, install the local `amazon-ses` skill and its reference files:

```
npx skills add https://github.com/aws/agent-toolkit-for-aws \
    --skill amazon-ses \
    --agent kiro-cli \
    --yes
```

Alternative: Automatic setup with the AWS Command Line Interface

If you have AWS Command Line Interface version 2.35.0 or later, run the setup wizard:

```
aws configure agent-toolkit
```

The wizard detects supported installed agents, installs the default AWS skills, and configures the AWS MCP Server. Restart your agent after the wizard completes.

Alternative: Connect directly with OAuth 2.1

If your agent supports remote MCP servers, you can connect directly with OAuth 2.1 instead of running the local proxy. This path uses browser sign-in and does not require `uv` or local AWS credentials for the MCP connection. For commands for Claude Code, Codex, Cursor, and Kiro, see OAuth 2.1 authentication for AWS MCP Server. A direct connection does not install the local skills included with `aws-core`.

Another agent or custom framework

If your agent supports MCP, connect it directly to the AWS MCP Server. For client configuration and authentication options, including OAuth and IAM SigV4, see Setting up the AWS MCP Server.

If the agent supports Agent Skills, you can also install `amazon-ses` locally from the Agent Toolkit for AWS on GitHub. Choose the target agent when prompted.

Step 2: Use the Amazon SES skill

For Amazon SES access-control guidance, see Identity and access management in Amazon SES. For connection or authentication problems, see Setting up the AWS MCP Server.

Describe the outcome that you want. You don't need to name the skill. For example, replace `example.com` with your domain and use this prompt:

```
Set up example.com as a sending domain in Amazon SES.
```

Keep your setup current

The AWS MCP Server provides the current `amazon-ses` skill. If you installed a local copy, repeat the installation process to update it.

Getting started with Amazon Simple Email Service

This chapter guides you through tasks required for initial set up of Amazon SES as well as tutorials to help you get started.

Topics

- Setting up Amazon Simple Email Service
- Migrating to Amazon SES from another email-sending solution
- Request production access (Moving out of the Amazon SES sandbox)

Setting up Amazon Simple Email Service

Before you start using Amazon SES, you must complete the following tasks.

Tasks

- Sign up for an AWS account
- Set up your SES account
- Download an AWS SDK (For using the SES APIs)

Sign up for an AWS account

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

Set up your SES account

Get started with SES by verifying an email address and sending domain so that you can start sending email through SES and request production access for your account by using the *SES account set up* wizard.

Using the *SES account set up* wizard to set up your account

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

2. Select **Get started** from the SES console home page and the wizard will walk you through the steps of setting up your SES account.

The SES account set up *wizard will only be presented if you have not yet created any identities (email address or domain) in SES.*

Download an AWS SDK (For using the SES APIs)

To call the SES APIs without having to handle low-level details like assembling raw HTTP requests, you can use an AWS SDK. The AWS SDKs provide functions and data types that encapsulate the functionality of SES and other AWS services. To download an AWS SDK, go to SDKs. After you download the SDK, create a shared credentials file and specify your AWS access keys.

Migrating to Amazon SES from another email-sending solution

This topic provides an overview of the steps that you have to take if you want to move your email-sending solution to Amazon SES from a solution that's hosted on-premises, or from one hosted on an Amazon EC2 instance.

Topics in this section:

- Step 1. Verify your domain
- Step 2. Request production access
- Step 3. Configure domain authentication systems
- Step 4. Generate your SMTP credentials
- Step 5. Connect to an SMTP endpoint
- Next steps

Step 1. Verify your domain

Before you can use Amazon SES to send email, you have to verify the identities that you plan to send email from. In Amazon SES, an identity can be an email address or an entire domain. When you verify a domain, you can use Amazon SES to send email from any address on that domain. For more information about verifying a domain, see Creating a domain identity.

Step 2. Request production access

When you first start using Amazon SES, your account is in a sandbox environment. While your account is in the sandbox, you can only send email to addresses that you've verified. Additionally, there are restrictions on the number of messages that you can send per day, and the number that you can send per second. For more information about requesting production access, see Request production access (Moving out of the Amazon SES sandbox).

Step 3. Configure domain authentication systems

You can configure your domain to use authentication systems such as DKIM and SPF. This step is technically optional. However, by setting up either DKIM or SPF (or both) for your domain, you can improve the deliverability of your emails, and increase the amount of trust that your customers have in you. For more information about setting up SPF, see Authenticating Email with SPF in Amazon SES. For more information about setting up DKIM, see Authenticating Email with DKIM in Amazon SES.

Step 4. Generate your SMTP credentials

If you plan to send email using an application that uses SMTP, you have to generate SMTP credentials. Your SMTP credentials are different from your regular AWS credentials. These credentials are also unique in each AWS Region. For more information about generating your SMTP credentials, see Obtaining Amazon SES SMTP credentials.

Step 5. Connect to an SMTP endpoint

If you use a message transfer agent such as postfix or sendmail, you have to update the configuration for that application to refer to an Amazon SES SMTP endpoint. For a complete list of SMTP endpoints, see Connecting to an Amazon SES SMTP endpoint. Note that the SMTP credentials that you created in the previous step are associated with a specific AWS Region. You have to connect to the SMTP endpoint in the region that you created the SMTP credentials in.

Next steps

At this point, you're ready to start sending email using Amazon SES. However, there are a few optional steps that you can take.

- You can create configuration sets, which are sets of rules that are applied to the emails that you send. For example, you can use configuration sets to specify where notifications are sent

when an email is delivered, when a recipient opens a message or clicks a link in it, when an email bounces, and when a recipient marks your email as spam. For more information, see Using configuration sets in Amazon SES.

- When you send email through Amazon SES, it's important to monitor the bounces and complaints for your account. Amazon SES includes a reputation metrics console page that you can use to keep track of the bounces and complaints for your account. For more information, see Using reputation metrics to track bounce and complaint rates. You can also create CloudWatch alarms that alert you when these rates get too high. For more information about creating CloudWatch alarms, see Creating reputation monitoring alarms using CloudWatch.
- Customers who send a large volume of email, or those who simply want to have full control over the reputations of their IP addresses, can lease dedicated IP addresses for an additional monthly charge. For more information, see Dedicated IP addresses for Amazon SES.

Request production access (Moving out of the Amazon SES sandbox)

To help prevent fraud and abuse, and to help protect your reputation as a sender, we apply certain restrictions to new Amazon SES accounts.

We place all new accounts in the Amazon SES *sandbox*. The sandbox status for your account is unique per each AWS Region. While your account is in the sandbox, you can use all of the features of Amazon SES. However, when your account is in the sandbox, we apply the following restrictions to your account:

- You can only send mail **to** verified email addresses and domains, or to the Amazon SES mailbox simulator.
- You can send a maximum of 200 messages per 24-hour period.
- You can send a maximum of 1 message per second.
- For sending authorization, neither you nor the delegate sender can send email to non-verified email addresses.
- For account-level suppression, bulk actions and SES API calls related to suppression list management are disabled.

When your account has moved out of the sandbox and into production, you can send email to any recipient, regardless of whether the recipient's address or domain is verified. However, you still have to verify all identities that you use as "From", "Source", "Sender", or "Return-Path" addresses.

Complete the procedures in this section to request that your account be removed from the sandbox and placed into production.

Tip

- If you're a new customer and haven't created any identities yet, then the SES account set up wizard in the console will be enabled to help you get started. See Set up your SES account for instructions on how to access the wizard.
- If you've already created one or more identities, you'll see the **Get set up** page instead of the account set up wizard.
  - If one of your identities is a verified domain, you'll be able to request production access directly from the **Get set up** page as well. This is because verifying your domain with SES before requesting production access is a best practice that helps to get your production access request approved faster so you can start sending email right away.

Note

- If you're using Amazon SES to send email from an Amazon EC2 instance, you might also need to request that the throttle be removed from port 25 on your Amazon EC2 instance. For more information, see How do I remove the throttle on port 25 from my EC2 instance? in the AWS Knowledge Center.

To request production access (remove your account from the sandbox) using the AWS Management Console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation panel, choose **Account dashboard**.
3. In the warning box at the top of the console that says, "Your Amazon SES account is in the sandbox", on the right-hand side, choose **View Get set up page** followed by **Request production access**.

4. In the account details modal, select either the **Marketing** or **Transactional** radio button that best describes the majority of mail you'll be sending.

   - *Marketing email* - Sent on a one-to-many basis to a targeted list of prospects or customers containing marketing and promotional content such as to make a purchase, download information, etc.
   - *Transactional email* - Sent on a one-to-one basis unique to each recipient usually triggered by a user action such as a website purchase, a password reset request, etc.
5. In **Website URL**, enter the URL of your website to help us better understand the kind of content you plan on sending.
6. In **Additional contacts**, tell us where you want to receive communications about your account. This can be a comma-separated list of up to 4 email addresses.
7. In **Preferred contact language**, choose whether you want to receive communications in **English** or **Japanese**.
8. In **Acknowledgement**, check the box that you agree to only send email to individuals who've explicitly requested it and confirm that you have a process in place for handling bounce and complaint notifications.
9. Choose the **Submit request** button - a banner will display to confirm your request was submitted and is currently under review.

Once you submit a review of your account details, you can't edit your details until the review is complete. The AWS Support team provides an initial response to your request within 96 hours.

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. If we're able to do so, we'll grant your request within this 96-hour period. However, if we need to obtain additional information from you, it might take longer to resolve your request.

Optionally, you can also submit your request for production access using the AWS CLI. Submitting your request using the AWS CLI is helpful when you want to request production access for a large number of identities, or when you want to automate the process of setting up Amazon SES.

To request that your account be removed from the Amazon SES sandbox using the AWS CLI

1. **Prerequisite**: you have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide.
2. At the command line, enter the following command:

```
aws sesv2 put-account-details \
--production-access-enabled \
--mail-type TRANSACTIONAL \
--website-url https://example.com \
--additional-contact-email-addresses info@example.com \
--contact-language EN
```

In the preceding command, do the following:

a. Replace *TRANSACTIONAL* with the type of email that you plan to send through Amazon SES. You can specify either TRANSACTIONAL or MARKETING. If more than one value applies, specify the option that applies to the majority of the email that you plan to send.

b. Replace *https://example.com* with the URL of your website. Providing this information helps us better understand the type of content that you plan to send.

c. Replace *info@example.com* with the email addresses where you want to receive communications about your account. This can be a comma-separated list of up to 4 email addresses.

d. Replace *EN* with your preferred language. You can specify EN for English or JA for Japanese.

Once you submit a review of your account details, you can't edit your details until the review is complete. The AWS Support team provides an initial response to your request within 96 hours.

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. If we're able to do so, we'll grant your request within this 96-hour period. However, if we need to obtain additional information from you, it might take longer to resolve your request.

Managing your Amazon SES sending limits

Your Amazon SES account has a set of sending quotas that regulate the number of email messages that you can send and the rate at which you can send them. Sending quotas benefit all Amazon SES customers because they help to maintain the trusted relationship between Amazon SES and email providers. Sending quotas help you to gradually ramp up your sending activity and decrease the likelihood that email providers block your emails because of sudden, unexpected spikes in your email sending volume or rate.

The following quotas apply to sending email through Amazon SES:

- **Sending quota**—The maximum number of emails that you can send in a 24-hour period. This quota is calculated on a rolling time period. Every time you try to send an email, Amazon SES determines the number of emails that you sent in the previous 24 hours. As long as the total number of emails that you have sent in the past 24 hours is less than this daily maximum, your send request is accepted and your email is sent.

  If sending a message would exceed the daily maximum for your account, your call to Amazon SES is rejected.
- **Sending rate**—The maximum number of emails that Amazon SES can accept from your account each second. You can exceed this quota for short bursts, but not for sustained periods of time.

  > **Note**
  >
  > The rate at which Amazon SES accepts your messages can be less than the maximum send rate for your account.

- **Maximum message size (MB)**—The maximum email size that you can send. This includes any images and attachments that are part of the email after MIME encoding. For example, if you attach a 5MB file, the attachment size in the email after MIME encoding will be ~6.85MB (about 137% of the original file size).

  > **Note**
  >
  > We recommend you upload your attachments to cloud drives and include the URL of cloud drive attachment to reduce email size and improve deliverability. SES cannot

guarantee that large emails will end up in the recipient mailbox as different mail servers will have varying size based policies.

Your Amazon SES sending quotas are separate for each AWS Region. For information about using Amazon SES in multiple AWS Regions, see Regions and Amazon SES.

When your account is in the Amazon SES sandbox, you can only send 200 messages per 24-hour period, and your maximum sending rate is one message per second. When you submit a request to have your account removed from the sandbox, you can also request that your quotas are increased at the same time. For more information about having your account removed from the sandbox, see Request production access (Moving out of the Amazon SES sandbox).

When your account has been removed from the sandbox, you can request additional quota increases at any time by creating a new case in the AWS Support Center. For more information, see Increasing your Amazon SES sending quotas.

Note

Sending quotas are based on recipients rather than on messages. For example, an email that has 10 recipients counts as 10 against your quota. However, we don't recommend that you send an email to multiple recipients in a single call to the `SendEmail` API operation, because if the call fails, the entire email is rejected. We recommend that you call `SendEmail` once for every recipient.

- To increase your sending quotas, see Increasing your Amazon SES sending quotas.
- To monitor your sending quotas by using the Amazon SES console or the Amazon SES API, see Monitoring your Amazon SES sending quotas.
- For information about the errors your application receives when you reach your sending quotas, see Errors related to the sending quotas for your Amazon SES account.

Increasing your Amazon SES sending quotas

Your account has the following quotas per your current region that can be increased.

Resource	Default quota	Description
Sending quota	200	Maximum number of emails that you can send in a 24-hour period for this account in the current AWS Region.
Sending rate	1	Maximum number of emails that Amazon SES can accept each second for this account in the current AWS Region.

Automatically increased sending quotas

When your account is out of the sandbox and you're sending high-quality production email, we might automatically increase the sending quotas for your account. Often, we automatically increase these quotas before you actually need them to be increased.

To qualify for automatic rate increases, all of the following statements have to be true:

- **You send high-quality content that your recipients want to receive** –Send content that recipients want and expect. Stop sending email to customers who don't open your email.
- **You send actual production content** – Sending test messages to fake email addresses can have a negative effect on your bounce and complaint rates. Also, sending messages only to internal recipients makes it difficult to determine if you're sending content that customers want to receive. However, when you send your production messages to non-internal recipients, we can accurately assess your email-sending practices.
- **You send near your current quota** – To qualify for an automatic quota increase, your daily email volume should regularly approach the daily maximum for your account without exceeding it.
- **You have low bounce and complaint rates** – Minimize the number of bounces and complaints that you receive. Having a high number of bounces and complaints can have a negative impact on your sending quotas.

User requested increased sending quotas

If your current sending quotas aren't adequate for your needs and we haven't automatically increased them, you can request an increase:

- **Sending quota or Sending rate** – Increase requests for either of these can be submitted through the *AWS Service Quotas console*.

To request an increase on your Amazon SES sending quotas using the Service Quotas console.

1. Open the Service Quotas console.
2. Select the region that you want the increase for by using the dropdown in the upper right-hand corner of the console (next to your account number).
3. In the navigation pane, choose **AWS services**.
4. Choose **Amazon Simple Email Service (SES)**.
5. Choose a quota, and follow the directions to request a quota increase.

> **AWS Support team SLA for increase requests types**
>
> In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. If we're able to do so, we'll grant your request within the specified times listed below for the type of increase requested. However, if we need to obtain additional information from you, it might take longer to resolve your request. We reserve the right not to grant your request if your use case doesn't align with our policies.
>
> - **Sending quota or Sending rate**: Up to 24 hours.

> **Note**
>
> While the Service Quotas console is available in many different languages, the actual support is only provided in English.

Monitoring your Amazon SES sending quotas

You can monitor your sending quotas by using the Amazon SES console or through the Amazon SES API, whether by calling the Query (HTTPS) interface directly or indirectly through an AWS SDK, the AWS Command Line Interface, or the AWS Tools for Windows PowerShell.

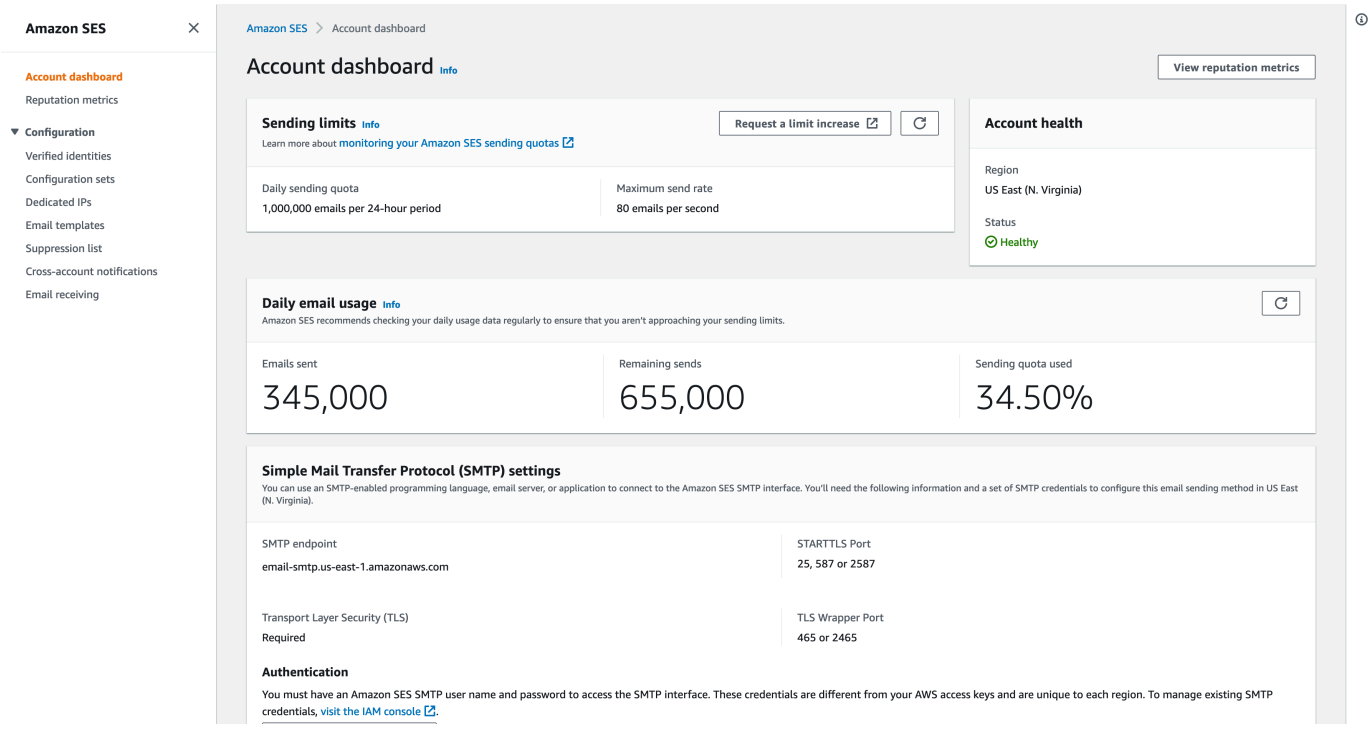
Important

We recommend that you frequently check your sending statistics to ensure that you are not close to your sending quotas. If you are close to your sending quotas, see Increasing your Amazon SES sending quotas for information about how to increase them. Don't wait until you reach your sending quotas to consider increasing them.

Monitoring your sending quotas using the Amazon SES console

The following procedure shows you how to view your sending quotas using the Amazon SES console.

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, choose **Account dashboard**. Your sending quotas are shown under **Sending Limits**. Total emails sent, remaining sends, and percentage of sending quota used is displayed under **Daily email usage**.



3. To update the display, select the refresh icon in the upper right-hand corner of the **Daily email usage** box.

Monitoring your sending quotas using the Amazon SES API

The Amazon SES API provides the `GetSendQuota` action, which returns your sending quotas. When you call `GetSendQuota` action, you receive the following information:

- Number of emails you have sent during the past 24 hours
- Sending quota for the current 24-hour period
- Maximum send rate

> **Note**
>
> For a description of `GetSendQuota`, see Amazon Simple Email Service API Reference.

Errors related to the sending quotas for your Amazon SES account

If you attempt to send an email after reaching your daily sending quota (the maximum amount of email you can send in a 24-hour period) or your maximum sending rate (the maximum number of messages you can send per second), Amazon SES drops the message and doesn't attempt to redeliver it. Amazon SES also provides an error message that explains the issue. The way that Amazon SES produces this error message depends on how you attempted to send the email. This topic includes information about the messages you receive through the Amazon SES API and through the SMTP interface.

For a technique that you can use when you reach your maximum send rate, see How to handle a "Throttling – Maximum sending rate exceeded" error on the AWS Messaging and Targeting Blog.

Reaching sending limits with the Amazon SES API

If you attempt to send an email by using the Amazon SES API (or an AWS SDK), but you've already exceeded your account's sending limits, the API produces a `ThrottlingException` error. The error message includes one of the following messages:

- `Daily message quota exceeded`
- `Maximum sending rate exceeded`

If you encounter a throttling error, you should program your application to wait for an interval of up to 10 minutes, and then retry the send request.

Reaching sending limits with SMTP

If you attempt to send an email by using the Amazon SES SMTP interface, but you've already exceeded your account's sending limits, your SMTP client might display one of the following errors:

- `454 Throttling failure: Maximum sending rate exceeded`
- `454 Throttling failure: Daily message quota exceeded`

Different SMTP clients handle these errors in different ways.

Set up email sending with Amazon SES

You can send an email with Amazon Simple Email Service (Amazon SES) using the Amazon SES console, the Amazon SES Simple Mail Transfer Protocol (SMTP) interface, or the Amazon SES API. You typically use the console to send test emails and manage your sending activity. To send bulk emails, you use either the SMTP interface or the API. For information about Amazon SES email pricing, see Amazon SES Pricing.

- If you want to use an SMTP-enabled software package, application, or programming language to send email through Amazon SES, or integrate Amazon SES with your existing mail server, use the Amazon SES SMTP interface. For more information, see Sending emails programmatically through the Amazon SES SMTP interface.
- If you want to call Amazon SES by using raw HTTP requests, use the Amazon SES API. For more information, see Using the Amazon SES API to send email.

> **Important**
>
> When you send an email to multiple recipients (recipients are "To", "CC", and "BCC" addresses) and the call to Amazon SES fails, the entire email is rejected and none of the recipients will receive the intended email. Therefore, we recommend that you send an email to one recipient at a time.

Using the Amazon SES SMTP interface to send email

To send production email through Amazon SES, you can use the Simple Mail Transfer Protocol (SMTP) interface or the Amazon SES API. For more information about the Amazon SES API, see Using the Amazon SES API to send email. This section describes the SMTP interface.

Amazon SES sends email using SMTP, which is the most common email protocol on the internet. You can send email through Amazon SES by using a variety of SMTP-enabled programming languages and software to connect to the Amazon SES SMTP interface. This section explains how to get your Amazon SES SMTP credentials, how to send email by using the SMTP interface, and how to configure several pieces of software and mail servers to use Amazon SES for email sending.

For solutions to common problems that you might encounter when you use Amazon SES through its SMTP interface, see Amazon SES SMTP issues.

Requirements to send email over SMTP

To send email using the Amazon SES SMTP interface, you need the following:

- The SMTP endpoint address. For a list of Amazon SES SMTP endpoints, see Connecting to an Amazon SES SMTP endpoint.
- The SMTP interface port number. The port number varies with the connection method. For more information, see Connecting to an Amazon SES SMTP endpoint.
- An SMTP user name and password. SMTP credentials are unique to each AWS Region. If you plan to use the SMTP interface to send email in multiple AWS Regions, you need SMTP credentials for each Region.

> **Important**
>
> Your SMTP credentials aren't identical to your AWS access keys or the credentials that you use to sign in to the Amazon SES console. For information about how to generate your SMTP credentials, see Obtaining Amazon SES SMTP credentials.

- Client software that can communicate using Transport Layer Security (TLS). For more information, see Connecting to an Amazon SES SMTP endpoint.
- An email address that you've verified with Amazon SES. For more information, see Verified identities in Amazon SES.
- Increased sending quotas, if you want to send large quantities of email. For more information, see Managing your Amazon SES sending limits.

Methods to send email over SMTP

You can send email over SMTP through any of the following methods:

- To configure SMTP-enabled software to send email through the Amazon SES SMTP interface, see Sending email through Amazon SES using software packages.
- To program an application to send email through Amazon SES, see Sending emails programmatically through the Amazon SES SMTP interface.
- To configure your existing email server to send all of your outgoing mail through Amazon SES, see Integrating Amazon SES with your existing email server.

- To interact with the Amazon SES SMTP interface using the command line, which can be useful for testing, see Testing your connection to the Amazon SES SMTP interface using the command line.

For a list of SMTP response codes, see SMTP response codes returned by Amazon SES.

Email information to provide

When you access Amazon SES through the SMTP interface, your SMTP client application assembles the message, so the information you need to provide depends on the application that you're using. At a minimum, the SMTP exchange between a client and a server requires the following:

- a source address
- a destination address
- message data

If you're using the SMTP interface and have feedback forwarding enabled, then your bounces, complaints, and delivery notifications are sent to the "MAIL FROM" address. Any "Reply-To" address that you specify isn't used.

Obtaining Amazon SES SMTP credentials

You need Amazon SES SMTP credentials to access the SES SMTP interface.

The credentials that you use to send email through the SES SMTP interface are unique to each AWS Region. If you use the SES SMTP interface to send email in more than one Region, you must generate a set of SMTP credentials for each Region that you plan to use.

Your SMTP password is different from your AWS secret access key. For more information about credentials, see Types of Amazon SES credentials.

> **Note**
>
> For a list of currently available SMTP endpoints, see SMTP endpoints in the *AWS General Reference*.

Obtaining SES SMTP credentials using the SES console

Requirement

An IAM user can create SES SMTP credentials, but the user's policy must give them permission to use IAM itself, because SES SMTP credentials are created by using IAM. Your IAM policy must allow you to perform the following IAM actions: `iam:CreateUser`, `iam:CreateGroup`, `iam:PutGroupPolicy`, `iam:AddUserToGroup`, and `iam:CreateAccessKey`. If you try to create SES SMTP credentials using the console and your IAM user doesn't have these permissions, you may see an error or the SMTP credentials may be created without the proper policy settings.

> **Important**
>
> Some of the IAM actions referenced above, specifically `iam:PutGroupPolicy` and `iam:AddUserToGroup`, have the Permission management access level which is the highest IAM level because it gives permission to grant or modify resource permissions in the service. Therefore, to improve the security of your AWS account, it is highly recommended that you restrict or regularly monitor these policies that include the Permissions management access level classification.

To create your SMTP credentials

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Choose **SMTP settings** in the left navigation pane - this will open the **Simple Mail Transfer Protocol (SMTP) settings** page.
3. Choose **Create SMTP Credentials** in the upper-right corner - the IAM console will open.
4. (Optional) If you need to view, edit, or delete SMTP users you've already created, choose **Manage my existing SMTP credentials** in the lower-right corner - the IAM console will open. Details for managing SMTP credentials is given following these procedures.
5. For **Create User for SMTP**, type a name for your SMTP user in the **User Name** field. Alternatively, you can use the default value that is provided in this field. When you finish, choose **Create user** in the bottom-right corner.
6. Select **Show** under *SMTP password* - your SMTP credentials are shown on the screen.
7. Download these credentials by choosing **Download .csv file** or copy them and store them in a safe place, because you can't view or save your credentials after you close this dialog box.

8. Choose **Return to SES console**.

You can view a list of the SMTP credentials you've created using this procedure in the IAM console under **Access management** and choosing **Users** followed by using the search bar to find all users that you've assigned SMTP credentials.

You can also use the IAM console to delete existing SMTP users. To learn more about deleting users, see Managing IAM Users in the *IAM Getting Started Guide*.

If you want to change your SMTP password, delete your existing SMTP user in the IAM console. Then, to generate a new set of SMTP credentials, complete the previous procedures.

Obtaining SES SMTP credentials by converting existing AWS credentials

If you have a user that you set up using the IAM interface, you can derive the user's SES SMTP credentials from their AWS credentials.

> **Important**
>
> Don't use temporary AWS credentials to derive SMTP credentials. The SES SMTP interface doesn't support SMTP credentials that have been generated from temporary security credentials.

To enable the IAM user to send email using the SES SMTP interface

1. Derive the user's SMTP credentials from their AWS credentials by using the algorithm provided in this section following these procedures.

   Because you're starting from AWS credentials, the SMTP user name is the same as the AWS access key ID, so you only need to generate the SMTP password.
2. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
3. Under **Access management**, choose **Polices** followed by **Create policy**.
4. In the **Policy editor**, select **JSON** and remove any example code in the editor.
5. Paste to the following permissions policy into the editor:

JSON

```
{
"Version":"2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": "ses:SendRawEmail",
            "Resource": "*"
        }
    ]
}
```

6. Select **Next** and enter `AmazonSesSendingAccess` in the **Policy name** field followed by **Create policy**.
7. Under **Access management**, choose **User groups** followed by **Create group**.
8. Enter `AWSSESSendingGroupDoNotRename` in the **User group name** field.
9. Add SMTP users to the group by selecting them from the **Add users to the group** table.
10. Attach the `AmazonSesSendingAccess` policy created previously by selecting it from the **Attach permissions policies** table followed by **Create user group**.

For more information about using SES with IAM, see Identity and access management in Amazon SES.

> **Note**
>
> Although you can generate SES SMTP credentials for any IAM user, we recommend that you create a separate IAM user when you generate your SMTP credentials. For information about why it's good practice to create users for specific purposes, go to IAM Best Practices.

The following pseudocode shows the algorithm that converts an AWS secret access key to an SES SMTP password.

```
// Modify this variable to include your AWS secret access key
key = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";
```

```
// Modify this variable to refer to the AWS Region that you want to use to send email.
region = "us-west-2";

// The values of the following variables should always stay the same.
date = "11111111";
service = "ses";
terminal = "aws4_request";
message = "SendRawEmail";
version = 0x04;

kDate = HmacSha256(date, "AWS4" + key);
kRegion = HmacSha256(region, kDate);
kService = HmacSha256(service, kRegion);
kTerminal = HmacSha256(terminal, kService);
kMessage = HmacSha256(message, kTerminal);
signatureAndVersion = Concatenate(version, kMessage);
smtpPassword = Base64(signatureAndVersion);
```

Some programming languages include libraries that you can use to convert an IAM secret access key into an SMTP password. This section includes a code example that you can use to convert an AWS secret access key to an SES SMTP password using Python.

Note

- The following example uses **f-strings** that were introduced in Python 3.6; if using an older version, they won't work.
- In the following example, the list of SMTP_REGIONS is simply an example—your actual list of regions could be shorter or longer depending on which regions you plan to send email in as you'll need SMTP credentials for each AWS Region.

Python

```
#!/usr/bin/env python3

import hmac
import hashlib
import base64
import argparse

SMTP_REGIONS = [
```

```
    "us-east-2",  # US East (Ohio)
    "us-east-1",  # US East (N. Virginia)
    "us-west-2",  # US West (Oregon)
    "ap-south-1",  # Asia Pacific (Mumbai)
    "ap-northeast-2",  # Asia Pacific (Seoul)
    "ap-southeast-1",  # Asia Pacific (Singapore)
    "ap-southeast-2",  # Asia Pacific (Sydney)
    "ap-northeast-1",  # Asia Pacific (Tokyo)
    "ca-central-1",  # Canada (Central)
    "eu-central-1",  # Europe (Frankfurt)
    "eu-west-1",  # Europe (Ireland)
    "eu-west-2",  # Europe (London)
    "eu-south-1",  # Europe (Milan)
    "eu-north-1",  # Europe (Stockholm)
    "sa-east-1",  # South America (Sao Paulo)
    "us-gov-west-1",  # AWS GovCloud (US)
    "us-gov-east-1",  # AWS GovCloud (US)
]

# These values are required to calculate the signature. Do not change them.
DATE = "11111111"
SERVICE = "ses"
MESSAGE = "SendRawEmail"
TERMINAL = "aws4_request"
VERSION = 0x04


def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()


def calculate_key(secret_access_key, region):
    if region not in SMTP_REGIONS:
        raise ValueError(f"The {region} Region doesn't have an SMTP endpoint.")

    signature = sign(("AWS4" + secret_access_key).encode("utf-8"), DATE)
    signature = sign(signature, region)
    signature = sign(signature, SERVICE)
    signature = sign(signature, TERMINAL)
    signature = sign(signature, MESSAGE)
    signature_and_version = bytes([VERSION]) + signature
    smtp_password = base64.b64encode(signature_and_version)
    return smtp_password.decode("utf-8")
```

```
def main():
    parser = argparse.ArgumentParser(
        description="Convert a Secret Access Key to an SMTP password."
    )
    parser.add_argument("secret", help="The Secret Access Key to convert.")
    parser.add_argument(
        "region",
        help="The AWS Region where the SMTP password will be used.",
        choices=SMTP_REGIONS,
    )
    args = parser.parse_args()
    print(calculate_key(args.secret, args.region))


if __name__ == "__main__":
    main()
```

To obtain your SMTP password by using this script, save the preceding code as `smtp_credentials_generate.py`. Then, at the command line, run the following command:

```
python path/to/smtp_credentials_generate.py wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY us-east-1
```

In the preceding command, do the following:

- Replace *path/to/* with the path to the location where you saved `smtp_credentials_generate.py`.
- Replace *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY* with the secret access key that you want to convert into an SMTP password.
- Replace *us-east-1* with the AWS Region in which you want to use the SMTP credentials.

When this script runs successfully, the only output is your SMTP password.

Migrating a SMTP user from an existing inline policy to a group policy (security recommendation)

> **Important**
>
> If you created SES SMTP credentials before September 6, 2024, an inline policy and a tag have been attached to your SMTP user. SES is moving away from inline policies and encourages you to do the same as a security recommendation.

Before migrating a SMTP user off of an existing inline policy to a group policy, you must first create an IAM user group with the SES permissions policy to take the place of the inline policy. If you've already created this IAM user group, or it was automatically created for SMTP credentials you created from September 6, 2024 onward, you can skip directly to *step 10* in the following procedures.

To migrate from an existing inline policy to a managed group

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. Under **Access management**, choose **Polices** followed by **Create policy**.
3. In the **Policy editor**, select **JSON** and remove any example code in the editor.
4. Paste to the following permissions policy into the editor:

   JSON

```
{
"Version":"2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": "ses:SendRawEmail",
            "Resource": "*"
        }
    ]
}
```

5. Select **Next** and enter `AmazonSesSendingAccess` in the **Policy name** field followed by **Create policy**.
6. Under **Access management**, choose **User groups** followed by **Create group**.
7. Enter `AWSSESSendingGroupDoNotRename` in the **User group name** field.
8. Add SMTP users to the group by selecting them from the **Add users to the group** table.
9. Attach the `AmazonSesSendingAccess` policy created previously by selecting it from the **Attach permissions policies** table followed by **Create user group**.

   Now that you've created the IAM user group with the SES permissions policy, you can migrate a SMTP user from their current inline policy to this group policy as explained in the remaining steps.
10. Under **Access management**, choose **Users** followed by selecting the SMTP user you want to migrate.
11. Select the **Groups** tab and choose **Add user to groups**.
12. Select the `AWSSESSendingGroupDoNotRename` group followed by **Add user to group(s)**.
13. Select the **Permissions** tab and confirm that there are two rows listed with `AmazonSesSendingAccess` the in the **Policy name** column, one with *Inline* and one with *Group `AWSSESSendingGroupDoNotRename`* listed in the **Attached via** column.
14. Select only the row that contains `AmazonSesSendingAccess` in the **Policy name** column and *Inline* in the **Attached via** column followed by **Remove** and confirm with **Remove policy**.

    Verify the row with *Group `AWSSESSendingGroupDoNotRename`* in the **Attached via** column remains.
15. Select the **Tags** tab followed by **Manage tags**.
16. Select **Remove** next to the row that contains *InvokedBy* in the **Key** column and *SESConsole* in the **Value** column followed by **Save changes**.

> **Important**
>
> The `AmazonSesSendingAccess` policy (either as an inline or group policy or both) must remain attached to the SMTP user to make sure their sending is not impacted. Only remove the inline policy after the group policy is attached to your user.

Creating SMTP credentials using Mail Manager

You can use Mail Manager to create SMTP credentials from the SES console. The console creates the supporting Mail Manager resources for you, including a Traffic policies and policy statements, a Rule sets and rules, an IAM role, and an authenticated Ingress endpoints.

In Regions where both options are available, we recommend using Mail Manager because it's available in all AWS Regions — including opt-in Regions and the European Sovereign Cloud — and the console creates and configures the required resources for you. The IAM user credential flow described in Obtaining Amazon SES SMTP credentials remains available in the Regions where it's currently supported.

Region availability

In opt-in Regions and the European Sovereign Cloud, Mail Manager is the only option for creating SMTP credentials. In all other Regions, you can use either Mail Manager (recommended) or the IAM user credential flow.

Prerequisites

Before you create SMTP credentials with Mail Manager, verify that your IAM user or role has permissions to perform the following actions:

- Create Mail Manager Traffic policies and policy statements
- Create Mail Manager Rule sets and rules
- Create Mail Manager Ingress endpoints
- Create IAM roles. If you provide your own role instead, it must include the permissions described in the section called “Send to internet policy”.
- If you choose to auto-generate a password, the `secretsmanager:GetRandomPassword` permission for AWS Secrets Manager. For more information, see GetRandomPassword in the *AWS Secrets Manager API Reference*.

To create SMTP credentials using Mail Manager

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

2. Choose **SMTP settings** in the left navigation pane.
3. Choose **Create SMTP credentials**. In Regions that offer both options, select **Mail Manager** (recommended).
4. For **Ingress endpoint name**, enter a name for your ingress endpoint or use the auto-populated default name.
5. For **Authentication type**, choose one of the following:
   - **SMTP password** – Enter a custom password or choose **Auto-generate** to create one automatically. The password must be 8–64 characters and include at least one uppercase letter, one lowercase letter, one digit, and one special character.
   - **Secret** – Select an existing secret in AWS Secrets Manager, or create a new secret. The secret must be an **Other type of secret** with a key/value pair in which the key is `password` and the value is your password. For more information about configuring the secret, see Creating an ingress endpoint in the SES console.
6. Choose **Create SMTP credentials**. The console creates a traffic policy, rule set, IAM role, and authenticated ingress endpoint with auto-generated names.
7. Wait for the ingress endpoint status to become **Active**. The console displays provisioning progress.
8. Choose **Download .csv** to save your credentials. The .csv file contains the SMTP endpoint, port, username (raw and base64-encoded), and password (raw and base64-encoded).

Important

You cannot retrieve your password after you leave this page. Download the .csv file or copy your credentials before closing the dialog.

Note

After you create your credentials, you can change the password or secret at any time on the ingress endpoint details page. For more information, see Ingress endpoints.

Advanced settings

The form includes an optional **Advanced settings** section where you can customize the following options:

- **Traffic policy** – Select an existing traffic policy or create a new one with a custom name.
- **Rule set** – Select an existing rule set or create a new one with a custom name.
- **IAM role** – Enter the ARN of an existing IAM role or create a new role in the IAM console. If you create the role yourself, it must include the permissions described in the section called "Send to internet policy".
- **Network type** – Choose **Public** (default) or **Private**. If you choose **Private**, you must provide a Amazon VPC endpoint ID. For more information, see Receiving email through Amazon VPC endpoints.
- **IP address type** (public only) – Choose **IPv4** (default) or **Dual-stack**.

If you don't expand the advanced settings, default values are used and new resources are created automatically.

Error handling

The following table describes errors that might occur during the creation process and how to resolve them.

Error	Cause	Solution
Resource creation failed	Your IAM user or role doesn't have permissions to create one of the required resources (traffic policy, rule set, IAM role, or ingress endpoint).	Verify that you have the permissions listed in the section called "Prerequisites", and then choose **Retry**. Any resources that were already created are rolled back automatically.
Ingress endpoint provisioning failed	An error occurred while provisioning the ingress endpoint.	Check the status of the ingress endpoint in Mail Manager. The error banner includes a link to Mail

Error	Cause	Solution
		Manager for further troublesh ooting.
Password generation failed	Your IAM user or role doesn't have the `secretsma nager:GetRandomPas sword` permission.	Enter a custom password manually, or add the permission and try again.
Provisioning timeout	The console stopped checking the ingress endpoint status after 10 minutes before it reached **Active**.	Use the link provided in the timeout message to check the current status of your ingress endpoint in Mail Manager.

Connecting to an Amazon SES SMTP endpoint

To send email using the Amazon SES SMTP interface, you connect to an SMTP endpoint. For a complete list of Amazon SES SMTP endpoints, see Amazon Simple Email Service endpoints and quotas in the *AWS General Reference*.

The Amazon SES SMTP endpoint requires that all connections be encrypted using Transport Layer Security (TLS). (Note that TLS is often referred to by the name of its predecessor protocol, SSL.) Amazon SES supports two mechanisms for establishing a TLS-encrypted connection: STARTTLS and TLS Wrapper. Check the documentation for your software to determine whether it supports STARTTLS, TLS Wrapper, or both.

Amazon Elastic Compute Cloud (Amazon EC2) throttles email traffic over port 25 by default. To avoid timeouts when sending email through the SMTP endpoint from EC2, submit a Request to Remove Email Sending Limitations to remove the throttle. Alternatively, you can send email using a different port, or use an Amazon VPC endpoint.

For SMTP connection issues, see SMTP issues.

STARTTLS

STARTTLS is a means of upgrading an unencrypted connection to an encrypted connection. There are versions of STARTTLS for a variety of protocols; the SMTP version is defined in RFC 3207.

To set up a STARTTLS connection, the SMTP client connects to the Amazon SES SMTP endpoint on port 25, 587, or 2587, issues an EHLO command, and waits for the server to announce that it supports the STARTTLS SMTP extension. The client then issues the STARTTLS command, initiating TLS negotiation. When negotiation is complete, the client issues an EHLO command over the new encrypted connection, and the SMTP session proceeds normally.

TLS Wrapper

TLS Wrapper (also known as SMTPS or the Handshake Protocol) is a means of initiating an encrypted connection without first establishing an unencrypted connection. With TLS Wrapper, the Amazon SES SMTP endpoint doesn't perform TLS negotiation: it's the client's responsibility to connect to the endpoint using TLS, and to continue using TLS for the entire conversation. TLS Wrapper is an older protocol, but many clients still support it.

To set up a TLS Wrapper connection, the SMTP client connects to the Amazon SES SMTP endpoint on port 465 or 2465. The server presents its certificate, the client issues an EHLO command, and the SMTP session proceeds normally.

Sending email through Amazon SES using software packages

There are a number of commercial and open source software packages that support sending email through SMTP. Here are some examples:

- Blogging platforms
- RSS aggregators
- List management software
- Workflow systems

You can configure any such SMTP-enabled software to send email through the Amazon SES SMTP interface. For instructions on how to configure SMTP for a particular software package, see the documentation for that software.

The following procedure shows how to set up Amazon SES sending in JIRA, a popular issue-tracking solution. With this configuration, JIRA can notify users through email whenever there is a change in the status of a software issue.

To configure JIRA to send email using Amazon SES

1. Using your web browser, log in to JIRA with administrator credentials.

2. In the browser window, choose **Administration**.
3. On the **System** menu, choose **Mail**.
4. On the **Mail administration** page, choose **Mail Servers**.
5. Choose **Configure new SMTP mail server**.
6. On the **Add SMTP Mail Server** form, fill in the following fields:

   a. **Name**—A descriptive name for this server.

   b. **From address**—The address from which email will be sent. You must verify this email address with Amazon SES before you can send from it. For more information about verification, see Verified identities in Amazon SES.

   c. **Email prefix**—A string that JIRA prepends to each subject line prior to sending.

   d. **Protocol**—Choose **SMTP**.

   > **Note**
   >
   > If you can't connect to Amazon SES using this setting, try **SECURE_SMTP**.

   e. **Hostname**—See Connecting to an Amazon SES SMTP endpoint for a list of Amazon SES SMTP endpoints. For example, if you want to use the Amazon SES endpoint in the US West (Oregon) Region, the hostname would be *email-smtp.us-west-2.amazonaws.com*.

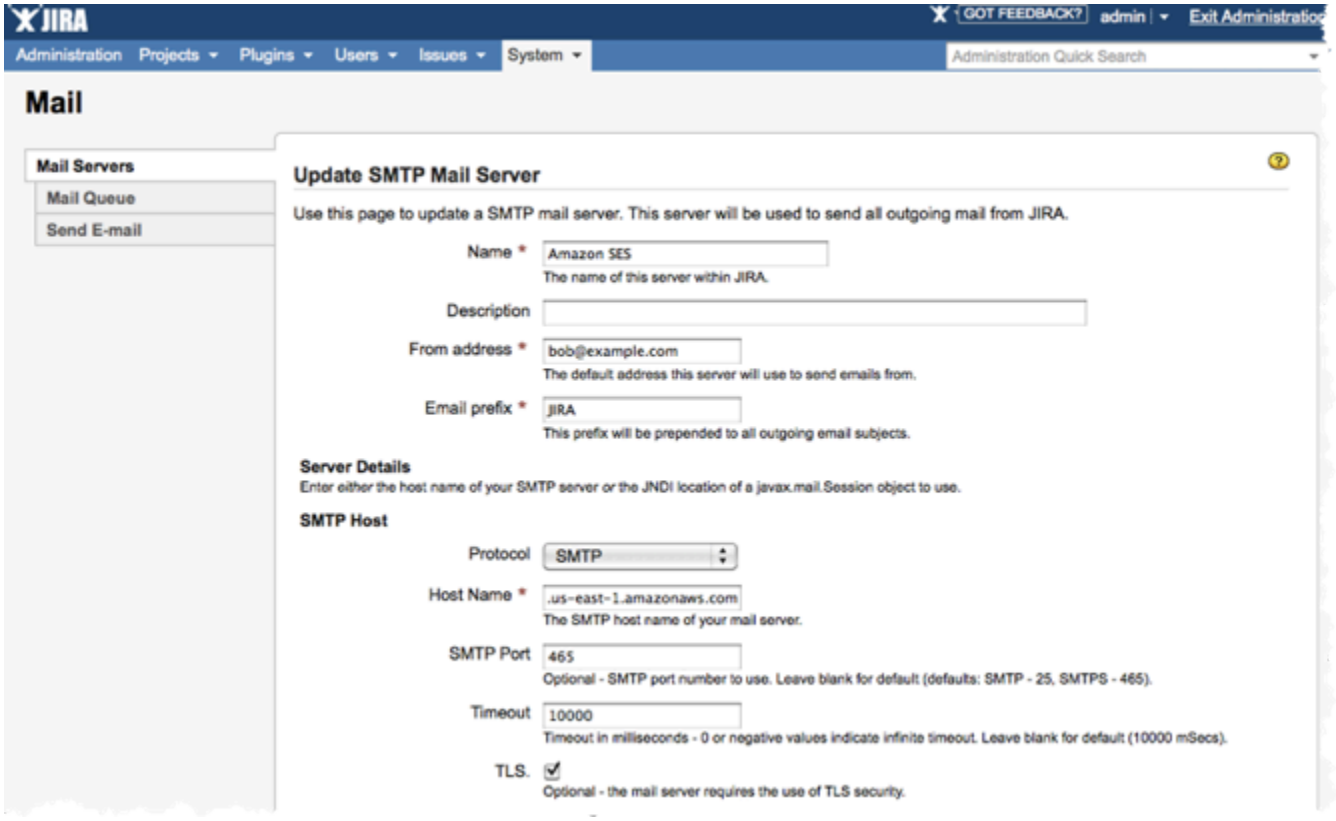
   f. **SMTP port**—25, 587, or 2587 (to connect using STARTTLS), or 465 or 2465 (to connect using TLS Wrapper).

   g. **TLS**—Select this check box.

   h. **User name**—Your SMTP user name.

   i. **Password**—Your SMTP password.

   You can see the settings for TLS Wrapper in the following image.



7. Choose **Test Connection**. If the test email that JIRA sends through Amazon SES arrives successfully, then your configuration is complete.

Sending emails programmatically through the Amazon SES SMTP interface

To send an email using the Amazon SES SMTP interface, you can use an SMTP-enabled programming language, email server, or application. Before you start, complete the tasks in Setting up Amazon Simple Email Service. You also need to get the following information:

- Your Amazon SES SMTP credentials, which enable you to connect to the Amazon SES SMTP endpoint. To get your Amazon SES SMTP credentials, see Obtaining Amazon SES SMTP credentials.

> **Important**
>
> Your SMTP credentials are different from your AWS credentials. For more information about credentials, see Types of Amazon SES credentials.

- The SMTP endpoint address. For a list of Amazon SES SMTP endpoints, see Connecting to an Amazon SES SMTP endpoint.
- The Amazon SES SMTP interface port number, which depends on the connection method. For more information, see Connecting to an Amazon SES SMTP endpoint.

Code examples

You can access the Amazon SES SMTP interface by using an SMTP-enabled programming language. You provide the Amazon SES SMTP hostname and port number along with your SMTP credentials and then use the programming language's generic SMTP functions to send the email.

Amazon Elastic Compute Cloud (Amazon EC2) restricts email traffic over port 25 by default. To avoid timeouts when sending email through the SMTP endpoint from Amazon EC2, you can request that these restrictions be removed. For more information, see How do I remove the restriction on port 25 from my Amazon EC2 instance or AWS Lambda function? in the AWS Knowledge Center.

The code examples in this section for Java and PHP use port 587 to avoid this issue.

> **Note**
>
> In these tutorials, you send an email to yourself so that you can check to see if you received it. For further experimentation or load testing, use the Amazon SES mailbox simulator. Emails that you send to the mailbox simulator do not count toward your sending quota or your bounce and complaint rates. For more information, see Using the mailbox simulator manually.

Select a programing language to view the example for that language:

> **Warning**
>
> Amazon SES does not recommend using static credentials. Refer to AWS Secrets Manager to learn how to improve your security posture by removing hard-coded credentials from your source code. This tutorial is only provided for the purpose of testing the Amazon SES SMTP interface in a non-production environment.

Java

This example uses the Eclipse IDE and the JavaMail API to send email through Amazon SES using the SMTP interface.

Before you perform the following procedure, complete the tasks in Setting up Amazon Simple Email Service.

To send an email using the Amazon SES SMTP interface with Java

1. In a web browser, go to the JavaMail GitHub page. Under **Assets**, choose **javax.mail.jar** to download the latest version of JavaMail.

   > **Important**
   >
   > This tutorial requires JavaMail version 1.5 or later. These procedures were tested using JavaMail version 1.6.1.

2. In a web browser, go to the Jakarta Activation GitHub page, and under JavaBeans Activation Framework 1.2.1 Final Release, download the **jakarta.activation.jar**
3. Create a project in Eclipse by performing the following steps:

   a. Start Eclipse.

   b. In Eclipse, choose **File**, choose **New**, and then choose **Java Project**.

   c. In the **Create a Java Project** dialog box, type a project name and then choose **Next.**

   d. In the **Java Settings** dialog box, choose the **Libraries** tab.

   e. Select **Classpath** and add the two external jar files **javax.mail.jar** and **jakarta.activation.jar** using the **Add External JARs** button.

   f. Choose **Add External JARs**.

   g. Browse to the folder in which you downloaded JavaMail. Choose the file `javax.mail.jar`, and then choose **Open**.

   h. In the **Java Settings** dialog box, choose **Finish**.

4. In Eclipse, in the **Package Explorer** window, expand your project.
5. Under your project, right-click the **src** directory, choose **New**, and then choose **Class**.
6. In the **New Java Class** dialog box, in the **Name** field, type `AmazonSESSample` and then choose **Finish**.

7. Replace the entire contents of **AmazonSESSample.java** with the following code:

```
import java.util.Properties;

import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class AmazonSESSample {

    // Replace sender@example.com with your "From" address.
    // This address must be verified.
    static final String FROM = "sender@example.com";
    static final String FROMNAME = "Sender Name";

    // Replace recipient@example.com with a "To" address. If your account
    // is still in the sandbox, this address must be verified.
    static final String TO = "recipient@example.com";

    // Replace smtp_username with your Amazon SES SMTP user name.
    static final String SMTP_USERNAME = "smtp_username";

    // The name of the Configuration Set to use for this message.
    // If you comment out or remove this variable, you will also need to
    // comment out or remove the header below.
    static final String CONFIGSET = "ConfigSet";

    // Amazon SES SMTP host name. This example uses the US West (Oregon) region.
    // See https://docs.aws.amazon.com/ses/latest/DeveloperGuide/
regions.html#region-endpoints
    // for more information.
    static final String HOST = "email-smtp.us-west-2.amazonaws.com";

    // The port you will connect to on the Amazon SES SMTP endpoint.
    static final int PORT = 587;

    static final String SUBJECT = "Amazon SES test (SMTP interface accessed
 using Java)";

    static final String BODY = String.join(
            System.getProperty("line.separator"),
```

```
            "<h1>Amazon SES SMTP Email Test</h1>",
            "<p>This email was sent with Amazon SES using the ",
            "<a href='https://github.com/javaee/javamail'>Javamail Package</a>",
            " for <a href='https://www.java.com'>Java</a>."
        );

    public static void main(String[] args) throws Exception {

        // Create a Properties object to contain connection configuration
information.
        Properties props = System.getProperties();
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.port", PORT);
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.auth", "true");

        // Create a Session object to represent a mail session with the
specified properties.
        Session session = Session.getDefaultInstance(props);

        // Create a message with the specified information.
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(FROM,FROMNAME));
        msg.setRecipient(Message.RecipientType.TO, new InternetAddress(TO));
        msg.setSubject(SUBJECT);
        msg.setContent(BODY,"text/html");

        // Add a configuration set header. Comment or delete the
        // next line if you are not using a configuration set
        msg.setHeader("X-SES-CONFIGURATION-SET", CONFIGSET);

        // Create a transport.
        Transport transport = session.getTransport();

        // Get the password
        String SMTP_PASSWORD = fetchSMTPPasswordFromSecureStorage();

        // Send the message.
        try
        {
            System.out.println("Sending...");

            // Connect to Amazon SES using the SMTP username and password you
specified above.
```

```
            transport.connect(HOST, SMTP_USERNAME, SMTP_PASSWORD);

            // Send the email.
            transport.sendMessage(msg, msg.getAllRecipients());
            System.out.println("Email sent!");
        }
        catch (Exception ex) {
            System.out.println("The email was not sent.");
            System.out.println("Error message: " + ex.getMessage());
        }
        finally
        {
            // Close and terminate the connection.
            transport.close();
        }
    }

    static String fetchSMTPPasswordFromSecureStorage() {
        /* IMPLEMENT THIS METHOD */
        // For example, you might fetch it from a secure location or AWS Secrets
 Manager: https://aws.amazon.com/secrets-manager/
    }
}
```

8. In **AmazonSESSample.java**, replace the following email addresses with your own values:

> **Important**
>
> The email addresses are case-sensitive. Make sure that the addresses are exactly the same as the ones you verified.

- *sender@example.com* – Replace with your "From" email address. You must verify this address before you run this program. For more information, see Verified identities in Amazon SES.
- *recipient@example.com* – Replace with your "To" email address. If your account is still in the sandbox, you must verify this address before you use it. For more information, see Request production access (Moving out of the Amazon SES sandbox).

9. In **AmazonSESSample.java** replace the following with your own values:

- *smtp_username* – Replace with your SMTP user name credential. Note that your SMTP user name credential is a 20-character string of letters and numbers, not an intelligible name.
- *smtp_password* – Implement `fetchSMTPPasswordFromSecureStorage` to fetch the password.

10. (Optional) If you want to use an Amazon SES SMTP endpoint in an AWS Region other than *email-smtp.us-west-2.amazonaws.com*, change the value of the variable HOST to the endpoint you want to use. For a list of regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.
11. (Optional) If you want to use a configuration set when sending this email, change the value of the variable *ConfigSet* to the name of the configuration set. For more information about configuration sets, see Using configuration sets in Amazon SES.
12. Save **AmazonSESSample.java**.
13. To build the project, choose **Project** and then choose **Build Project**. (If this option is disabled, then you may have automatic building enabled.)
14. To start the program and send the email, choose **Run** and then choose **Run** again.
15. Review the output. If the email was successfully sent, the console displays *"Email sent!"* Otherwise, it displays an error message.
16. Sign into the email client of the recipient address. You will see the message that you sent.

PHP

This example uses the PHPMailer class to send email through Amazon SES using the SMTP interface.

Before you perform the following procedure you must complete the tasks in Setting up Amazon Simple Email Service. In addition to setting up Amazon SES you must complete the following prerequisites to sending email with PHP:

Prerequisites:

- **Install PHP** – PHP is available at http://php.net/downloads.php. After you install PHP, add the path to PHP in your environment variables so that you can run PHP from any command prompt.

- **Install the Composer dependency manager** – After you install the Composer dependency manager, you can download and install the PHPMailer class and its dependencies. To install Composer, follow the installation instructions at https://getcomposer.org/download.
- **Install the PHPMailer class** – After you install Composer, run the following command to install PHPMailer:

```
path/to/composer require phpmailer/phpmailer
```

In the preceding command, replace *path/to/* with the path where you installed Composer.

To send an email using the Amazon SES SMTP interface with PHP

1. Create a file named **amazon-ses-smtp-sample.php**. Open the file with a text editor and paste in the following code:

```
<?php

// Import PHPMailer classes into the global namespace
// These must be at the top of your script, not inside a function
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

// If necessary, modify the path in the require statement below to refer to the
// location of your Composer autoload.php file.
require 'vendor/autoload.php';

// Replace sender@example.com with your "From" address.
// This address must be verified with Amazon SES.
$sender = 'sender@example.com';
$senderName = 'Sender Name';

// Replace recipient@example.com with a "To" address. If your account
// is still in the sandbox, this address must be verified.
$recipient = 'recipient@example.com';

// Replace smtp_username with your Amazon SES SMTP user name.
$usernameSmtp = 'smtp_username';

// Specify a configuration set. If you do not want to use a configuration
// set, comment or remove the next line.
$configurationSet = 'ConfigSet';
```

```
// If you're using Amazon SES in a region other than US West (Oregon),
// replace email-smtp.us-west-2.amazonaws.com with the Amazon SES SMTP
// endpoint in the appropriate region.
$host = 'email-smtp.us-west-2.amazonaws.com';
$port = 587;

// The subject line of the email
$subject = 'Amazon SES test (SMTP interface accessed using PHP)';

// The plain-text body of the email
$bodyText =  "Email Test\r\nThis email was sent through the
    Amazon SES SMTP interface using the PHPMailer class.";

// The HTML-formatted body of the email
$bodyHtml = '<h1>Email Test</h1>
    <p>This email was sent through the
    <a href="https://aws.amazon.com/ses">Amazon SES</a> SMTP
    interface using the <a href="https://github.com/PHPMailer/PHPMailer">
    PHPMailer</a> class.</p>';

$mail = new PHPMailer(true);

try {
    // Specify the SMTP settings.
    $mail->isSMTP();
    $mail->setFrom($sender, $senderName);
    $mail->Username   = $usernameSmtp;
    $mail->Password   = fetchSMTPPasswordFromSecureStorage();
    $mail->Host       = $host;
    $mail->Port       = $port;
    $mail->SMTPAuth   = true;
    $mail->SMTPSecure = 'tls';
    $mail->addCustomHeader('X-SES-CONFIGURATION-SET', $configurationSet);

    // Specify the message recipients.
    $mail->addAddress($recipient);
    // You can also add CC, BCC, and additional To recipients here.

    // Specify the content of the message.
    $mail->isHTML(true);
    $mail->Subject    = $subject;
    $mail->Body       = $bodyHtml;
    $mail->AltBody    = $bodyText;
```

```
    $mail->Send();
    echo "Email sent!" , PHP_EOL;
} catch (phpmailerException $e) {
    echo "An error occurred. {$e->errorMessage()}", PHP_EOL; //Catch errors from
 PHPMailer.
} catch (Exception $e) {
    echo "Email not sent. {$mail->ErrorInfo}", PHP_EOL; //Catch errors from
 Amazon SES.
}
function fetchSMTPPasswordFromSecureStorage() {
/* IMPLEMENT THIS METHOD */
// For example, you might fetch it from a secure location or AWS Secrets
 Manager: https://aws.amazon.com/secrets-manager/
}

?>
```

2. In **amazon-ses-smtp-sample.php**, replace the following with your own values:

   - *sender@example.com* – Replace with an email address that you have verified with Amazon SES. For more information, see Verified identities. Email addresses in Amazon SES are case-sensitive. Make sure that the address you enter is exactly the same as the one you verified.
   - *recipient@example.com* – Replace with the address of the recipient. If your account is still in the sandbox, you must verify this address before you use it. For more information, see Request production access (Moving out of the Amazon SES sandbox). Make sure that the address you enter is exactly the same as the one you verified.
   - *smtp_username* – Replace with your SMTP user name credential, which you obtained from the SMTP Settings page of the Amazon SES console. This is **not** the same as your AWS access key ID. Note that your SMTP user name credential is a 20-character string of letters and numbers, not an intelligible name.
   - *smtp_password* – Implement `fetchSMTPPasswordFromSecureStorage` to fetch the password.
   - (Optional) *ConfigSet* – If you want to use a configuration set when sending this email, replace this value with the name of the configuration set. For more information about configuration sets, see Using configuration sets in Amazon SES.
   - (Optional) *email-smtp.us-west-2.amazonaws.com* – If you want to use an Amazon SES SMTP endpoint in a Region other than US West (Oregon), replace this with the Amazon SES SMTP endpoint in the Region you want to use. For a list of SMTP endpoint

URLs for the AWS Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

3. Save **amazon-ses-smtp-sample.php**.
4. To run the program, open a command prompt in the same directory as **amazon-ses-smtp-sample.php**, and then type **php amazon-ses-smtp-sample.php**.
5. Review the output. If the email was successfully sent, the console displays *"Email sent!"* Otherwise, it displays an error message.
6. Sign in to the email client of the recipient address. You will see the message that you sent.

Integrating Amazon SES with your existing email server

If you currently administer your own email server, you can use the Amazon SES SMTP endpoint to send all of your outgoing email to Amazon SES. There is no need to modify your existing email clients and applications; the changeover to Amazon SES will be transparent to them.

Several mail transfer agents (MTAs) support sending email through SMTP relays. This section provides general guidance on how to configure some popular MTAs to send email using Amazon SES SMTP interface.

The Amazon SES SMTP endpoint requires that all connections be encrypted using Transport Layer Security (TLS).

Topics

- Integrating Amazon SES with Postfix
- Integrating Amazon SES with Sendmail
- Integrating Amazon SES with Microsoft Windows Server IIS SMTP

Integrating Amazon SES with Postfix

Postfix is an alternative to the widely used Sendmail Message Transfer Agent (MTA). For information about Postfix, go to http://www.postfix.org. The procedures in this topic will work with Linux, macOS, or Unix.

> **Note**
>
> Postfix is a third-party application, and isn't developed or supported by Amazon Web Services. The procedures in this section are provided for informational purposes only, and are subject to change without notice.

Prerequisites

Before you complete the procedures in this section, you have to perform the following tasks:

- Uninstall the Sendmail application if it's installed on your system. The procedure for completing this step varies depending on the operating system you use.

  > **Important**
  >
  > Following references to *sendmail* refer to the Postfix command `sendmail`, not to be confused with the Sendmail application.

- Install Postfix. The procedure for completing this step varies depending on the operating system you use.
- Install a SASL authentication package. The procedure for completing this step varies depending on the operating system you use. For example, if you use a RedHat-based system, you should install the `cyrus-sasl-plain` package. If you use a Debian- or Ubuntu-based system, you should install the `libsasl2-modules` package.
- Verify an email address or domain to use for sending email. For more information, see Creating an email address identity.
- If your account is still in the sandbox, you can only send email to verified email addresses. For more information, see Request production access (Moving out of the Amazon SES sandbox).

Configuring Postfix

Complete the following procedures to configure your mail server to send email through Amazon SES using Postfix.

To configure Postfix

1. At the command line, type the following command:

```
sudo postconf -e "relayhost = [email-smtp.us-west-2.amazonaws.com]:587" \
"smtp_sasl_auth_enable = yes" \
"smtp_sasl_security_options = noanonymous" \
"smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd" \
"smtp_use_tls = yes" \
"smtp_tls_security_level = secure" \
"smtp_tls_note_starttls_offer = yes"
```

> **Note**
>
> If you use Amazon SES in an AWS Region other than US West (Oregon), replace *email-smtp.us-west-2.amazonaws.com* in the preceding command with the SMTP endpoint of the appropriate Region. For more information, see the section called "Regions".

2. In a text editor, open the file `/etc/postfix/master.cf`. Search for the following entry:

```
-o smtp_fallback_relay=
```

If you find this entry, comment it out by placing a # (hash) character at the beginning of the line. Save and close the file.

Otherwise, if this entry isn't present, continue to the next step.

3. In a text editor, open the file `/etc/postfix/sasl_passwd`. If the file doesn't already exist, create it.
4. Add the following line to `/etc/postfix/sasl_passwd`:

```
[email-smtp.us-west-2.amazonaws.com]:587 SMTPUSERNAME:SMTPPASSWORD
```

> **Note**
>
> Replace *SMTPUSERNAME* and *SMTPPASSWORD* with your SMTP sign-in credentials. Your SMTP sign-in credentials aren't the same as your AWS access key ID and secret access key. For more information about credentials, see the section called "Obtaining SMTP credentials".
>
> If you use Amazon SES in an AWS Region other than US West (Oregon), replace *email-smtp.us-west-2.amazonaws.com* in the preceding example with the SMTP

endpoint of the appropriate Region. For more information, see the section called "Regions".

Save and close `sasl_passwd`.

5. At a command prompt, type the following command to create a hashmap database file containing your SMTP credentials:

```
sudo postmap hash:/etc/postfix/sasl_passwd
```

6. (Optional) The `/etc/postfix/sasl_passwd` and `/etc/postfix/sasl_passwd.db` files you created in the previous steps aren't encrypted. Because these files contain your SMTP credentials, we recommend that you modify the files' ownership and permissions in order to restrict access to them. To restrict access to these files:

   a. At a command prompt, type the following command to change the ownership of the files:

   ```
   sudo chown root:root /etc/postfix/sasl_passwd /etc/postfix/sasl_passwd.db
   ```

   b. At a command prompt, type the following command to change the permissions of the files so that only the root user can read or write to them:

   ```
   sudo chmod 0600 /etc/postfix/sasl_passwd /etc/postfix/sasl_passwd.db
   ```

7. Tell Postfix where to find the CA certificate (needed to verify the Amazon SES server certificate). The command you use in this step varies based on your operating system.

   - If you use Amazon Linux, Red Hat Enterprise Linux, or a related distribution, type the following command:

   ```
   sudo postconf -e 'smtp_tls_CAfile = /etc/ssl/certs/ca-bundle.crt'
   ```

   - If you use Ubuntu or a related distribution, type the following command:

   ```
   sudo postconf -e 'smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt'
   ```

   - If you use macOS, you can generate the certificate from your system keychain. To generate the certificate, type the following command at the command line:

```
sudo security find-certificate -a -p /System/Library/Keychains/
SystemRootCertificates.keychain | sudo tee /etc/ssl/certs/ca-bundle.crt > /dev/
null
```

After you generate the certificate, type the following command:

```
sudo postconf -e 'smtp_tls_CAfile = /etc/ssl/certs/ca-bundle.crt'
```

8. Type the following command to start the Postfix server (or to reload the configuration settings if the server is already running):

```
sudo postfix start; sudo postfix reload
```

9. Send a test email by typing the following at a command line, pressing Enter after each line. Replace *sender@example.com* with your From email address. The From address has to be verified for use with Amazon SES. Replace *recipient@example.com* with the destination address. If your account is still in the sandbox, the recipient address also has to be verified. Finally, the final line of the message has to contain a single period (.) with no other content.

```
sendmail -f sender@example.com recipient@example.com
From: Sender Name <sender@example.com>
Subject: Amazon SES Test
This message was sent using Amazon SES.
.
```

10. Check the mailbox associated with the recipient address. If the email doesn't arrive, check your junk mail folder. If you still can't locate the email, check the mail log on the system that you used to send the email (typically located at `/var/log/maillog`) for more information.

Advanced usage example

This example shows how to send an email that uses a configuration set, and that uses MIME-multipart encoding to send both a plain text and an HTML version of the message, along with an attachment. It also includes a link tag, which can be used for categorizing click events. The content of the email is specified in an external file, so that you do not have to manually type the commands in the Postfix session.

To send a multipart MIME email using Postfix

1. In a text editor, create a new file called `mime-email.txt`.
2. In the text file, paste the following content, replacing the values in red with the appropriate values for your account:

```
X-SES-CONFIGURATION-SET: ConfigSet
From:Sender Name <sender@example.com>
Subject:Amazon SES Test
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="YWVhZDFlY2QzMGQ2N2U0YTZmODU"

--YWVhZDFlY2QzMGQ2N2U0YTZmODU
Content-Type: multipart/alternative; boundary="3NjM0N2QwMTE4MWQ0ZTg2NTYxZQ"

--3NjM0N2QwMTE4MWQ0ZTg2NTYxZQ
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

Amazon SES Test

This message was sent from Amazon SES using the SMTP interface.

For more information, see:
http://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-email-smtp.html

--3NjM0N2QwMTE4MWQ0ZTg2NTYxZQ
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<html>
  <head>
</head>
  <body>
    <h1>Amazon SES Test</h1>
      <p>This message was sent from Amazon SES using the SMTP interface.</p>
      <p>For more information, see
      <a ses:tags="samplekey0:samplevalue0;samplekey1:samplevalue1;"
      href="http://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-email-
smtp.html">
      Using the Amazon SES SMTP Interface to Send Email</a> in the <em>Amazon SES
      Developer Guide</em>.</p>
  </body>
```

```
</html>
--3NjM0N2QwMTE4MWQ0ZTg2NTYxZQ--
--YWVhZDFlY2QzMGQ2N2U0YTZmODU
Content-Type: application/octet-stream
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="customers.txt"

SUQsRmlyc3ROYW1lLExhc3ROYW1lLENvdW50cnkKMzQ4LEpvaG4sU3RpbGVzLENh
bmFkYQo5MjM4OSxKaWUsTGl1LENoaW5hCjczNCxTaGlybGV5LFJvZHJpZ3VleixV
bml0ZWQgU3RhdGVzCjI4OTMsQW5heWEsSXllbmdhcixJbmRpYQ==
--YWVhZDFlY2QzMGQ2N2U0YTZmODU--
```

Save and close the file.

3. At the command line, type the following command. Replace *sender@example.com* with your email address, and replace *recipient@example.com* with the recipient's email address.

```
sendmail -f sender@example.com recipient@example.com < mime-email.txt
```

If the command runs successfully, it exits without providing any output.

4. Check your inbox for the email. If the message wasn't delivered, check your system's mail log.

Integrating Amazon SES with Sendmail

Sendmail was released in the early 1980s, and has been continuously improved ever since. It's a flexible and configurable message transfer agent (MTA) with a large community of users. Sendmail was acquired by Proofpoint in 2013, but Proofpoint continues to offer an open source version of Sendmail. You can download the open source version of Sendmail from the Proofpoint website, or through the package managers of most Linux distributions.

The procedure in this section shows you how to configure Sendmail to send email through Amazon SES. This procedure was tested on a server running Ubuntu 18.04.2 LTS.

> **Note**
>
> Sendmail is a third-party application, and isn't developed or supported by Amazon Web Services. The procedures in this section are provided for informational purposes only, and are subject to change without notice.

Prerequisites

Before you complete the procedure in this section, you should complete the following steps:

- Install the Sendmail package on your server.

  > **Note**
  >
  > Depending on which operating system distribution you use, you might also need to install the following packages: `sendmail-cf`, `m4`, and `cyrus-sasl-plain`.

- Verify an identity to use as your From address. For more information, see Creating an email address identity.

  If your account is in the Amazon SES sandbox, you must also verify the addresses that you send email to. For more information, see Request production access (Moving out of the Amazon SES sandbox).

If you're using Amazon SES to send email from an Amazon EC2 instance, you should also complete the following steps:

- You may need to assign an Elastic IP Address to your Amazon EC2 instance in order for receiving email providers to accept your email. For more information, see Amazon EC2 Elastic IP addresses in the *Amazon EC2 User Guide*.
- Amazon Elastic Compute Cloud (Amazon EC2) restricts email traffic over port 25 by default. To avoid timeouts when sending email through the SMTP endpoint from Amazon EC2, you can request that these restrictions be removed. For more information, see How do I remove the restriction on port 25 from my Amazon EC2 instance or AWS Lambda function? in the AWS Knowledge Center.

  Alternatively, you can modify the procedure in this section to use port 587 rather than port 25.

Configuring Sendmail

Complete the steps in this section to configure Sendmail to send email by using Amazon SES.

Important

The procedure in this section assumes that you want to use Amazon SES in the US West (Oregon) AWS Region. If you want to use a different Region, replace all instances of *email-smtp.us-west-2.amazonaws.com* in this procedure with the SMTP endpoint of the desired Region. For a list of SMTP endpoint URLs for the AWS Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

To configure Sendmail

1. In a file editor, open the file `/etc/mail/authinfo`. If the file doesn't exist, create it.

   Add the following line to */etc/mail/authinfo*:

   ```
   AuthInfo:email-smtp.us-west-2.amazonaws.com "U:root" "I:smtpUsername"
    "P:smtpPassword" "M:PLAIN"
   ```

   In the preceding example, make the following changes:

   - Replace *email-smtp.us-west-2.amazonaws.com* with the Amazon SES SMTP endpoint that you want to use.
   - Replace *smtpUsername* with your Amazon SES SMTP user name.
   - Replace *smtpPassword* with your Amazon SES SMTP password.

   Note

   Your SMTP sign-in credentials are different from your AWS Access Key ID and Secret Access Key. For more information about obtaining your SMTP sign-in credentials, see Obtaining Amazon SES SMTP credentials.

   When you finish, save `authinfo`.

2. At the command line, enter the following command to generate the `/etc/mail/authinfo.db` file:

```
sudo sh -c 'makemap hash /etc/mail/authinfo.db < /etc/mail/authinfo'
```

3. At the command line, type the following command to add support for relaying to the Amazon SES SMTP endpoint.

```
sudo sh -c 'echo "Connect:email-smtp.us-west-2.amazonaws.com RELAY" >> /etc/mail/
access'
```

   In the preceding command, replace *email-smtp.us-west-2.amazonaws.com* with the address of the Amazon SES SMTP endpoint that you want to use.

4. At the command line, type the following command to regenerate */etc/mail/access.db*:

```
sudo sh -c 'makemap hash /etc/mail/access.db < /etc/mail/access'
```

5. At the command line, type the following command to create backups of the `sendmail.cf` and `sendmail.mc` files:

```
sudo sh -c 'cp /etc/mail/sendmail.cf /etc/mail/sendmail_cf.backup && cp /etc/mail/
sendmail.mc /etc/mail/sendmail_mc.backup'
```

6. Add the following lines to the */etc/mail/sendmail.mc* file before any `MAILER()` definitions.

```
define(`SMART_HOST', `email-smtp.us-west-2.amazonaws.com')dnl
define(`RELAY_MAILER_ARGS', `TCP $h 25')dnl
define(`confAUTH_MECHANISMS', `LOGIN PLAIN')dnl
FEATURE(`authinfo', `hash -o /etc/mail/authinfo.db')dnl
MASQUERADE_AS(`example.com')dnl
FEATURE(masquerade_envelope)dnl
FEATURE(masquerade_entire_domain)dnl
```

   In the preceding text, do the following:

   - Replace *email-smtp.us-west-2.amazonaws.com* with the Amazon SES SMTP endpoint that you want to use.
   - Replace *example.com* with the domain that you want to use to send email.

   When you finish, save the file.

> **Note**
>
> Amazon EC2 restricts communications over port 25 by default. If you're using Sendmail in an Amazon EC2 instance, you should complete the Request to Remove Email Sending Limitations.

7. At the command line, type the following command to make *sendmail.cf* writeable:

```
sudo chmod 666 /etc/mail/sendmail.cf
```

8. At the command line, type the following command to regenerate *sendmail.cf*:

```
sudo sh -c 'm4 /etc/mail/sendmail.mc > /etc/mail/sendmail.cf'
```

> **Note**
>
> If you encounter errors such as "Command not found" and "No such file or directory," make sure that the `m4` and `sendmail-cf` packages are installed on your system.

9. At the command line, type the following command to reset the permissions of *sendmail.cf* to read only:

```
sudo chmod 644 /etc/mail/sendmail.cf
```

10. At the command line, type the following command to restart Sendmail:

```
sudo /etc/init.d/sendmail restart
```

Depending on the version of Linux or Sendmail, if the above doesn't work, try the following:

```
sudo su service sendmail restart
```

11. Complete the following steps to send a test email:

    a. At the command line, enter the following command.

```
/usr/sbin/sendmail -vf sender@example.com recipient@example.com
```

Replace *sender@example.com* with your From email address. Replace *recipient@example.com* with the To address. When you finish, press **Enter**.

b. Enter the following message content. Press **Enter** at the end of each line.

```
From: sender@example.com
To: recipient@example.com
Subject: Amazon SES test email

This is a test message sent from Amazon SES using Sendmail.
```

When you finish entering the content of the email, press **Ctrl+D** to send it.

12. Check the recipient email's client for the email. If you can't find the email, check the junk mail folder. If you still can't find the email, check the Sendmail log on your mail server. The log is often located at */var/log/mail.log* or */var/log/maillog*.

Integrating Amazon SES with Microsoft Windows Server IIS SMTP

You can configure Microsoft Windows Server's IIS SMTP server to send email through Amazon SES. These instructions were written using Microsoft Windows Server 2022 on an Amazon EC2 instance. You can use the same configuration on Microsoft Windows Server 2016.

> **Note**
>
> Windows Server is a third-party application, and isn't developed or supported by Amazon Web Services. The procedures in this section are provided for informational purposes only, and are subject to change without notice.

To integrate the Microsoft Windows Server IIS SMTP server with Amazon SES

1. First, set up Microsoft Windows Server 2022 using the following instructions.

   a. From the Amazon EC2 management console, launch a new Microsoft Windows Server 2022 Base Amazon EC2 instance.

   b. Connect to the instance and log into it using Remote Desktop by following the instructions in Getting Started with Amazon EC2 Windows Instances.

   c. Launch the Server Manager Dashboard.

d. Install the **Web Server** role. Be sure to include the **IIS 10 Management Compatibility tools** (an option under the **Web Server** check box).

e. Install the **SMTP Server** feature.

2. Next, configure the IIS SMTP service using the following instructions.

a. Return to the Server Manager Dashboard.

b. From the **Tools** menu, choose **Internet Information Services (IIS) 10.0 Manager**.

c. Right-click **SMTP Virtual Server #1** and then select **Properties**.

d. On the **Access** tab, under **Relay Restrictions**, choose **Relay**.

e. In the **Relay Restrictions** dialog box, choose **Add**.

f. Under **Single Computer**, enter **127.0.0.1** for the IP address. You have now granted access for this server to relay email to Amazon SES through the IIS SMTP service.

In this procedure, we assume that your emails are generated on this server. If the application that generates the email runs on a separate server, you must grant relaying access for that server in IIS SMTP.

> **Note**
>
> To extend the SMTP relay to private subnets, for **Relay Restriction**, use **Single Computer** 127.0.0.1 and **Group of Computers** 172.1.1.0 - 255.255.255.0 (in the netmask section). For **Connection**, use **Single Computer** 127.0.0.1 and **Group of Computers** 172.1.1.0 - 255.255.255.0 (in the netmask section).

3. Finally, configure the server to send email through Amazon SES using the following instructions.

a. Return to the **SMTP Virtual Server #1 Properties** dialog box and then choose the **Delivery** tab.

b. On the **Delivery** tab, choose **Outbound Security**.

c. Select **Basic Authentication** and then enter your Amazon SES SMTP credentials. You can obtain these credentials from the Amazon SES console using the procedure in Obtaining Amazon SES SMTP credentials.

> **Important**
>
> Your SMTP credentials are not the same as your AWS access key ID and secret access key. Do not attempt to use your AWS credentials to authenticate yourself against the SMTP endpoint. For more information about credentials, see Types of Amazon SES credentials.

d. Ensure that **TLS encryption** is selected.

e. Return to the **Delivery** tab.

f. Choose **Outbound Connections**.

g. In the **Outbound Connections** dialog box, ensure that the port is 25 or 587.

h. Choose **Advanced**.

i. For the **Smart host** name, enter the Amazon SES endpoint that you will use (for example, *email-smtp.us-west-2.amazonaws.com*). For a list of endpoint URLs for the AWS Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

j. Return to the Server Manager Dashboard.

k. On the Server Manager Dashboard, right-click **SMTP Virtual Server #1** and then restart the service to pick up the new configuration.

l. Send an email through this server. You can examine the message headers to confirm that it was delivered through Amazon SES.

Testing your connection to the Amazon SES SMTP interface using the command line

You can use the methods described in this section from the command line to test your connection to the Amazon SES SMTP endpoint, validate your SMTP credentials, and troubleshoot connection issues. These procedures use tools and libraries that are included with most common operating systems.

For additional information about troubleshooting SMTP connection problems, see Amazon SES SMTP issues.

Prerequisites

When you connect to the Amazon SES SMTP interface, you have to provide a set of SMTP credentials. These SMTP credentials are different from your standard AWS credentials. The two types of credentials aren't interchangeable. For more information about obtaining your SMTP credentials, see the section called “Obtaining SMTP credentials”.

Testing your connection to the Amazon SES SMTP interface

You can use the command line to test your connection to the Amazon SES SMTP interface without authenticating or sending any messages. This procedure is useful for troubleshooting basic connectivity issues. If your test connection fails, see SMTP issues.

This section includes procedures for testing your connection using both OpenSSL (which is included with most Linux, macOS, and Unix distributions, and is also available for Windows) and the `Test-NetConnection` cmdlet in PowerShell (which is included with most recent versions of Windows).

Linux, macOS, or Unix

There are two ways to connect to the Amazon SES SMTP interface with OpenSSL: using explicit SSL over port 587, or using implicit SSL over port 465.

To connect to the SMTP interface using explicit SSL

- At the command line, enter the following command to connect to the Amazon SES SMTP server:

```
openssl s_client -crlf -quiet -starttls smtp -connect email-smtp.us-west-2.amazonaws.com:587
```

In the preceding command, replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region. For more information, see the section called “Regions”.

If the connection was successful, you see output similar to the following:

```
depth=2 C = US, O = Amazon, CN = Amazon Root CA 1
verify return:1
depth=1 C = US, O = Amazon, OU = Server CA 1B, CN = Amazon
verify return:1
```

```
depth=0 CN = email-smtp.us-west-2.amazonaws.com
verify return:1
250 Ok
```

The connection automatically closes after about 10 seconds of inactivity.

Alternatively, you can use implicit SSL to connect to the SMTP interface over port 465.

To connect to the SMTP interface using implicit SSL

- At the command line, enter the following command to connect to the Amazon SES SMTP server:

  ```
  openssl s_client -crlf -quiet -connect email-smtp.us-west-2.amazonaws.com:465
  ```

  In the preceding command, replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region. For more information, see the section called “Regions”.

  If the connection was successful, you see output similar to the following:

  ```
  depth=2 C = US, O = Amazon, CN = Amazon Root CA 1
  verify return:1
  depth=1 C = US, O = Amazon, OU = Server CA 1B, CN = Amazon
  verify return:1
  depth=0 CN = email-smtp.us-west-2.amazonaws.com
  verify return:1
  220 email-smtp.amazonaws.com ESMTP SimpleEmailService-d-VCSHDP1YZ
   A1b2C3d4E5f6G7h8I9j0
  ```

  The connection automatically closes after about 10 seconds of inactivity.

PowerShell

You can use the Test-NetConnection cmdlet in PowerShell to connect to the Amazon SES SMTP server.

> **Note**
>
> The `Test-NetConnection` cmdlet can determine whether your computer can connect to the Amazon SES SMTP endpoint. However, it doesn't test whether your computer can make an implicit or explicit SSL connection to the SMTP endpoint. To test an SSL connection, you can install OpenSSL for Windows to send a test email.

To connect to the SMTP interface using the `Test-NetConnection` cmdlet

- In PowerShell, enter the following command to connect to the Amazon SES SMTP server:

  ```
  Test-NetConnection -Port 587 -ComputerName email-smtp.us-west-2.amazonaws.com
  ```

  In the preceding command, replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region, and replace *587* with the port number. For more information about regional endpoints in Amazon SES, see the section called “Regions”.

  If the connection was successful, you see output that resembles the following example:

  ```
  ComputerName     : email-smtp.us-west-2.amazonaws.com
  RemoteAddress    : 198.51.100.126
  RemotePort       : 587
  InterfaceAlias   : Ethernet
  SourceAddress    : 203.0.113.46
  TcpTestSucceeded : True
  ```

Using the command line to send email using the Amazon SES SMTP interface

You can also use the command line to send messages using the Amazon SES SMTP interface. This procedure is useful for testing SMTP credentials and for testing the ability of specific recipients to receive messages that you send by using Amazon SES.

Linux, macOS, or Unix

When an email sender connects to an SMTP server, the client issues a standard set of requests, and the server replies to each request with a standard response. This series of requests and

responses is called an *SMTP conversation*. When you connect to the Amazon SES SMTP server using OpenSSL, the server expects an SMTP conversation to occur.

When you use OpenSSL to connect to the SMTP interface, you have to encode your SMTP credentials using base64 encoding. This section includes procedures for encoding your credentials using base64.

To send an email from the command line using the SMTP interface

1. Enter the following at the command line and replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region. For more information, see the section called “Regions”.:

```
#!/bin/bash

# Prompt user to provide following information
read -p "Configuration set: " CONFIGSET
read -p "Enter SMTP username: " SMTPUsername
read -p "Enter SMTP password: " SMTPPassword
read -p "Sender email address: " MAILFROM
read -p "Receiver email address: " RCPT
read -p "Email subject: " SUBJECT
read -p "Message to send: " DATA

echo

# Encode SMTP username and password using base64
EncodedSMTPUsername=$(echo -n "$SMTPUsername" | openssl enc -base64)
EncodedSMTPPassword=$(echo -n "$SMTPPassword" | openssl enc -base64)

# Construct the email
Email="EHLO example.com
AUTH LOGIN
$EncodedSMTPUsername
$EncodedSMTPPassword
MAIL FROM: $MAILFROM
RCPT TO: $RCPT
DATA
X-SES-CONFIGURATION-SET: $CONFIGSET
From: $MAILFROM
To: $RCPT
Subject: $SUBJECT
```

```
$DATA
.
QUIT"

echo "$Email" | openssl s_client -crlf -quiet -starttls smtp -connect email-smtp.us-west-2.amazonaws.com:587
```

2. At the prompt for each variable, enter your values.
3. • To send using implicit SSL over port 465, use:

```
openssl s_client -crlf -quiet -connect email-smtp.us-west-2.amazonaws.com:465
```

If the message was accepted by Amazon SES, you see output that resembles the following example:

```
250 Ok 01010160d7de98d8-21e57d9a-JZho-416c-bbe1-8ebaAexample-000000
```

The string of numbers and text that follows `250 Ok` is the message ID of the email.

> **Note**
>
> The connection closes automatically after about 10 seconds of inactivity.

PowerShell

You can use the Net.Mail.SmtpClient class to send email using explicit SSL over port 587.

> **Note**
>
> The `Net.Mail.SmtpClient` class is officially obsolete, and Microsoft recommends that you use third-party libraries. This code is intended for testing purposes only, and shouldn't be used for production workloads.

To send an email through PowerShell using explicit SSL

1. In a text editor, create a new file. Paste the following code into the file:

```
function SendEmail($Server, $Port, $Sender, $Recipient, $Subject, $Body) {
    $Credentials = [Net.NetworkCredential](Get-Credential)

    $SMTPClient = New-Object Net.Mail.SmtpClient($Server, $Port)
    $SMTPClient.EnableSsl = $true
    $SMTPClient.Credentials = New-Object
 System.Net.NetworkCredential($Credentials.Username, $Credentials.Password);

    try {
        Write-Output "Sending message..."
        $SMTPClient.Send($Sender, $Recipient, $Subject, $Body)
        Write-Output "Message successfully sent to $($Recipient)"
    } catch [System.Exception] {
        Write-Output "An error occurred:"
        Write-Error $_
    }
}

function SendTestEmail(){
    $Server = "email-smtp.us-west-2.amazonaws.com"
    $Port = 587

    $Subject = "Test email sent from Amazon SES"
    $Body = "This message was sent from Amazon SES using PowerShell (explicit
 SSL, port 587)."

    $Sender = "sender@example.com"
    $Recipient = "recipient@example.com"

    SendEmail $Server $Port $Sender $Recipient $Subject $Body
}

SendTestEmail
```

When you finish, save the file as `SendEmail.ps1`.

2. Make the following changes to the file that you created in the previous step:

   - Replace *sender@example.com* with the email address that you want to send the message from.
   - Replace *recipient@example.com* with the email address that you want to send the message to.

- Replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region. For more information, see Regions and Amazon SES.

3. In PowerShell, enter the following command:

```
.\path\to\SendEmail.ps1
```

   In the preceding command, replace *path\to\SendEmail.ps1* with the path to the file that you created in step 1.

4. When prompted, enter your SMTP user name and password.

Alternatively, you can use the System.Web.Mail.SmtpMail class to send email using implicit SSL over port 465.

> **Note**
>
> The `System.Web.Mail.SmtpMail` class is officially obsolete, and Microsoft recommends that you use third-party libraries. This code is intended for testing purposes only, and shouldn't be used for production workloads.

To send an email through PowerShell using implicit SSL

1. In a text editor, create a new file. Paste the following code into the file:

```
[System.Reflection.Assembly]::LoadWithPartialName("System.Web") > $null

function SendEmail($Server, $Port, $Sender, $Recipient, $Subject, $Body) {
    $Credentials = [Net.NetworkCredential](Get-Credential)

    $mail = New-Object System.Web.Mail.MailMessage
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
smtpserver", $Server)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
smtpserverport", $Port)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
smtpusessl", $true)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
sendusername", $Credentials.UserName)
```

```
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
sendpassword", $Credentials.Password)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
smtpconnectiontimeout", $timeout / 1000)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/sendusing",
 2)
    $mail.Fields.Add("http://schemas.microsoft.com/cdo/configuration/
smtpauthenticate", 1)

    $mail.From = $Sender
    $mail.To = $Recipient
    $mail.Subject = $Subject
    $mail.Body = $Body

    try {
        Write-Output "Sending message..."
        [System.Web.Mail.SmtpMail]::Send($mail)
        Write-Output "Message successfully sent to $($Recipient)"
    } catch [System.Exception] {
        Write-Output "An error occurred:"
        Write-Error $_
    }
}

function SendTestEmail(){
    $Server = "email-smtp.us-west-2.amazonaws.com"
    $Port = 465

    $Subject = "Test email sent from Amazon SES"
    $Body = "This message was sent from Amazon SES using PowerShell (implicit
 SSL, port 465)."

    $Sender = "sender@example.com"
    $Recipient = "recipient@example.com"

    SendEmail $Server $Port $Sender $Recipient $Subject $Body
}

SendTestEmail
```

When you finish, save the file as `SendEmail.ps1`.

2. Make the following changes to the file that you created in the previous step:

- Replace *sender@example.com* with the email address that you want to send the message from.
- Replace *recipient@example.com* with the email address that you want to send the message to.
- Replace *email-smtp.us-west-2.amazonaws.com* with the URL of the Amazon SES SMTP endpoint for your AWS Region. For more information, see Regions and Amazon SES.

3. In PowerShell, enter the following command:

```
.\path\to\SendEmail.ps1
```

   In the preceding command, replace *path\to\SendEmail.ps1* with the path to the file that you created in step 1.

4. When prompted, enter your SMTP user name and password.

Using the Amazon SES API to send email

To send production email through Amazon SES, you can use the Simple Mail Transfer Protocol (SMTP) interface or the Amazon SES API. For more information about the SMTP interface, see Using the Amazon SES SMTP interface to send email. This section describes how to send email by using the API.

When you send an email using the Amazon SES API, you specify the content of the message, and Amazon SES assembles a MIME email for you. Alternatively, you can assemble the email yourself so that you have complete control over the content of the message. For more information about the API, see the Amazon Simple Email Service API Reference. For a list of endpoint URLs for the AWS Regions where Amazon SES is available, see Amazon Simple Email Service endpoints and quotas in the *AWS General Reference*.

You can call the API in the following ways:

- **Make direct HTTPS requests—**This is the most advanced method, because you have to manually handle authentication and signing of your requests, and then manually construct the requests. For information about the Amazon SES API, see the Welcome page in the *API v2 Reference*.
- **Use an AWS SDK—**AWS SDKs make it easy to access the APIs for several AWS services, including Amazon SES. When you use an SDK, it takes care of authentication, request signing, retry logic,

error handling, and other low-level functions so that you can focus on building applications that delight your customers.

- **Use a command line interface—**The AWS Command Line Interface is the command line tool for Amazon SES. We also offer the AWS Tools for PowerShell for those who script in the PowerShell environment.

Regardless of whether you access the Amazon SES API directly or indirectly through an AWS SDK, the AWS Command Line Interface or the AWS Tools for PowerShell, the Amazon SES API provides two different ways for you to send an email, depending on how much control you want over the composition of the email message:

- **Formatted—**Amazon SES composes and sends a properly formatted email message. You need only supply "From:" and "To:" addresses, a subject, and a message body. Amazon SES takes care of all the rest. For more information, see Sending formatted email using the Amazon SES API.
- **Raw—**You manually compose and send an email message, specifying your own email headers and MIME types. If you're experienced in formatting your own email, the raw interface gives you more control over the composition of your message. For more information, see Sending raw email using the Amazon SES API v2.

Contents

- Sending formatted email using the Amazon SES API
- Sending raw email using the Amazon SES API v2
- Using templates to send personalized email with the Amazon SES API
- Sending email through Amazon SES using an AWS SDK
- Content encodings supported by Amazon SES

Sending formatted email using the Amazon SES API

You can send a formatted email by using the AWS Management Console or by calling the Amazon SES API through an application directly, or indirectly through an AWS SDK, the AWS Command Line Interface, or the AWS Tools for Windows PowerShell.

The Amazon SES API provides the `SendEmail` action, which lets you compose and send a formatted email. `SendEmail` requires a From: address, To: address, message subject, and message

body—text, HTML, or both. For more information, see SendEmail (API Reference) or SendEmail (API v2 Reference).

> **Note**
>
> The email address string must be 7-bit ASCII. If you want to send to or from email addresses that contain Unicode characters in the domain part of an address, you must encode the domain using Punycode. For more information, see RFC 3492.

For examples of how to compose a formatted message using various programming languages, see Code examples.

For tips on how to increase your email sending speed when you make multiple calls to `SendEmail`, see Increasing throughput with Amazon SES.

Sending raw email using the Amazon SES API v2

You can use the Amazon SES API v2 `SendEmail` operation with the content type specified as `raw` to send customized messages to your recipients using the raw email format.

About email header fields

Simple Mail Transfer Protocol (SMTP) specifies how email messages are to be sent by defining the mail envelope and some of its parameters, but it does not concern itself with the content of the message. Instead, the Internet Message Format (RFC 5322) defines how the message is to be constructed.

With the Internet Message Format specification, every email message consists of a header and a body. The header consists of message metadata, and the body contains the message itself. For more information about email headers and bodies, see Email format in Amazon SES.

Using raw email MIME message construction

The SMTP protocol was originally designed to send email messages that only contained 7-bit ASCII characters. This specification makes SMTP insufficient for non-ASCII text encodings (such as Unicode), binary content, or attachments. The Multipurpose Internet Mail Extensions standard (MIME) was developed to make it possible to send many other kinds of content using SMTP.

The MIME standard works by breaking the message body into multiple parts and then specifying what is to be done with each part. For example, one part of an email message body might be plain

text, while another might be HTML. In addition, MIME allows email messages to contain one or more attachments. Message recipients can view the attachments from within their email clients, or they can save the attachments.

The message header and content are separated by a blank line. Each part of the email is separated by a boundary, a string of characters that denotes the beginning and ending of each part.

The multipart message in the following example contains a text and an HTML part, and an attachment. The attachment should be placed just below the attachment headers and is most often encoded in `base64` as shown in this example.

```
From: "Sender Name" <sender@example.com>
To: recipient@example.com
Subject: Customer service contact info
Content-Type: multipart/mixed;
    boundary="a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a"

--a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a
Content-Type: multipart/alternative;
    boundary="sub_a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a"

--sub_a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

Please see the attached file for a list of customers to contact.

--sub_a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a
Content-Type: text/html; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

<html>
<head></head>
<body>
<h1>Hello!</h1>
<p>Please see the attached file for a list of customers to contact.</p>
</body>
</html>

--sub_a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a--

--a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a
Content-Type: text/plain; name="customers.txt"
```

```
Content-Description: customers.txt
Content-Disposition: attachment;filename="customers.txt";
    creation-date="Sat, 05 Aug 2017 19:35:36 GMT";
Content-Transfer-Encoding: base64

SUQsRmlyc3ROYW1lLExhc3ROYW1lLENvdW50cnkKMzQ4LEpvaG4sU3RpbGVzLENhbmFkYQo5MjM4
OSxKaWUsTGl1LENoaW5hCjczNCxTaGlybGV5LFJvZHJpZ3VleixVbml0ZWQgU3RhdGVzCjI4OTMs
QW5heWEsSXllbmdhcixJbmRpYQ==

--a3f166a86b56ff6c37755292d690675717ea3cd9de81228ec2b76ed4a15d6d1a--
```

The content type for the message is `multipart/mixed`, which indicates that the message has many parts (in this example, a body and an attachment), and the receiving client must handle each part separately.

Nested within the body section is a second part that uses the `multipart/alternative` content type. This content type indicates that each part contains alternative versions of the same content (in this case, a text version and an HTML version). If the recipient's email client can display HTML content, then it shows the HTML version of the message body. If the recipient's email client can't display HTML content, then it shows the plain text version of the message body.

Both versions of the message also contain an attachment (in this case, a short text file that contains some customer names).

When you nest a MIME part within another part, as in this example, the nested part must use a `boundary` parameter that is distinct from the `boundary` parameter in the parent part. These boundaries should be unique strings of characters. To define a boundary between MIME parts, type two hyphens (--) followed by the boundary string. At the end of a MIME part, place two hyphens at both the beginning and the end of the boundary string.

> **Note**
>
> A message cannot have more than 500 MIME parts.

MIME Encoding

To maintain compatibility with older systems, Amazon SES honors the 7-bit ASCII limitation of SMTP as defined in RFC 2821. If you want to send content that contains non-ASCII characters, you must encode those characters into a format that uses 7-bit ASCII characters.

Email addresses

The email address string must be 7-bit ASCII. If you want to send to or from email addresses that contain Unicode characters in the domain part of an address, you must encode the domain using Punycode. Punycode is not permitted in the local part of the email address (the part before the @ sign) nor in the "friendly from" name. If you want to use Unicode characters in the "friendly from" name, you must encode the "friendly from" name using MIME encoded-word syntax, as described in this section. For more information about Punycode, see RFC 3492.

> **Note**
>
> This rule only applies to email addresses that you specify in the message envelope, not the message headers. When you use the Amazon SES API v2 `SendEmail` operation, the addresses you specify in the `Source` and `Destinations` parameters define the envelope sender and recipients, respectively.

Email headers

To encode a message header, use MIME encoded-word syntax. MIME encoded word syntax uses the following format:

```
=?charset?encoding?encoded-text?=
```

The value of *encoding* can be either Q or B. If the value of encoding is Q, then the value *encoded-text* has to use Q-encoding. If the value of encoding is B, then the value of *encoded-text* has to use base64 encoding.

For example, if you want to use the string "Як ти поживаєш?" in the subject line of an email, you can use either of the following encodings:

- **Q-encoding**

```
=?utf-8?Q?
=D0=AF=D0=BA_=D1=82=D0=B8_=D0=BF=D0=BE=D0=B6=D0=B8=D0=B2=D0=B0=D1=94=D1=88=3F?=
```

- **Base64 encoding**

```
=?utf-8?B?0K/QuiDRgtC4INC/0L7QttC40LLQsNGU0Yg/?=
```

For more information about Q-encoding, see RFC 2047. For more information about base64 encoding, see RFC 2045.

Message body

To encode the body of a message, you can use quoted-printable encoding or base64 encoding. Then, use the `Content-Transfer-Encoding` header to indicate which encoding scheme you used.

For example, assume the body of your message contains the following text:

१९७२ मे रे टॉमलंसिन ने पहला ई-मेल सेंदश भेजा | रे टॉमलंसिन ने ही स्रव्परथम @ चन्हि का चयन कयिा और इनही को ईमल का आवषिकारक माना जाता है

If you choose to encode this text using base64 encoding, first specify the following header:

```
Content-Transfer-Encoding: base64
```

Then, in the body section of the email, include the base64-encoded text:

```
4KWn4KWv4KWt4KWoIOCkruClhyDgpLDgpYcg4KSf4KWJ4KSu4KSy4KS/4KSC4KS44KSoIOCkqOCl
hyDgpKrgpLngpLLgpL4g4KSILeCkruClh+CksiDgpLjgpILgpKbgpYfgpLYg4KSt4KWH4KSc4KS+
IHwg4KSw4KWHIOCkn+ClieCkruCksuCkv+CkguCkuOCkqCDgpKjgpYcg4KS54KWAIOCkuOCksOCl
jeCkteCkquCljeCksOCkpeCkriBAIOCkmuCkv+CkqOCljeCkuSDgpJXgpL4g4KSa4KSv4KSoIOCk
leCkv+Ckr+CkviDgpJTgpLAg4KSH4KSo4KWN4KS54KWAIOCkleCliyDgpIjgpK7gpYfgpLIg4KSV
4KS+IOCkhuCkteCkv+Ckt+CljeCkleCkvuCksOCklSDgpK7gpL7gpKjgpL4g4KSc4KS+4KSk4KS+
IOCkueCliAo=
```

> **Note**
>
> In some cases, you can use the 8bit `Content-Transfer-Encoding` in messages that you send using Amazon SES. However, if Amazon SES has to make any changes to your messages (for example, when you use open and click tracking), 8-bit-encoded content might not appear correctly when it arrives in recipients' inboxes. For this reason, you should always encode content that isn't 7-bit ASCII.

File attachments

To attach a file to an email, you have to encode the attachment using base64 encoding. Attachments are typically placed in dedicated MIME message parts, which include the following headers:

- **Content-Type** – The file type of the attachment. The following are examples of common MIME Content-Type declarations:
  - **Plain text file** – `Content-Type: text/plain; name="sample.txt"`
  - **Microsoft Word Document** – `Content-Type: application/msword; name="document.docx"`
  - **JPG image** – `Content-Type: image/jpeg; name="photo.jpeg"`
- **Content-Disposition** – Specifies how the recipient's email client should handle the content. For attachments, this value is `Content-Disposition: attachment`.
- **Content-Transfer-Encoding** – The scheme that was used to encode the attachment. For file attachments, this value is almost always `base64`.
- **The encoded attachment** – You must encode the actual attachment and include it in the body below the attachment headers as shown in the example.

Amazon SES accepts most common file types. For a list of file types that Amazon SES doesn't accept, see SES unsupported attachment types.

Sending raw email using the Amazon SES API v2

The Amazon SES API v2 provides the `SendEmail` action, which lets you compose and send an email message in the format that you specify when you set the content type to either simple, raw, or templated. For a complete description, see `SendEmail`. The following example will specify the content type as `raw` to send a messages using the raw email format.

> **Note**
>
> For tips on how to increase your email sending speed when you make multiple calls to `SendEmail`, see Increasing throughput with Amazon SES.

The message body must contain a properly formatted, raw email message, with appropriate header fields and message body encoding. Although it's possible to construct the raw message manually within an application, it's much easier to do so using existing mail libraries.

Java

The following code example shows how to use the JavaMail library and the AWS SDK for Java to compose and send a raw email.

```
package com.amazonaws.samples;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.ByteBuffer;
import java.util.Properties;

// JavaMail libraries. Download the JavaMail API
// from https://javaee.github.io/javamail/
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

// AWS SDK libraries. Download the AWS SDK for Java
// from sdk-for-java
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleemail.AmazonSimpleEmailService;
import com.amazonaws.services.simpleemail.AmazonSimpleEmailServiceClientBuilder;
import com.amazonaws.services.simpleemail.model.RawMessage;
import com.amazonaws.services.simpleemail.model.SendRawEmailRequest;

public class AmazonSESSample {

 // Replace sender@example.com with your "From" address.
 // This address must be verified with Amazon SES.
```

```
private static String SENDER = "Sender Name <sender@example.com>";

// Replace recipient@example.com with a "To" address. If your account
// is still in the sandbox, this address must be verified.
private static String RECIPIENT = "recipient@example.com";

// Specify a configuration set. If you do not want to use a configuration
// set, comment the following variable, and the
// ConfigurationSetName=CONFIGURATION_SET argument below.
private static String CONFIGURATION_SET = "ConfigSet";

// The subject line for the email.
private static String SUBJECT = "Customer service contact info";

// The full path to the file that will be attached to the email.
// If you're using Windows, escape backslashes as shown in this variable.
private static String ATTACHMENT = "C:\\Users\\sender\\customers-to-contact.xlsx";

// The email body for recipients with non-HTML email clients.
private static String BODY_TEXT = "Hello,\r\n"
                                  + "Please see the attached file for a list "
                                  + "of customers to contact.";

// The HTML body of the email.
private static String BODY_HTML = "<html>"
                                  + "<head></head>"
                                  + "<body>"
                                  + "<h1>Hello!</h1>"
                                  + "<p>Please see the attached file for a "
                                  + "list of customers to contact.</p>"
                                  + "</body>"
                                  + "</html>";

   public static void main(String[] args) throws AddressException,
MessagingException, IOException {

    Session session = Session.getDefaultInstance(new Properties());

       // Create a new MimeMessage object.
       MimeMessage message = new MimeMessage(session);

       // Add subject, from and to lines.
       message.setSubject(SUBJECT, "UTF-8");
       message.setFrom(new InternetAddress(SENDER));
```

```
        message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(RECIPIENT));

        // Create a multipart/alternative child container.
        MimeMultipart msg_body = new MimeMultipart("alternative");

        // Create a wrapper for the HTML and text parts.
        MimeBodyPart wrap = new MimeBodyPart();

        // Define the text part.
        MimeBodyPart textPart = new MimeBodyPart();
        textPart.setContent(BODY_TEXT, "text/plain; charset=UTF-8");

        // Define the HTML part.
        MimeBodyPart htmlPart = new MimeBodyPart();
        htmlPart.setContent(BODY_HTML,"text/html; charset=UTF-8");

        // Add the text and HTML parts to the child container.
        msg_body.addBodyPart(textPart);
        msg_body.addBodyPart(htmlPart);

        // Add the child container to the wrapper object.
        wrap.setContent(msg_body);

        // Create a multipart/mixed parent container.
        MimeMultipart msg = new MimeMultipart("mixed");

        // Add the parent container to the message.
        message.setContent(msg);

        // Add the multipart/alternative part to the message.
        msg.addBodyPart(wrap);

        // Define the attachment
        MimeBodyPart att = new MimeBodyPart();
        DataSource fds = new FileDataSource(ATTACHMENT);
        att.setDataHandler(new DataHandler(fds));
        att.setFileName(fds.getName());

        // Add the attachment to the message.
        msg.addBodyPart(att);

        // Try to send the email.
        try {
```

```
            System.out.println("Attempting to send an email through Amazon SES "
                               +"using the AWS SDK for Java...");

            // Instantiate an Amazon SES client, which will make the service
            // call with the supplied AWS credentials.
            AmazonSimpleEmailService client =
                    AmazonSimpleEmailServiceClientBuilder.standard()
                    // Replace US_WEST_2 with the AWS Region you're using for
                    // Amazon SES.
                    .withRegion(Regions.US_WEST_2).build();

            // Print the raw email content on the console
            PrintStream out = System.out;
            message.writeTo(out);

            // Send the email.
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            message.writeTo(outputStream);
            RawMessage rawMessage =
              new RawMessage(ByteBuffer.wrap(outputStream.toByteArray()));

            SendRawEmailRequest rawEmailRequest =
              new SendRawEmailRequest(rawMessage)
                  .withConfigurationSetName(CONFIGURATION_SET);

            client.sendRawEmail(rawEmailRequest);
            System.out.println("Email sent!");
        // Display an error if something goes wrong.
        } catch (Exception ex) {
          System.out.println("Email Failed");
            System.err.println("Error message: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Python

The following code example shows how to use the Python email.mime packages and the AWS SDK for Python (Boto) to compose and send a raw email.

```
import json
import boto3
```

```
from botocore.exceptions import ClientError
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
import os

def boto3_rawemailv2():
    SENDER = "Sender <sender@example.com>"
    RECIPIENT = "recipient@example.com"
    CONFIGURATION_SET = "ConfigSet"
    AWS_REGION = "us-east-1"
    SUBJECT = "Customer service contact info"
    ATTACHMENT = "path/to/customers-to-contact.xlsx"
    BODY_TEXT = "Hello,\r\nPlease see the attached file for a list of customers to
 contact."

    # The HTML body of the email.
    BODY_HTML = """\
    <html>
    <head/>
    <body>
    <h1>Hello!</h1>
    <p>Please see the attached file for a list of customers to contact.</p>
    </body>
    </html>
    """

    # The character encoding for the email.
    CHARSET = "utf-8"
    msg = MIMEMultipart('mixed')
    # Add subject, from and to lines.
    msg['Subject'] = SUBJECT
    msg['From'] = SENDER
    msg['To'] = RECIPIENT

    # Create a multipart/alternative child container.
    msg_body = MIMEMultipart('alternative')

    # Encode the text and HTML content and set the character encoding. This step is
    # necessary if you're sending a message with characters outside the ASCII range.
    textpart = MIMEText(BODY_TEXT.encode(CHARSET), 'plain', CHARSET)
    htmlpart = MIMEText(BODY_HTML.encode(CHARSET), 'html', CHARSET)

    # Add the text and HTML parts to the child container.
```

```
    msg_body.attach(textpart)
    msg_body.attach(htmlpart)

    # Define the attachment part and encode it using MIMEApplication.
    att = MIMEApplication(open(ATTACHMENT, 'rb').read())

    # Add a header to tell the email client to treat this part as an attachment,
    # and to give the attachment a name.
    att.add_header('Content-
Disposition','attachment',filename=os.path.basename(ATTACHMENT))

    # Attach the multipart/alternative child container to the multipart/mixed
    # parent container.
    msg.attach(msg_body)
    msg.attach(att)

    #changes start from here
    strmsg = str(msg)
    body = bytes (strmsg, 'utf-8')




    client = boto3.client('sesv2')
    response = client.send_email(
    FromEmailAddress=SENDER,
    Destination={
        'ToAddresses': [RECIPIENT]
    },
    Content={
        'Raw': {
            'Data': body
        }
    }
    )
    print(response)
boto3_rawemailv2 ()
```

Using templates to send personalized email with the Amazon SES API

In Amazon SES you can send templated email either by using a *stored template* or by using an *inline template*.

- **Stored template** – Refers to the `Template` resource that is created and saved in SES by using the `CreateEmailTemplate` operation in the Amazon SES v2 API. The template contains the subject and body of the email containing variables (placeholders) inline with the written content. The name of the stored template and the dynamic data to the placeholder variables in the template are provided when calling either the `SendEmail` or `SendBulkEmail` v2 API operations.

  Stored templates can be easily reused and can save you time and effort when sending similar types of emails. Instead of creating each email from scratch, you only need to create the base structure and design once, then simply update the dynamic content within the template.
- **Inline template** – The `Template` resource is not used, but rather, the subject and body of the email containing variables (placeholders) inline with the written content along with the values for those placeholder variables are provided when calling either the `SendEmail` or `SendBulkEmail` v2 API operations.

  Inline templates streamline the process for sending bulk email by eliminating the need to manage template resources in your SES account and simplify the integration process by allowing you to include template content directly within your application logic. They do not count against the 20,000-template limit per AWS Region.

The following limits apply when using *stored templates*:

- You can create up to 20,000 email templates in each AWS Region.
- Each template can be up to 500 KB in size, including both the text and HTML parts.

The following limit applies when using *inline templates*:

- Each input JSON file can be up to 1 MB in size, including both the text and HTML parts.

The following applies to both *stored* and *inline templates*:

- There are no limits to the number of replacement variables that can be used.
- You can send email to up to 50 destination objects in each call to the `SendBulkEmail` operation. The `Destination` object can contain multiple recipients defined in **ToAddresses**, **CcAddresses**, and **BccAddresses**. The number of destinations you can contact in a single call to

the v2 API may be limited by your account's maximum sending rate. For more information, see Managing your Amazon SES sending limits.

This chapter includes procedures with examples for using both *stored templates* and *inline templates*.

> **Note**
>
> The procedures in this section assume that you've already installed and configured the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

(Optional) Part 1: Set up Rendering Failure event notifications

If you send an email that contains invalid personalization content, Amazon SES might accept the message, but won't be able to deliver it. For this reason, if you plan to send personalized email, you should configure SES to send Rendering Failure event notifications through Amazon SNS. When you receive a Rendering Failure event notification, you can identify which message contained the invalid content, fix the issues, and send the message again.

The procedure in this section is optional, but highly recommended.

To configure Rendering Failure event notifications

1. Create an Amazon SNS topic. For procedures, see Create a Topic in the *Amazon Simple Notification Service Developer Guide*.
2. Subscribe to the Amazon SNS topic. For example, if you want to receive Rendering Failure notifications by email, subscribe an email endpoint (that is, your email address) to the topic.

   For procedures, see Subscribe to a Topic in the *Amazon Simple Notification Service Developer Guide*.
3. Complete the procedures in the section called “Set up an Amazon SNS destination” to set up your configuration sets to publish Rendering Failure events to your Amazon SNS topic.

(Optional) Part 2: Create an email template

If you intend on using a *stored template*, this section will show you how to use the `CreateEmailTemplate` SES v2 API operation to create the template. You can skip this step if you want to use an *inline template*.

This procedure assumes that you've already installed and configured the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To create the template

1. In a text editor, create a new file and paste the following code customizing it as you need.

```
{
    "TemplateName": "MyTemplate",
    "TemplateContent": {
        "Subject": "Greetings, {{name}}!",
        "Text": "Dear {{name}},\r\nYour favorite animal is {{favoriteanimal}}.",
        "Html": "<h1>Hello {{name}},</h1><p>Your favorite animal is
 {{favoriteanimal}}.</p>"
    }
}
```

   This code contains the following properties:

   - **TemplateName** – The name of the `Template` resource. When you send the email, you refer to this name.
   - **TemplateContent** – A container for the following attributes:
     - **Subject** – The subject line of the email. This property may contain replacement tags. These tags use the following format: `{{tagname}}`. When you send the email, you can specify a value for `tagname` for each destination.
     - **Html** – The HTML body of the email. This property may contain replacement tags. The preceding example includes two tags: `{{name}}` and `{{favoriteanimal}}`.
     - **Text** – The text body of the email. Recipients whose email clients don't display HTML content will see this version of the email. This property may also contain replacement tags.
2. Customize the preceding example to fit your needs, and then save the file as *`mytemplate.json`*.

3. At the command line, type the following command to create a new template using the `CreateEmailTemplate` v2 API operation:

```
aws sesv2 create-email-template --cli-input-json file://mytemplate.json
```

Part 3: Sending the personalized email

You can use the following two SES v2 API operations to send emails using either *stored templates* or *inline templates*:

- The `SendEmail` operation is useful for sending a customized email to a single destination object. The v2 API `Destination` object can contain the *ToAddresses*, *CcAddresses*, and *BccAddresses* properties. These can be used in any combination and can contain one or more email addresses that will receive the same email.
- The `SendBulkEmail` operation is useful for sending unique emails to multiple destination objects in a single call to the v2 API.

This section provides examples of how to use the AWS CLI to send templated email using both of these send operations.

Sending templated email to a single destination object

You can use the `SendEmail` operation to send an email to one or more recipients defined in a single destination object. All of the recipients in the `Destination` object will receive the same email.

To send a templated email to a single destination object

1. Depending on whether you want to use a *stored template* or *inline template*, select the respective code example to paste into a text editor, customizing it as you need.

   Stored template code example

   Notice that the template you created in the previous step, *MyTemplate*, is being referenced as the value for the `TemplateName` parameter.

```
{
    "FromEmailAddress": "Mary Major <mary.major@example.com>",
    "Destination": {
```

```
        "ToAddresses": [
            "alejandro.rosalez@example.com", "jimmy.jet@example.com"
        ]
    },
    "Content": {
        "Template": {
            "TemplateName": "MyTemplate",
            "TemplateData": "{ \"name\":\"Alejandro\", \"favoriteanimal\":
 \"alligator\" }"
        }
    },
    "ConfigurationSetName": "ConfigSet"
}
```

This code contains the following properties:

- **FromEmailAddress** – The email address of the sender.
- **Destination** – An object containing the email recipients defined in the *ToAddresses*, *CcAddresses*, and *BccAddresses* properties. These can be used in any combination and can contain one or more email addresses that will receive the same email.
- **TemplateName** – The name of the `Template` resource to apply to the email.
- **TemplateData** – An escaped JSON string that contains key-value pairs. The keys correspond to the variables defined in the `TemplateContent` properties in the stored template, for example, `{{name}}`. The values represent the content that replaces the variables.
- **ConfigurationSetName** – The name of the configuration set to use when sending the email.

> **Note**
>
> We recommend that you use a configuration set that is configured to publish Rendering Failure events to Amazon SNS. For more information, see the section called “(Optional) Part 1: Set up notifications”.

Inline template code example

Notice that the `TemplateContent` properties (that would normally be defined in a *stored template*), are being defined *inline* along with the `TemplateData` property which makes this an *inline template*.

```
{
    "FromEmailAddress": "Mary Major <mary.major@example.com>",
    "Destination": {
        "ToAddresses": [
            "alejandro.rosalez@example.com", "jimmy.jet@example.com"
        ]
    },
    "Content": {
        "Template": {
            "TemplateContent": {
                "Subject": "Greetings, {{name}}!",
                "Text": "Dear {{name}},\r\nYour favorite animal is
 {{favoriteanimal}}.",
                "Html": "<h1>Hello {{name}},</h1><p>Your favorite animal is
 {{favoriteanimal}}.</p>"
            },
            "TemplateData": "{ \"name\":\"Alejandro\", \"favoriteanimal\":
 \"alligator\" }"
        }
    },
    "ConfigurationSetName": "ConfigSet"
}
```

This code contains the following properties:

- **FromEmailAddress** – The email address of the sender.
- **Destination** – An object containing the email recipients defined in the *ToAddresses*, *CcAddresses*, and *BccAddresses* properties. These can be used in any combination and can contain one or more email addresses that will receive the same email.
- **TemplateContent** – A container for the following attributes:
  - **Subject** – The subject line of the email. This property may contain replacement tags. These tags use the following format: `{{tagname}}`. When you send the email, you can specify a value for `tagname` for each destination.

- **Html** – The HTML body of the email. This property may contain replacement tags. The preceding example includes two tags: `{{name}}` and `{{favoriteanimal}}`.
- **Text** – The text body of the email. Recipients whose email clients don't display HTML content will see this version of the email. This property may also contain replacement tags.

- **TemplateData** – An escaped JSON string that contains key-value pairs. The keys correspond to the variables defined in the `TemplateContent` properties in this file, for example, `{{name}}`. The values represent the content that replaces the variables.
- **ConfigurationSetName** – The name of the configuration set to use when sending the email.

> **Note**
>
> We recommend that you use a configuration set that is configured to publish Rendering Failure events to Amazon SNS. For more information, see the section called "(Optional) Part 1: Set up notifications".

2. Customize the preceding example to fit your needs, and then save the file as *`myemail.json`*.
3. At the command line, type the following v2 API command to send the email:

```
aws sesv2 send-email --cli-input-json file://myemail.json
```

Sending templated email to multiple destination objects

You can use the `SendBulkEmail` operation to send an email to multiple destination objects in a single call to the SES v2 API. SES sends a unique email to the recipient or recipients in each `Destination` object.

To send a templated email to multiple destination objects

1. Depending on whether you want to use a *stored template* or *inline template*, select the respective code example to paste into a text editor, customizing it as you need.

   Stored template code example

   Notice that the template you created in the previous step, *MyTemplate*, is being referenced as the value for the `TemplateName` parameter.

```
{
    "FromEmailAddress": "Mary Major <mary.major@example.com>",
    "DefaultContent": {
        "Template": {
            "TemplateName": "MyTemplate",
            "TemplateData": "{ \"name\":\"friend\", \"favoriteanimal\":\"unknown
\" }"
        }
    },
    "BulkEmailEntries": [
        {
            "Destination": {
                "ToAddresses": [
                    "anaya.iyengar@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Anaya\",
 \"favoriteanimal\":\"angelfish\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "liu.jie@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Liu\",
 \"favoriteanimal\":\"lion\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "shirley.rodriguez@example.com"
                ]
            },
            "ReplacementEmailContent": {
```

```
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Shirley\",
\"favoriteanimal\":\"shark\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "richard.roe@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{}"
                }
            }
        }
    ],
    "ConfigurationSetName": "ConfigSet"
}
```

This code contains the following properties:

- **FromEmailAddress** – The email address of the sender.
- **DefaultContent** – A JSON object that contains the `TemplateName` and `TemplateData` objects.
- **TemplateName** – The name of the `Template` resource to apply to the email.
- **TemplateData** – Contains key-value pairs that will be used if the `ReplacementEmailContent` object contains an empty JSON object, `{}`, in the `ReplacementTemplateData` property.
- **BulkEmailEntries** – An array that contains one or more `Destination` objects.
- **Destination** – An object containing the email recipients defined in the *ToAddresses*, *CcAddresses*, and *BccAddresses* properties. These can be used in any combination and can contain one or more email addresses that will receive the same email.
- **ReplacementTemplateData** – An escaped JSON string that contains key-value pairs. The keys correspond to the variables in the template, for example, `{{name}}`. The values represent the content that replaces the variables in the email. (If the JSON string here

is empty, indicated by {}, the key-value pairs defined in the `TemplateData` property within the `DefaultContent` object will be used.)

- **ConfigurationSetName** – The name of the configuration set to use when sending the email.

> **Note**
>
> We recommend that you use a configuration set that is configured to publish Rendering Failure events to Amazon SNS. For more information, see the section called "(Optional) Part 1: Set up notifications".

Inline template code example

Notice that the `TemplateContent` properties (that would normally be defined in a *stored template*), are being defined *inline* along with the `TemplateData` property which makes this an *inline template*.

```
{
    "FromEmailAddress": "Mary Major <mary.major@example.com>",
    "DefaultContent": {
        "Template": {
            "TemplateContent": {
                "Subject": "Greetings, {{name}}!",
                "Text": "Dear {{name}},\r\nYour favorite animal is
 {{favoriteanimal}}.",
                "Html": "<h1>Hello {{name}},</h1><p>Your favorite animal is
 {{favoriteanimal}}.</p>"
            },
            "TemplateData": "{ \"name\":\"friend\", \"favoriteanimal\":\"unknown
\" }"
        }
    },
    "BulkEmailEntries": [
        {
            "Destination": {
                "ToAddresses": [
                    "anaya.iyengar@example.com"
                ]
            },
            "ReplacementEmailContent": {
```

```
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Anaya\",
\"favoriteanimal\":\"angelfish\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "liu.jie@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Liu\",
\"favoriteanimal\":\"lion\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "shirley.rodriguez@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{ \"name\":\"Shirley\",
\"favoriteanimal\":\"shark\" }"
                }
            }
        },
        {
            "Destination": {
                "ToAddresses": [
                    "richard.roe@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{}"
                }
            }
        }
```

```
    ],
    "ConfigurationSetName": "ConfigSet"
}
```

This code contains the following properties:

- **FromEmailAddress** – The email address of the sender.
- **DefaultContent** – A JSON object that contains the `TemplateContent` and `TemplateData` objects.
- **TemplateContent** – A container for the following attributes:
  - **Subject** – The subject line of the email. This property may contain replacement tags. These tags use the following format: `{{tagname}}`. When you send the email, you can specify a value for `tagname` for each destination.
  - **Html** – The HTML body of the email. This property may contain replacement tags. The preceding example includes two tags: `{{name}}` and `{{favoriteanimal}}`.
  - **Text** – The text body of the email. Recipients whose email clients don't display HTML content will see this version of the email. This property may also contain replacement tags.
- **TemplateData** – Contains key-value pairs that will be used if the `ReplacementEmailContent` object contains an empty JSON object, `{}`, in the `ReplacementTemplateData` property.
- **BulkEmailEntries** – An array that contains one or more `Destination` objects.
- **Destination** – An object containing the email recipients defined in the *ToAddresses*, *CcAddresses*, and *BccAddresses* properties. These can be used in any combination and can contain one or more email addresses that will receive the same email.
- **ReplacementTemplateData** – An escaped JSON string that contains key-value pairs. The keys correspond to the variables defined in the `TemplateContent` properties in this file, for example, `{{name}}`. The values represent the content that replaces the variables in the email. (If the JSON string here is empty, indicated by `{}`, the key-value pairs defined in the `TemplateData` property within the `DefaultContent` object will be used.)
- **ConfigurationSetName** – The name of the configuration set to use when sending the email.

> **Note**
>
> We recommend that you use a configuration set that is configured to publish Rendering Failure events to Amazon SNS. For more information, see the section called "(Optional) Part 1: Set up notifications".

2. Change the values in the code in the previous step to meet your needs, and then save the file as *mybulkemail.json*.
3. At the command line, type the following v2 API command to send the bulk email:

```
aws sesv2 send-bulk-email --cli-input-json file://mybulkemail.json
```

Advanced email personalization

If you're using a *stored template*, that is, you've created a `Template` resource in Amazon SES by using the `CreateEmailTemplate` operation with the SES v2 API, you can take advantage of the Handlebars system to create templates that include advanced features, such as nested attributes, array iteration, basic conditional statements, and the creation of inline partials. This section provides examples of these features.

Handlebars includes additional features beyond those documented in this section. For more information, see Built-In Helpers at handlebarsjs.com.

> **Note**
>
> SES doesn't escape HTML content when rendering the HTML template for a message. This means if you're including user inputted data, such as from a contact form, you will need to escape it on the client side.

Topics

- Parsing nested attributes
- Iterating through lists
- Using basic conditional statements
- Creating inline partials

Parsing nested attributes

Handlebars includes support for nested paths, which makes it easy to organize complex customer data, and then refer to that data in your email templates.

For example, you can organize recipient data into several general categories. Within each of those categories, you can include detailed information. The following code example shows an example of this structure for a single recipient:

```
{
  "meta":{
    "userId":"51806220607"
  },
  "contact":{
    "firstName":"Anaya",
    "lastName":"Iyengar",
    "city":"Bengaluru",
    "country":"India",
    "postalCode":"560052"
  },
  "subscription":[
    {
      "interest":"Sports"
    },
    {
      "interest":"Travel"
    },
    {
      "interest":"Cooking"
    }
  ]
}
```

In your email templates, you can refer to nested attributes by providing the name of the parent attribute, followed by a period (.), followed by the name of the attribute for which you want to include the value. For example, if you use the data structure shown in the preceding example, and you want to include each recipient's first name in the email template, include the following text in your email template: `Hello {{contact.firstName}}!`

Handlebars can parse paths that are nested several levels deep, which means you have flexibility in how you structure your template data.

Iterating through lists

The `each` helper function iterates through items in an array. The following code is an example of an email template that uses the `each` helper function to create an itemized list of each recipient's interests.

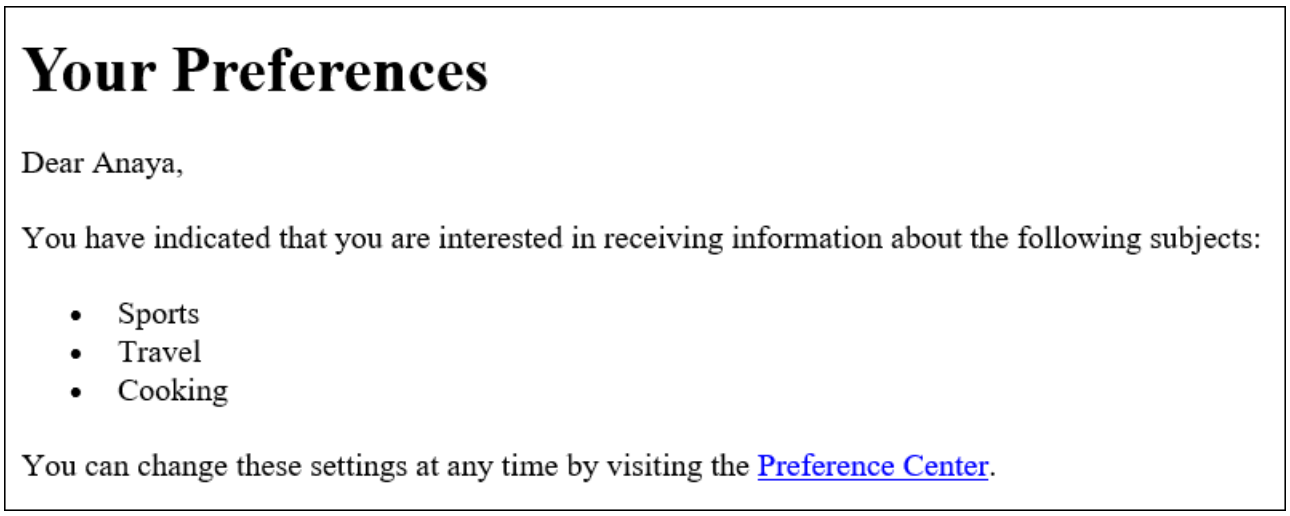
```
{
  "Template": {
    "TemplateName": "Preferences",
    "SubjectPart": "Subscription Preferences for {{contact.firstName}}
 {{contact.lastName}}",
    "HtmlPart": "<h1>Your Preferences</h1>
                <p>You have indicated that you are interested in receiving
                  information about the following subjects:</p>
                <ul>
                  {{#each subscription}}
                    <li>{{interest}}</li>
                  {{/each}}
                </ul>
                <p>You can change these settings at any time by visiting
                   the <a href=https://www.example.com/prefererences/i.aspx?
id={{meta.userId}}>
                   Preference Center</a>.</p>",
    "TextPart": "Your Preferences\n\nYou have indicated that you are interested in
                receiving information about the following subjects:\n
                {{#each subscription}}
                  - {{interest}}\n
                {{/each}}
                \nYou can change these settings at any time by
                visiting the Preference Center at
                https://www.example.com/prefererences/i.aspx?id={{meta.userId}}"
  }
}
```

> **Important**
>
> In the preceding code example, the values of the `HtmlPart` and `TextPart` attributes include line breaks to make the example easier to read. The JSON file for your template can't contain line breaks within these values. If you copied and pasted this example into your own JSON file, remove the line breaks and extra spaces from the `HtmlPart` and `TextPart` sections before proceeding.

After you create the template, you can use the `SendEmail` or the `SendBulkEmail` operation to send email to recipients using this template. As long as each recipient has at least one value in the `Interests` object, they receive an email that includes an itemized list of their interests. The following example shows a JSON file that can be used to send email to multiple recipients using the preceding template:

```
{
  "Source":"Sender Name <sender@example.com>",
  "Template":"Preferences",
  "Destinations":[
    {
      "Destination":{
        "ToAddresses":[
          "anaya.iyengar@example.com"
        ]
      },
      "ReplacementTemplateData":"{\"meta\":{\"userId\":\"51806220607\"},\"contact\":
{\"firstName\":\"Anaya\",\"lastName\":\"Iyengar\"},\"subscription\":[{\"interest\":
\"Sports\"},{\"interest\":\"Travel\"},{\"interest\":\"Cooking\"}]}"
      },
    {
      "Destination":{
        "ToAddresses":[
          "shirley.rodriguez@example.com"
        ]
      },
      "ReplacementTemplateData":"{\"meta\":{\"userId\":\"1981624758263\"},\"contact\":
{\"firstName\":\"Shirley\",\"lastName\":\"Rodriguez\"},\"subscription\":[{\"interest\":
\"Technology\"},{\"interest\":\"Politics\"}]}"
    }
  ],
  "DefaultTemplateData":"{\"meta\":{\"userId\":\"\"},\"contact\":{\"firstName\":
\"Friend\",\"lastName\":\"\"},\"subscription\":[]}"
}
```

When you send an email to the recipients listed in the preceding example using the `SendBulkEmail` operation, they receive a message that resembles the example shown in the following image:



Using basic conditional statements

This section builds on the example described in the previous section. The example in the previous section uses the `each` helper to iterate through a list of interests. However, recipients for whom no interests are specified receive an email that contains an empty list. By using the `{{if}}` helper, you can format the email differently if a certain attribute is present in the template data. The following code uses the `{{if}}` helper to display the bulleted list from the preceding section if the `Subscription` array contains any values. If the array is empty, a different block of text is displayed.

```
{
  "Template": {
    "TemplateName": "Preferences2",
    "SubjectPart": "Subscription Preferences for {{contact.firstName}}
 {{contact.lastName}}",
    "HtmlPart": "<h1>Your Preferences</h1>
                <p>Dear {{contact.firstName}},</p>
                {{#if subscription}}
                  <p>You have indicated that you are interested in receiving
                    information about the following subjects:</p>
                    <ul>
                    {{#each subscription}}
                      <li>{{interest}}</li>
                    {{/each}}
                    </ul>
                    <p>You can change these settings at any time by visiting
                      the <a href=https://www.example.com/prefererences/i.aspx?
id={{meta.userId}}>
                      Preference Center</a>.</p>
```

```
                {{else}}
                  <p>Please update your subscription preferences by visiting
                    the <a href=https://www.example.com/prefererences/i.aspx?
id={{meta.userId}}>
                    Preference Center</a>.
                {{/if}}",
    "TextPart": "Your Preferences\n\nDear {{contact.firstName}},\n\n
                {{#if subscription}}
                  You have indicated that you are interested in receiving
                  information about the following subjects:\n
                  {{#each subscription}}
                    - {{interest}}\n
                  {{/each}}
                  \nYou can change these settings at any time by visiting the
                  Preference Center at https://www.example.com/prefererences/i.aspx?
id={{meta.userId}}.
                {{else}}
                  Please update your subscription preferences by visiting the
                  Preference Center at https://www.example.com/prefererences/i.aspx?
id={{meta.userId}}.
                {{/if}}"
  }
}
```

Important

In the preceding code example, the values of the `HtmlPart` and `TextPart` attributes include line breaks to make the example easier to read. The JSON file for your template can't contain line breaks within these values. If you copied and pasted this example into your own JSON file, remove the line breaks and extra spaces from the `HtmlPart` and `TextPart` sections before proceeding.

The following example shows a JSON file that can be used to send email to multiple recipients using the preceding template:

```
{
  "Source":"Sender Name <sender@example.com>",
  "Template":"Preferences2",
  "Destinations":[
    {
      "Destination":{
```

```
        "ToAddresses":[
          "anaya.iyengar@example.com"
        ]
      },
      "ReplacementTemplateData":"{\"meta\":{\"userId\":\"51806220607\"},\"contact\":
{\"firstName\":\"Anaya\",\"lastName\":\"Iyengar\"},\"subscription\":[{\"interest\":
\"Sports\"},{\"interest\":\"Cooking\"}]}"
      },
    {
      "Destination":{
        "ToAddresses":[
          "shirley.rodriguez@example.com"
        ]
      },
      "ReplacementTemplateData":"{\"meta\":{\"userId\":\"1981624758263\"},\"contact\":
{\"firstName\":\"Shirley\",\"lastName\":\"Rodriguez\"}}"
    }
  ],
  "DefaultTemplateData":"{\"meta\":{\"userId\":\"\"},\"contact\":{\"firstName\":
\"Friend\",\"lastName\":\"\"},\"subscription\":[]}"
}
```

In this example, the recipient whose template data included a list of interests receives the same email as the example shown in the previous section. The recipient whose template data did not include any interests, however, receives an email that resembles the example shown in the following image:

Creating inline partials

You can use inline partials to simplify templates that include repeated strings. For example, you could create an inline partial that includes the recipient's first name, and, if it's available, their last name by adding the following code to the beginning of your template:

```
{{#* inline \"fullName\"}}{{firstName}}{{#if lastName}} {{lastName}}{{/if}}{{/
inline}}\n
```

> **Note**
>
> The newline character (\n) is required to separate the `{{inline}}` block from the content in your template. The newline isn't rendered in the final output.

After you create the `fullName` partial, you can include it anywhere in your template by preceding the name of the partial with a greater-than (>) sign followed by a space, as in the following example: `{{> fullName}}`. Inline partials are not transferred between parts of the email. For example, if you want to use the same inline partial in both the HTML and the text version of the email, you must define it in both the `HtmlPart` and the `TextPart` sections.

You can also use inline partials when iterating through arrays. You can use the following code to create a template that uses the `fullName` inline partial. In this example, the inline partial applies to both the recipient's name and to an array of other names:

```
{
  "Template": {
    "TemplateName": "Preferences3",
    "SubjectPart": "{{firstName}}'s Subscription Preferences",
    "HtmlPart": "{{#* inline \"fullName\"}}
                  {{firstName}}{{#if lastName}} {{lastName}}{{/if}}
                {{/inline~}}\n
                <h1>Hello {{> fullName}}!</h1>
                <p>You have listed the following people as your friends:</p>
                <ul>
                {{#each friends}}
                  <li>{{> fullName}}</li>
                {{/each}}</ul>",
    "TextPart": "{{#* inline \"fullName\"}}
                  {{firstName}}{{#if lastName}} {{lastName}}{{/if}}
                {{/inline~}}\n
                Hello {{> fullName}}! You have listed the following people
                as your friends:\n
                {{#each friends}}
                  - {{> fullName}}\n
                {{/each}}"
  }
}
```

Important

In the preceding code example, the values of the `HtmlPart` and `TextPart` attributes include line breaks to make the example easier to read. The JSON file for your template can't contain line breaks within these values. If you copied and pasted this example into your own JSON file, remove the line breaks and extra spaces from these sections.

Managing email templates

In addition to creating email templates, you can also use the Amazon SES v2 API to update or delete existing templates, to list all of your existing templates, or to view the contents of a template.

This section contains procedures for using the AWS CLI to perform tasks related to SES templates.

Note

The procedures in this section assume that you've already installed and configured the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

Viewing a list of email templates

You can use the `ListEmailTemplate` SES v2 API operation to view a list of all of your existing email templates.

To view a list of email templates

- At the command line, enter the following command:

```
aws sesv2 list-email-templates
```

If there are existing email templates in your SES account in the current Region, this command returns a response that resembles the following example:

```
{
    "TemplatesMetadata": [
```

```
        {
            "Name": "SpecialOffers",
            "CreatedTimestamp": "2020-08-05T16:04:12.640Z"
        },
        {
            "Name": "NewsAndUpdates",
            "CreatedTimestamp": "2019-10-03T20:03:34.574Z"
        }
    ]
}
```

If you haven't created any templates, the command returns a `TemplatesMetadata` object with no members.

Viewing the contents of a specific email template

You can use the `GetEmailTemplate` SES v2 API operation to view the contents of a specific email template.

To view the contents of an email template

- At the command line, enter the following command:

```
aws sesv2 get-email-template --template-name MyTemplate
```

In the preceding command, replace *MyTemplate* with the name of the template that you want to view.

If the template name that you provided matches a template that exists in your SES account, this command returns a response that resembles the following example:

```
{
    "Template": {
        "TemplateName": "TestMessage",
        "SubjectPart": "Amazon SES Test Message",
        "TextPart": "Hello! This is the text part of the message.",
        "HtmlPart": "<html>\n<body>\n<h2>Hello!</h2>\n<p>This is the HTML part of
 the message.</p></body>\n</html>"
    }
}
```

If the template name that you provided doesn't match a template that exists in your SES account, the command returns a `NotFoundException` error.

Deleting an email template

You can use the `DeleteEmailTemplate` SES v2 API operation to delete a specific email template.

To delete an email template

- At the command line, enter the following command:

```
aws sesv2 delete-email-template --template-name MyTemplate
```

  In the preceding command, replace *MyTemplate* with the name of the template that you want to delete.

  This command doesn't provide any output. You can verify that the template was deleted by using the GetTemplate operation.

Updating an email template

You can use the `UpdateEmailTemplate` SES v2 API operation to update an existing email template. For example, this operation is helpful if you want to change the subject line of the email template, or if you need to modify the body of the message itself.

To update an email template

1. Use the `GetEmailTemplate` command to retrieve the existing template by entering the following command on the command line:

```
aws sesv2 get-email-template --template-name MyTemplate
```

   In the preceding command, replace *MyTemplate* with the name of the template that you want to update.

   If the template name that you provided matches a template that exists in your SES account, this command returns a response that resembles the following example:

```
{
    "Template": {
        "TemplateName": "TestMessage",
        "SubjectPart": "Amazon SES Test Message",
        "TextPart": "Hello! This is the text part of the message.",
        "HtmlPart": "<html>\n<body>\n<h2>Hello!</h2>\n<p>This is the HTML part of
 the message.</p></body>\n</html>"
    }
}
```

2. In a text editor, create a new file. Paste the output of the previous command into the file.
3. Modify the template as needed. Any lines that you omit are removed from the template. For example, if you only want to change the `SubjectPart` of the template, you still need to include the `TextPart` and `HtmlPart` properties.

   When you finish, save the file as `update_template.json`.
4. At the command line, enter the following command:

   ```
   aws sesv2 update-email-template --cli-input-json file://path/to/
   update_template.json
   ```

   In the preceding command, replace *`path/to/update_template.json`* with the path to the `update_template.json` file that you created in the previous step.

   If the template is updated successfully, this command doesn't provide any output. You can verify that the template was updated by using the `GetEmailTemplate` operation.

   If the template that you specified doesn't exist, this command returns a `TemplateDoesNotExist` error. If the template doesn't contain either the `TextPart` or `HtmlPart` property (or both), this command returns an `InvalidParameterValue` error.

Sending email through Amazon SES using an AWS SDK

You can use an AWS SDK to send email through Amazon SES. AWS SDKs are available for several programming languages. For more information, see Tools for Amazon Web Services.

Prerequisites

The following prerequisites must be completed in order to complete any of the code samples in the next section:

- If you haven't already done so, complete the tasks in Setting up Amazon Simple Email Service.
- **Verify your email address with Amazon SES**—Before you can send an email with Amazon SES, you must verify that you own the sender's email address. If your account is still in the Amazon SES sandbox, you must also verify the recipient email address. We recommend you use the Amazon SES console to verify email addresses. For more information, see Creating an email address identity.
- **Get your AWS credentials**—You need an AWS access key ID and AWS secret access key to access Amazon SES using an SDK. You can find your credentials by using the Security Credentials page in the AWS Management Console. For more information about credentials, see Types of Amazon SES credentials.
- **Create a shared credentials file**—For the sample code in this section to function properly, you must create a shared credentials file. For more information, see Creating a shared credentials file to use when sending email through Amazon SES using an AWS SDK.

Code examples

> **Important**
>
> In the following tutorials, you send an email to yourself so that you can check to see if you received it. For further experimentation or load testing, use the Amazon SES mailbox simulator. Emails that you send to the mailbox simulator do not count toward your sending quota or your bounce and complaint rates. For more information, see Using the mailbox simulator manually.

.NET

The following procedure shows you how to send an email through Amazon SES using Visual Studio and the AWS SDK for .NET.

This solution was tested using the following components:

- Microsoft Visual Studio Community 2017, version 15.4.0.

- Microsoft .NET Framework version 4.6.1.
- The AWSSDK.Core package (version 3.3.19), installed using NuGet.
- The AWSSDK.SimpleEmail package (version 3.3.6.1), installed using NuGet.

Before you begin, perform the following tasks:

- **Install Visual Studio**—Visual Studio is available at https://www.visualstudio.com/.

To send an email using the AWS SDK for .NET

1. Create a new project by performing the following steps:
   a. Start Visual Studio.
   b. On the **File** menu, choose **New**, **Project**.
   c. On the **New Project** window, in the panel on the left, expand **Installed**, and then expand **Visual C#**.
   d. In the panel on the right, choose **Console App (.NET Framework)**.
   e. For **Name**, type **`AmazonSESSample`**, and then choose **OK**.
2. Use NuGet to include the Amazon SES packages in your solution by completing the following steps:
   a. In the **Solution Explorer** pane, right-click your project, and then choose **Manage NuGet Packages**.
   b. On the **NuGet: AmazonSESSample** tab, choose **Browse**.
   c. In the search box, type **`AWSSDK.SimpleEmail`**.
   d. Choose the **AWSSDK.SimpleEmail** package, and then choose **Install**.
   e. On the **Preview Changes** window, choose **OK**.
3. On the **Program.cs** tab, paste the following code:

```
using Amazon;
using System;
using System.Collections.Generic;
using Amazon.SimpleEmail;
using Amazon.SimpleEmail.Model;

namespace AmazonSESSample
```

```
{
    class Program
    {
        // Replace sender@example.com with your "From" address.
        // This address must be verified with Amazon SES.
        static readonly string senderAddress = "sender@example.com";

        // Replace recipient@example.com with a "To" address. If your account
        // is still in the sandbox, this address must be verified.
        static readonly string receiverAddress = "recipient@example.com";

        // The configuration set to use for this email. If you do not want to
 use a
        // configuration set, comment out the following property and the
        // ConfigurationSetName = configSet argument below.
        static readonly string configSet = "ConfigSet";

        // The subject line for the email.
        static readonly string subject = "Amazon SES test (AWS SDK for .NET)";

        // The email body for recipients with non-HTML email clients.
        static readonly string textBody = "Amazon SES Test (.NET)\r\n"
                                        + "This email was sent through Amazon
 SES "
                                        + "using the AWS SDK for .NET.";

        // The HTML body of the email.
        static readonly string htmlBody = @"<html>
<head></head>
<body>
  <h1>Amazon SES Test (SDK for .NET)</h1>
  <p>This email was sent with
    <a href='ses/'>Amazon SES</a> using the
    <a href='sdk-for-net/'>
      AWS SDK for .NET</a>.</p>
</body>
</html>";

        static void Main(string[] args)
        {
            // Replace USWest2 with the AWS Region you're using for Amazon SES.
            // Acceptable values are EUWest1, USEast1, and USWest2.
            using (var client = new
 AmazonSimpleEmailServiceClient(RegionEndpoint.USWest2))
```

```
            {
                var sendRequest = new SendEmailRequest
                {
                    Source = senderAddress,
                    Destination = new Destination
                    {
                        ToAddresses =
                        new List<string> { receiverAddress }
                    },
                    Message = new Message
                    {
                        Subject = new Content(subject),
                        Body = new Body
                        {
                            Html = new Content
                            {
                                Charset = "UTF-8",
                                Data = htmlBody
                            },
                            Text = new Content
                            {
                                Charset = "UTF-8",
                                Data = textBody
                            }
                        }
                    },
                    // If you are not using a configuration set, comment
                    // or remove the following line
                    ConfigurationSetName = configSet
                };
                try
                {
                    Console.WriteLine("Sending email using Amazon SES...");
                    var response = client.SendEmail(sendRequest);
                    Console.WriteLine("The email was sent successfully.");
                }
                catch (Exception ex)
                {
                    Console.WriteLine("The email was not sent.");
                    Console.WriteLine("Error message: " + ex.Message);

                }
            }
```

```
            Console.Write("Press any key to continue...");
            Console.ReadKey();
        }
    }
}
```

4. In the code editor, do the following:

   - Replace *sender@example.com* with the "From:" email address. This address must be verified. For more information, see *Verified identities*.
   - Replace *recipient@example.com* with the "To:" address. If your account is still in the sandbox, this address must also be verified.
   - Replace *ConfigSet* with the name of the configuration set to use when sending this email.
   - Replace *USWest2* with the name of the AWS Region endpoint you use to send email using Amazon SES. For a list of regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

   When you finish, save `Program.cs`.

5. Build and run the application by completing the following steps:

   a. On the **Build** menu, choose **Build Solution**.

   b. On the **Debug** menu, choose **Start Debugging**. A console window appears.

6. Review the output of the console. If the email was successfully sent, the console displays `"The email was sent successfully."`

7. If the email was successfully sent, sign in to the email client of the recipient address. You will see the message that you sent.

Java

The following procedure shows you how to use Eclipse IDE for Java EE Developers and AWS Toolkit for Eclipse to create an AWS SDK project and modify the Java code to send an email through Amazon SES.

Before you begin, perform the following tasks:

- **Install Eclipse**—Eclipse is available at https://www.eclipse.org/downloads. The code in this tutorial was tested using Eclipse Neon.3 (version 4.6.3), running version 1.8 of the Java Runtime Environment.
- **Install the AWS Toolkit for Eclipse**—Instructions for adding the AWS Toolkit for Eclipse to your Eclipse installation are available at https://aws.amazon.com/eclipse. The code in this tutorial was tested using version 2.3.1 of the AWS Toolkit for Eclipse.

To send an email using the AWS SDK for Java

1. Create an AWS Java Project in Eclipse by performing the following steps:
   a. Start Eclipse.
   b. On the **File** menu, choose **New**, and then choose **Other**. On the **New** window, expand the **AWS** folder, and then choose **AWS Java Project**.
   c. In the **New AWS Java Project** dialog box, do the following:
      i. For **Project name**, type a project name.
      ii. Under **AWS SDK for Java Samples**, select **Amazon Simple Email Service JavaMail Sample**.
      iii. Choose **Finish**.
2. In Eclipse, in the **Package Explorer** pane, expand your project.
3. Under your project, expand the `src/main/java` folder, expand the `com.amazon.aws.samples` folder, and then double-click `AmazonSESSample.java`.
4. Replace the entire contents of `AmazonSESSample.java` with the following code:

```
package com.amazonaws.samples;

import java.io.IOException;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleemail.AmazonSimpleEmailService;
import com.amazonaws.services.simpleemail.AmazonSimpleEmailServiceClientBuilder;
import com.amazonaws.services.simpleemail.model.Body;
import com.amazonaws.services.simpleemail.model.Content;
import com.amazonaws.services.simpleemail.model.Destination;
import com.amazonaws.services.simpleemail.model.Message;
```

```
import com.amazonaws.services.simpleemail.model.SendEmailRequest;

public class AmazonSESSample {

  // Replace sender@example.com with your "From" address.
  // This address must be verified with Amazon SES.
  static final String FROM = "sender@example.com";

  // Replace recipient@example.com with a "To" address. If your account
  // is still in the sandbox, this address must be verified.
  static final String TO = "recipient@example.com";

  // The configuration set to use for this email. If you do not want to use a
  // configuration set, comment the following variable and the
  // .withConfigurationSetName(CONFIGSET); argument below.
  static final String CONFIGSET = "ConfigSet";

  // The subject line for the email.
  static final String SUBJECT = "Amazon SES test (AWS SDK for Java)";

  // The HTML body for the email.
  static final String HTMLBODY = "<h1>Amazon SES test (AWS SDK for Java)</h1>"
      + "<p>This email was sent with <a href='ses/'>"
      + "Amazon SES</a> using the <a href='sdk-for-java/'>"
      + "AWS SDK for Java</a>";

  // The email body for recipients with non-HTML email clients.
  static final String TEXTBODY = "This email was sent through Amazon SES "
      + "using the AWS SDK for Java.";

  public static void main(String[] args) throws IOException {

    try {
      AmazonSimpleEmailService client =
          AmazonSimpleEmailServiceClientBuilder.standard()
          // Replace US_WEST_2 with the AWS Region you're using for
          // Amazon SES.
            .withRegion(Regions.US_WEST_2).build();
      SendEmailRequest request = new SendEmailRequest()
          .withDestination(
              new Destination().withToAddresses(TO))
          .withMessage(new Message()
              .withBody(new Body()
                  .withHtml(new Content()
```

```
                        .withCharset("UTF-8").withData(HTMLBODY))
                    .withText(new Content()
                        .withCharset("UTF-8").withData(TEXTBODY)))
                .withSubject(new Content()
                    .withCharset("UTF-8").withData(SUBJECT)))
            .withSource(FROM)
            // Comment or remove the next line if you are not using a
            // configuration set
            .withConfigurationSetName(CONFIGSET);
        client.sendEmail(request);
        System.out.println("Email sent!");
    } catch (Exception ex) {
        System.out.println("The email was not sent. Error message: "
            + ex.getMessage());
    }
  }
}
```

5. In `AmazonSESSample.java`, replace the following with your own values:

> **Important**
>
> The email addresses are case-sensitive. Make sure that the addresses are exactly the same as the ones you verified.

- `SENDER@EXAMPLE.COM`—Replace with your "From" email address. You must verify this address before you run this program. For more information, see Verified identities in Amazon SES.
- `RECIPIENT@EXAMPLE.COM`—Replace with your "To" email address. If your account is still in the sandbox, you must verify this address before you use it. For more information, see Request production access (Moving out of the Amazon SES sandbox).
- **(Optional) `us-west-2`**—If you want to use Amazon SES in a Region other than US West (Oregon), replace this with the Region you want to use. For a list of Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

6. Save `AmazonSESSample.java`.
7. To build the project, choose **Project** and then choose **Build Project**.

> **Note**
>
> If this option is disabled, automatic building may be enabled; if so, skip this step.

8. To start the program and send the email, choose **Run** and then choose **Run** again.
9. Review the output of the console pane in Eclipse. If the email was successfully sent, the console displays "`Email sent!`" Otherwise, it displays an error message.
10. If the email was successfully sent, sign in to the email client of the recipient address. You will see the message that you sent.

PHP

This topic shows how to use the AWS SDK for PHP to send an email through Amazon SES.

Before you begin, perform the following tasks:

- **Install PHP**—PHP is available at http://php.net/downloads.php. This tutorial requires PHP version 5.5 or higher. After you install PHP, add the path to PHP in your environment variables so that you can run PHP from any command prompt. The code in this tutorial was tested using PHP 7.2.7.
- **Install the AWS SDK for PHP version 3**—For download and installation instructions, see the AWS SDK for PHP documentation. The code in this tutorial was tested using version 3.64.13 of the SDK.

To send an email through Amazon SES using the AWS SDK for PHP

1. In a text editor, create a file named `amazon-ses-sample.php`. Paste the following code:

```
<?php

// If necessary, modify the path in the require statement below to refer to the
// location of your Composer autoload.php file.
require 'vendor/autoload.php';

use Aws\Ses\SesClient;
use Aws\Exception\AwsException;

// Create an SesClient. Change the value of the region parameter if you're
```

```
// using an AWS Region other than US West (Oregon). Change the value of the
// profile parameter if you want to use a profile in your credentials file
// other than the default.
$SesClient = new SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region'  => 'us-west-2'
]);

// Replace sender@example.com with your "From" address.
// This address must be verified with Amazon SES.
$sender_email = 'sender@example.com';

// Replace these sample addresses with the addresses of your recipients. If
// your account is still in the sandbox, these addresses must be verified.
$recipient_emails = ['recipient1@example.com','recipient2@example.com'];

// Specify a configuration set. If you do not want to use a configuration
// set, comment the following variable, and the
// 'ConfigurationSetName' => $configuration_set argument below.
$configuration_set = 'ConfigSet';

$subject = 'Amazon SES test (AWS SDK for PHP)';
$plaintext_body = 'This email was sent with Amazon SES using the AWS SDK for
 PHP.' ;
$html_body =  '<h1>AWS Amazon Simple Email Service Test Email</h1>'.
              '<p>This email was sent with <a href="ses/">'.
              'Amazon SES</a> using the <a href="sdk-for-php/">'.
              'AWS SDK for PHP</a>.</p>';
$char_set = 'UTF-8';

try {
    $result = $SesClient->sendEmail([
        'Destination' => [
            'ToAddresses' => $recipient_emails,
        ],
        'ReplyToAddresses' => [$sender_email],
        'Source' => $sender_email,
        'Message' => [
          'Body' => [
              'Html' => [
                  'Charset' => $char_set,
                  'Data' => $html_body,
              ],
```

```
                'Text' => [
                    'Charset' => $char_set,
                    'Data' => $plaintext_body,
                ],
            ],
            'Subject' => [
                'Charset' => $char_set,
                'Data' => $subject,
            ],
        ],
        // If you aren't using a configuration set, comment or delete the
        // following line
        'ConfigurationSetName' => $configuration_set,
    ]);
    $messageId = $result['MessageId'];
    echo("Email sent! Message ID: $messageId"."\n");
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo("The email was not sent. Error message: ".$e-
>getAwsErrorMessage()."\n");
    echo "\n";
}
```

2. In `amazon-ses-sample.php`, replace the following with your own values:

   - **`path_to_sdk_inclusion`**—Replace with the path required to include the AWS SDK for PHP in the program. For more information, see the AWS SDK for PHP documentation.
   - **`sender@example.com`**—Replace with an email address that you have verified with Amazon SES. For more information, see Verified identities. Email addresses in Amazon SES are case-sensitive. Make sure that the address you enter is exactly the same as the one you verified.
   - **`recipient1@example.com`, `recipient2@example.com`**—Replace with the addresses of your recipients. If your account is still in the sandbox, your recipients' addresses must also be verified. For more information, see Request production access (Moving out of the Amazon SES sandbox). Make sure that the address you enter is exactly the same as the one you verified.
   - **(Optional) `ConfigSet`**—If you want to use a configuration set when sending this email, replace this value with the name of the configuration set. For more information about configuration sets, see Using configuration sets in Amazon SES.

- **(Optional) `us-west-2`**—If you want to use Amazon SES in a Region other than US West (Oregon), replace this with the Region you want to use. For a list of Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

3. Save `amazon-ses-sample.php`.
4. To run the program, open a command prompt in the same directory as `amazon-ses-sample.php`, and then type the following command:

```
$ php amazon-ses-sample.php
```

5. Review the output. If the email was successfully sent, the console displays "`Email sent!`" Otherwise, it displays an error message.

> **Note**
>
> If you encounter a "cURL error 60: SSL certificate problem" error when you run the program, download the latest CA bundle as described in the AWS SDK for PHP documentation. Then, in `amazon-ses-sample.php`, add the following lines to the `SesClient::factory` array, replace `path_of_certs` with the path to the CA bundle you downloaded, and re-run the program.
>
> ```
> 'http' => [
>    'verify' => 'path_of_certs\ca-bundle.crt'
> ]
> ```

6. Sign in to the email client of the recipient address. You will see the message that you sent.

Ruby

This topic shows how to use the AWS SDK for Ruby to send an email through Amazon SES.

Before you begin, perform the following tasks:

- **Install Ruby**—Ruby is available at https://www.ruby-lang.org/en/downloads/. The code in this tutorial was tested using Ruby 1.9.3. After you install Ruby, add the path to Ruby in your environment variables so that you can run Ruby from any command prompt.

- **Install the AWS SDK for Ruby**—For download and installation instructions, see Installing the AWS SDK for Ruby in the *AWS SDK for Ruby Developer Guide*. The sample code in this tutorial was tested using version 2.9.36 of the AWS SDK for Ruby.
- **Create a shared credentials file**—For the sample code in this section to function properly, you must create a shared credentials file. For more information, see Creating a shared credentials file to use when sending email through Amazon SES using an AWS SDK.

To send an email through Amazon SES using the AWS SDK for Ruby

1. In a text editor, create a file named `amazon-ses-sample.rb`. Paste the following code into the file:

```
require 'aws-sdk'

# Replace sender@example.com with your "From" address.
# This address must be verified with Amazon SES.
sender = "sender@example.com"

# Replace recipient@example.com with a "To" address. If your account
# is still in the sandbox, this address must be verified.
recipient = "recipient@example.com"

# Specify a configuration set. If you do not want to use a configuration
# set, comment the following variable and the
# configuration_set_name: configsetname argument below.
configsetname = "ConfigSet"

# Replace us-west-2 with the AWS Region you're using for Amazon SES.
awsregion = "us-west-2"

# The subject line for the email.
subject = "Amazon SES test (AWS SDK for Ruby)"

# The HTML body of the email.
htmlbody =
  '<h1>Amazon SES test (AWS SDK for Ruby)</h1>'\
  '<p>This email was sent with <a href="ses/">'\
  'Amazon SES</a> using the <a href="sdk-for-ruby/">'\
  'AWS SDK for Ruby</a>.'

# The email body for recipients with non-HTML email clients.
```

```
textbody = "This email was sent with Amazon SES using the AWS SDK for Ruby."

# Specify the text encoding scheme.
encoding = "UTF-8"

# Create a new SES resource and specify a region
ses = Aws::SES::Client.new(region: awsregion)

# Try to send the email.
begin

  # Provide the contents of the email.
  resp = ses.send_email({
    destination: {
      to_addresses: [
        recipient,
      ],
    },
    message: {
      body: {
        html: {
          charset: encoding,
          data: htmlbody,
        },
        text: {
          charset: encoding,
          data: textbody,
        },
      },
      subject: {
        charset: encoding,
        data: subject,
      },
    },
  source: sender,
  # Comment or remove the following line if you are not using
  # a configuration set
  configuration_set_name: configsetname,
  })
  puts "Email sent!"

# If something goes wrong, display an error message.
rescue Aws::SES::Errors::ServiceError => error
  puts "Email not sent. Error message: #{error}"
```

```
end
```

2. In `amazon-ses-sample.rb`, replace the following with your own values:

   - **`sender@example.com`**—Replace with an email address that you have verified with Amazon SES. For more information, see Verified identities. Email addresses in Amazon SES are case-sensitive. Make sure that the address you enter is exactly the same as the one you verified.
   - **`recipient@example.com`**—Replace with the address of the recipient. If your account is still in the sandbox, you must verify this address before you use it. For more information, see Request production access (Moving out of the Amazon SES sandbox). Make sure that the address you enter is exactly the same as the one you verified.
   - **(Optional) `us-west-2`**—If you want to use Amazon SES in a Region other than US West (Oregon), replace this with the Region you want to use. For a list of Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

3. Save `amazon-ses-sample.rb`.
4. To run the program, open a command prompt in the same directory as `amazon-ses-sample.rb`, and type **ruby amazon-ses-sample.rb**
5. Review the output. If the email was successfully sent, the console displays `"Email sent!"` Otherwise, it displays an error message.
6. Sign in to the email client of the recipient address. You will find the message that you sent.

Python

This topic shows how to use the AWS SDK for Python (Boto3) to send an email through Amazon SES.

Before you begin, perform the following tasks:

- **Verify your email address with Amazon SES**—Before you can send an email with Amazon SES, you must verify that you own the sender's email address. If your account is still in the Amazon SES sandbox, you must also verify the recipient email address. We recommend you use the Amazon SES console to verify email addresses. For more information, see Creating an email address identity.

- **Get your AWS credentials**—You need an AWS access key ID and AWS secret access key to access Amazon SES using an SDK. You can find your credentials by using the Security Credentials page of the AWS Management Console. For more information about credentials, see Types of Amazon SES credentials.
- **Install Python**—Python is available at https://www.python.org/downloads/. The code in this tutorial was tested using Python 2.7.6 and Python 3.6.1. After you install Python, add the path to Python in your environment variables so that you can run Python from any command prompt.
- **Install the AWS SDK for Python (Boto3)**—For download and installation instructions, see the AWS SDK for Python (Boto3) documentation. The sample code in this tutorial was tested using version 1.4.4 of the SDK for Python (Boto3).

To send an email through Amazon SES using the SDK for Python (Boto3)

1. In a text editor, create a file named `amazon-ses-sample.py`. Paste the following code into the file:

```
import boto3
from botocore.exceptions import ClientError

# Replace sender@example.com with your "From" address.
# This address must be verified with Amazon SES.
SENDER = "Sender Name <sender@example.com>"

# Replace recipient@example.com with a "To" address. If your account
# is still in the sandbox, this address must be verified.
RECIPIENT = "recipient@example.com"

# Specify a configuration set. If you do not want to use a configuration
# set, comment the following variable, and the
# ConfigurationSetName=CONFIGURATION_SET argument below.
CONFIGURATION_SET = "ConfigSet"

# If necessary, replace us-west-2 with the AWS Region you're using for Amazon
 SES.
AWS_REGION = "us-west-2"

# The subject line for the email.
SUBJECT = "Amazon SES Test (SDK for Python (Boto3))"
```

```
# The email body for recipients with non-HTML email clients.
BODY_TEXT = ("Amazon SES Test (Python)\r\n"
             "This email was sent with Amazon SES using the "
             "AWS SDK for Python (Boto3)."
            )

# The HTML body of the email.
BODY_HTML = """<html>
<head></head>
<body>
  <h1>Amazon SES Test (SDK for Python (Boto3))</h1>
  <p>This email was sent with
    <a href='ses/'>Amazon SES</a> using the
    <a href='sdk-for-python/'>
      AWS SDK for Python (Boto3)</a>.</p>
</body>
</html>
            """

# The character encoding for the email.
CHARSET = "UTF-8"

# Create a new SES resource and specify a region.
client = boto3.client('ses',region_name=AWS_REGION)

# Try to send the email.
try:
    #Provide the contents of the email.
    response = client.send_email(
        Destination={
            'ToAddresses': [
                RECIPIENT,
            ],
        },
        Message={
            'Body': {
                'Html': {
                    'Charset': CHARSET,
                    'Data': BODY_HTML,
                },
                'Text': {
                    'Charset': CHARSET,
                    'Data': BODY_TEXT,
                },
```

```
            },
            'Subject': {
                'Charset': CHARSET,
                'Data': SUBJECT,
            },
        },
        Source=SENDER,
        # If you are not using a configuration set, comment or delete the
        # following line
        ConfigurationSetName=CONFIGURATION_SET,
    )
# Display an error if something goes wrong.
except ClientError as e:
    print(e.response['Error']['Message'])
else:
    print("Email sent! Message ID:"),
    print(response['MessageId'])
```

2. In `amazon-ses-sample.py`, replace the following with your own values:

   - **`sender@example.com`**—Replace with an email address that you have verified with Amazon SES. For more information, see Verified identities. Email addresses in Amazon SES are case sensitive. Make sure that the address you enter is exactly the same as the one you verified.
   - **`recipient@example.com`**—Replace with the address of the recipient. If your account is still in the sandbox, you must verify this address before you use it. For more information, see Request production access (Moving out of the Amazon SES sandbox). Make sure that the address you enter is exactly the same as the one you verified.
   - **(Optional) `us-west-2`**—If you want to use Amazon SES in a Region other than US West (Oregon), replace this with the Region you want to use. For a list of Regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.

3. Save `amazon-ses-sample.py`.
4. To run the program, open a command prompt in the same directory as `amazon-ses-sample.py`, and then type **python amazon-ses-sample.py**.
5. Review the output. If the email was successfully sent, the console displays "`Email sent!`" Otherwise, it displays an error message.
6. Sign in to the email client of the recipient address. You will see the message that you sent.

Creating a shared credentials file to use when sending email through Amazon SES using an AWS SDK

The following procedure shows how to create a shared credentials file in your home directory. For the SDK sample code to function properly, you must create this file.

1. In a text editor, create a new file. In the file, paste the following code:

```
[default]
aws_access_key_id = YOUR_AWS_ACCESS_KEY_ID
aws_secret_access_key = YOUR_AWS_SECRET_ACCESS_KEY
```

2. In the text file you just created, replace `YOUR_AWS_ACCESS_KEY` with your unique AWS access key ID, and replace `YOUR_AWS_SECRET_ACCESS_KEY` with your unique AWS secret access key.
3. Save the file. The following table shows the correct location and file name for your operating system.

If you're using...	Save the file as...
Windows	`C:\Users\<yourUserName>\.aws\credentials`
Linux, macOS, or Unix	`~/.aws/credentials`

> **Important**
>
> Don't include a file extension when saving the credentials file.

Content encodings supported by Amazon SES

The following is provided for reference.

Amazon SES supports the following content encodings:

- `deflate`
- `gzip`

- `identity`

Amazon SES also supports the following Accept-Encoding header format, according to the RFC 7231 specification:

- `Accept-Encoding:deflate,gzip`
- `Accept-Encoding:`
- `Accept-Encoding:*`
- `Accept-Encoding:deflate;q=0.5,gzip;q=1.0`
- `Accept-Encoding:gzip;q=1.0,identity;q=0.5,*;q=0`

Using configuration overrides in Amazon SES

Most settings that control how Amazon SES processes your email are defined before you send it, at the account level or in a configuration set. That works well for settings that are stable across a mail stream, but not for settings that change from one recipient to the next. For example, tracking consent is one such setting: whether a recipient has agreed to open and click tracking is a decision that belongs to the recipient, not to your account.

Configuration overrides let you change selected settings for a single email sending request, without creating a separate configuration set for every combination of settings you need. You specify them in the `ConfigurationOverrides` parameter of the `SendEmail` and `SendBulkEmail` operations. An override applies only to the messages in the request that contains it, and it never changes your account-level settings or the configuration set that the request uses.

What are configuration overrides?

A configuration override is a set of values that take precedence, for one request only, over the values that otherwise apply to the messages in that request. Each member of the `ConfigurationOverrides` parameter is a category of settings that you can override. You include only the categories that you want to change, and settings that you don't override keep the values that your account-level and configuration set settings define.

Key benefits of using configuration overrides include:

- Changing a setting for a single request, without maintaining a separate configuration set for every combination of settings that you need.

- Keeping a single configuration set per mail stream, rather than one for every combination of sending domain, dedicated IP pool, event destination, and the settings that change from one request to the next.

> **Note**
>
> Configuration overrides apply to the whole request. In a `SendBulkEmail` request, the override applies to every entry in the request. To send with different settings, group your recipients by the setting that you want to override and send one request per group.

SES supports the following override categories:

Category	Settings that you can override
`Tracking`	Open and click tracking for the messages in the request, through the `OpenTrackingEnabled` and `ClickTrackingEnabled` settings.

How tracking overrides work

Common uses of tracking overrides include:

- Honoring per-recipient tracking consent, including requirements such as GDPR and the CNIL guidance on tracking pixels, without maintaining one configuration set per consent state.
- Suppressing tracking for individual messages, for example, a transactional receipt sent from a configuration set that otherwise has tracking enabled.

To understand what a tracking override changes, it helps to know which settings control tracking in the absence of one. Without an override, open and click tracking are enabled when either of the following is true:

- The configuration set that the message uses has an event destination whose `MatchingEventTypes` include the `OPEN` or `CLICK` event types. For more information, see Monitor email sending using Amazon SES event publishing.

- Engagement metrics are enabled for Virtual Deliverability Manager, either through the `EngagementMetrics` setting of the configuration set, which you configure using the `PutConfigurationSetVdmOptions` operation, or through your account-level `EngagementMetrics` setting, which you configure using the `PutAccountVdmAttributes` operation. The configuration set setting takes precedence over the account-level setting.

When you provide an override, SES uses it in place of the default behavior described above. Each tracking type is resolved on its own, so you can override one and inherit the other. The following table describes each value.

Value	Behavior
`DISABLED`	SES doesn't track the message, even when your account-level or configuration set settings enable tracking. For open tracking, SES doesn't add the tracking image to the message, and removes the `{{ses:openTracker}}` placeholder if the message contains one. For click tracking, SES doesn't rewrite the links in the message.
`ENABLED`	SES tracks the message, even when your account-level and configuration set settings don't enable tracking.
Not specified	SES uses the tracking setting that otherwise applies to the message.

Note

Enabling open or click tracking with an override doesn't create an event destination. SES records the resulting open and click events in Virtual Deliverability Manager, where you can review them using Virtual Deliverability Manager metrics and Message Insights. To also receive these events at a destination that you own, the configuration set that the message uses must have an event destination that publishes open and click events.

> **Note**
>
> An override can enable tracking only for a message that SES is already configured to publish events for. If the configuration set that the message uses has no event destination, and engagement metrics aren't enabled for Virtual Deliverability Manager, setting `OpenTrackingEnabled` or `ClickTrackingEnabled` to `ENABLED` has no effect, and SES sends the message without returning an error. To enable tracking with an override, the configuration set must have an event destination that matches at least one event type, or the other tracking type must already be enabled for the message.

An override changes only the tracking behavior of the messages in the request. It doesn't modify the configuration set, and a subsequent request that uses the same configuration set without an override behaves as it did before.

IAM permissions for tracking overrides

Because a tracking override changes tracking behavior at send time, SES authorizes it separately from the send itself. A request that contains a `Tracking` override requires both of the following:

- Permission to send the email, such as `ses:SendEmail` or `ses:SendBulkEmail` on the identity and configuration set that the request uses.
- Permission for the `ses:ApplyTrackingConfigurationOverrides` action. This action has no SES resource of its own, so you must grant it on `"Resource": "*"`.

If a principal is allowed to send but not allowed to apply tracking overrides, SES returns an `AccessDeniedException` and doesn't send the message. Requests that don't contain `ConfigurationOverrides` are unaffected, so adding this action doesn't change the behavior of your existing sending permissions.

The following policy allows a principal to send email and to apply any tracking override:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSend",
      "Effect": "Allow",
```

```
      "Action": [
        "ses:SendEmail",
        "ses:SendBulkEmail"
      ],
      "Resource": [
        "arn:aws-eusc:ses:us-east-1:111122223333:identity/example.com",
        "arn:aws-eusc:ses:us-east-1:111122223333:configuration-set/my-configuration-
set"
      ]
    },
    {
      "Sid": "AllowTrackingOverrides",
      "Effect": "Allow",
      "Action": "ses:ApplyTrackingConfigurationOverrides",
      "Resource": "*"
    }
  ]
}
```

Make the following changes to the preceding policy example:

- Replace *us-east-1* with the AWS Region that you send from.
- Replace *111122223333* with your AWS account ID.
- Replace *example.com* and *my-configuration-set* with the identity and configuration set that you send with.

Limiting which tracking overrides a principal can request

To allow some tracking overrides but not others, use the following condition keys with the `ses:ApplyTrackingConfigurationOverrides` action. Each key carries the value that the request supplied.

- `ses:OpenTrackingEnabled`
- `ses:ClickTrackingEnabled`

The following policy allows a principal to disable tracking, but not enable it:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "AllowDisablingTrackingOnly",
      "Effect": "Allow",
      "Action": "ses:ApplyTrackingConfigurationOverrides",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ses:OpenTrackingEnabled": "DISABLED",
          "ses:ClickTrackingEnabled": "DISABLED"
        }
      }
    }
  ]
}
```

Important

Only the tracking values that a request actually supplies are present in the request context. Because the preceding policy conditions on both keys, it allows a request that sets both `OpenTrackingEnabled` and `ClickTrackingEnabled` to `DISABLED`, but denies a request that sets only one of them. To allow a request that overrides either tracking type on its own, write a separate statement for each key, or use the `Null` condition operator to account for a key that isn't present.

The following statements deny enabling either tracking type, while still allowing a request to override one tracking type on its own. Because each key is always either `ENABLED` or `DISABLED`, matching `ENABLED` is enough to deny enabling tracking. Grant the sending permissions separately, scoped to the identity and configuration set that the request uses, as shown in the previous policy example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTrackingOverrides",
      "Effect": "Allow",
      "Action": "ses:ApplyTrackingConfigurationOverrides",
      "Resource": "*"
    },
    {
```

```
      "Sid": "DenyEnablingOpenTracking",
      "Effect": "Deny",
      "Action": "ses:ApplyTrackingConfigurationOverrides",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ses:OpenTrackingEnabled": "ENABLED"
        }
      }
    },
    {
      "Sid": "DenyEnablingClickTracking",
      "Effect": "Deny",
      "Action": "ses:ApplyTrackingConfigurationOverrides",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ses:ClickTrackingEnabled": "ENABLED"
        }
      }
    }
  ]
}
```

Each tracking type needs its own statement, because multiple keys in a single condition block are combined with AND. With these statements:

- A request that sets only `ClickTrackingEnabled` to `DISABLED` is authorized to apply the override. `StringEquals` doesn't match a key that isn't present, so the open tracking `Deny` doesn't apply.
- A request that sets either tracking type to `ENABLED` is denied.
- A request that contains no `ConfigurationOverrides` is unaffected, because SES doesn't authorize the override action for it.

To match on whether an override is present in the request at all, use the `Null` condition operator. A value of `false` means "the key is present in the request." The following statement grants the override action only to requests that state an open tracking preference explicitly:

```
{
  "Sid": "AllowTrackingOverridesOnlyWhenOpenTrackingIsSpecified",
```

```
  "Effect": "Allow",
  "Action": "ses:ApplyTrackingConfigurationOverrides",
  "Resource": "*",
  "Condition": {
    "Null": {
      "ses:OpenTrackingEnabled": "false"
    }
  }
}
```

A request that overrides only click tracking doesn't satisfy this condition, so it isn't granted the override action. Use a value of `true` to match the opposite case, where the key isn't present in the request.

Using tracking overrides

To override tracking for a request, add the `Tracking` member of the `ConfigurationOverrides` parameter to your send request. The following example uses the AWS CLI to send a message with both open and click tracking disabled, from a configuration set that has tracking enabled:

```
aws sesv2 send-email \
    --from-email-address "sender@example.com" \
    --destination "ToAddresses=recipient@example.com" \
    --content "Simple={Subject={Data=Test
 email,Charset=UTF-8},Body={Html={Data=<html><body><p>Hello</p></body></
html>,Charset=UTF-8}}}" \
    --configuration-set-name my-configuration-set \
    --configuration-overrides '{"Tracking":
{"OpenTrackingEnabled":"DISABLED","ClickTrackingEnabled":"DISABLED"}}'
```

You can override one tracking type and inherit the other. The following example disables click tracking while leaving open tracking as the configuration set defines it:

```
--configuration-overrides '{"Tracking":{"ClickTrackingEnabled":"DISABLED"}}'
```

The same parameter applies to `SendBulkEmail`, where the override applies to every entry in the request:

```
aws sesv2 send-bulk-email \
    --from-email-address "sender@example.com" \
```

```
    --default-content '{"Template":{"TemplateName":"my-template","TemplateData":"{}"}}'
 \
    --bulk-email-entries '[{"Destination":{"ToAddresses":["recipient1@example.com"]}},
{"Destination":{"ToAddresses":["recipient2@example.com"]}}]' \
    --configuration-set-name my-configuration-set \
    --configuration-overrides '{"Tracking":
{"OpenTrackingEnabled":"DISABLED","ClickTrackingEnabled":"DISABLED"}}'
```

In the preceding examples, replace *my-configuration-set* and *my-template* with the names of your configuration set and email template, and replace the sender and recipient addresses with your own.

Effect of tracking overrides on events and metrics

When an override disables tracking for a message, SES doesn't generate open or click events for it, so no `Open` or `Click` events are published to the event destinations of the configuration set for that message. Other event types, such as `Send`, `Delivery`, and `Bounce`, are unaffected.

Delivery, bounce, and complaint reporting are not affected by tracking overrides, so you can continue to monitor the deliverability of messages that you send without tracking.

Best practices and considerations

Tracking overrides change open and click tracking for a single send request. Follow these best practices to apply them effectively:

- Treat the override as the expression of a recipient's consent, and derive it from the consent state that you store for that recipient rather than hard-coding it per mail stream.
- Group recipients by tracking preference and send one request per group. An override applies to every message in the request.
- Keep your configuration sets organized around settings that are stable for a mail stream, such as the dedicated IP pool, event destinations, and the custom domain used for tracking. Use overrides for the settings that change per recipient.
- Use the `ses:OpenTrackingEnabled` and `ses:ClickTrackingEnabled` condition keys to constrain what a principal can request. For example, an application that only ever suppresses tracking doesn't need permission to enable it.
- To disable click tracking for specific links rather than for a whole message, use the `ses:no-track` attribute instead. For more information, see Q2. Can I disable click tracking?.

Amazon SES and security protocols

This topic describes the security protocols that you can use when you connect to Amazon SES, and when Amazon SES delivers an email to a receiver.

Email sender to Amazon SES

The security protocol that you use to connect to Amazon SES depends on whether you are using the Amazon SES API or the Amazon SES SMTP interface, as described next.

HTTPS

If you're using the Amazon SES API (either directly or through an AWS SDK), then all communications are encrypted by TLS through the Amazon SES HTTPS endpoint. The Amazon SES HTTPS endpoint supports TLS 1.2 and TLS 1.3.

SMTP interface

If you are accessing Amazon SES through the SMTP interface, you're required to encrypt your connection using Transport Layer Security (TLS). Note that TLS is often referred to by the name of its predecessor protocol, Secure Sockets Layer (SSL).

Amazon SES supports two mechanisms for establishing a TLS-encrypted connection: STARTTLS and TLS Wrapper.

- **STARTTLS**—STARTTLS is a means of upgrading an unencrypted connection to an encrypted connection. There are versions of STARTTLS for a variety of protocols; the SMTP version is defined in RFC 3207. For STARTTLS connections, Amazon SES supports TLS 1.2 and TLS 1.3.
- **TLS Wrapper**—TLS Wrapper (also known as SMTPS or the Handshake Protocol) is a means of initiating an encrypted connection without first establishing an unencrypted connection. With TLS Wrapper, the Amazon SES SMTP endpoint does not perform TLS negotiation: it is the client's responsibility to connect to the endpoint using TLS, and to continue using TLS for the entire conversation. TLS Wrapper is an older protocol, but many clients still support it. For TLS Wrapper connections, Amazon SES supports TLS 1.2 and TLS 1.3.

For information about connecting to the Amazon SES SMTP interface using these methods, see Connecting to an Amazon SES SMTP endpoint.

Amazon SES to receiver

While TLS 1.3 is our default delivery method, SES can deliver email to mail servers using earlier versions of TLS.

By default, Amazon SES uses *opportunistic TLS*. Opportunistic TLS in SES always uses STARTTLS and does not include the TLS Wrapper. The flow involves establishing an initial plaintext connection, followed by upgrading to a TLS-encrypted session if both the client and server support STARTTLS. If SES can't establish a secure connection, it sends the message unencrypted.

You can change this behavior by using configuration sets. Use the PutConfigurationSetDeliveryOptions API operation to set the `TlsPolicy` property for a configuration set to `Require`. You can use the AWS CLI to make this change.

To configure Amazon SES to require TLS connections for a configuration set

- At the command line, enter the following command:

  ```
  aws sesv2 put-configuration-set-delivery-options --configuration-set-
  name MyConfigurationSet --tls-policy REQUIRE
  ```

  In the preceding example, replace *MyConfigurationSet* with the name of your configuration set.

  When you send an email using this configuration set, Amazon SES only sends the message to the receiving email server if it can establish a secure connection. If Amazon SES can't make a secure connection to the receiving email server, it drops the message.

End-to-end encryption

You can use Amazon SES to send messages that are encrypted using S/MIME or PGP. Messages that use these protocols are encrypted by the sender. Their contents can only be viewed by recipients who possess the private keys that are required to decrypt the messages.

Amazon SES supports the following MIME types, which you can use to send S/MIME encrypted email:

- `application/pkcs7-mime`
- `application/pkcs7-signature`

- `application/x-pkcs7-mime`
- `application/x-pkcs7-signature`

Amazon SES also supports the following MIME types, which you can use to send PGP-encrypted email:

- `application/pgp-encrypted`
- `application/pgp-keys`
- `application/pgp-signature`

Amazon SES header fields

Amazon SES can accept all email headers that follow the format described in RFC 822.

The following fields can't appear more than once in the header section of a message:

- `Accept-Language`
- `acceptLanguage`
- `Archived-At`
- `Auto-Submitted`
- `Bounces-to`
- `Comments`
- `Content-Alternative`
- `Content-Base`
- `Content-Class`
- `Content-Description`
- `Content-Disposition`
- `Content-Duration`
- `Content-ID`
- `Content-Language`
- `Content-Length`
- `Content-Location`

- Content-MD5
- Content-Transfer-Encoding
- Content-Type
- Date
- Delivered-To
- Disposition-Notification-Options
- Disposition-Notification-To
- DKIM-Signature
- DomainKey-Signature
- Errors-To
- From
- Importance
- In-Reply-To
- Keywords
- List-Archive
- List-Help
- List-Id
- List-Owner
- List-Post
- List-Subscribe
- List-Unsubscribe
- List-Unsubscribe-Post
- Message-Context
- Message-ID
- MIME-Version
- Organization
- Original-From
- Original-Message-ID
- Original-Recipient

- Original-Subject
- Precedence
- Priority
- References
- Reply-To
- Return-Path
- Return-Receipt-To
- Sender
- Solicitation
- Sensitivity
- Subject
- Thread-Index
- Thread-Topic
- User-Agent
- VBR-Info

Considerations

- The acceptLanguage field is non-standard. If possible, you should use the Accept-Language header instead.
- If you specify a Date header, Amazon SES overrides it with a timestamp that corresponds to the date and time in the UTC time zone when Amazon SES accepted the message.
- If you provide a Message-ID header, Amazon SES overrides the header with its own value.
- If you specify a Return-Path header, Amazon SES sends bounce and complaint notifications to the address that you specified. However, the message that your recipients receive contains a different value for the Return-Path header.
- If you use the Amazon SES API v2 SendEmail operation with either *Simple* or *Templated* content, or use the SendBulkEmail operation, you cannot set custom header content for headers that are set by SES; therefore, the following headers are disallowed as custom headers:
  - BCC, CC, Content-Disposition, Content-Type, Date, From, Message-ID, MIME-Version, Reply-To, Return-Path, Subject, To

Working with email attachments in SES

Email attachments in SES are files that you can include with your email messages when using the SES API v2 `SendEmail` and `SendBulkEmail` operations. This feature enables you to enrich your email content by including documents such as PDFs, Word files, images, or other file types that comply with SES supported MIME types. You can also include inline images that render directly in the email content without requiring recipients to download them separately. You can include multiple attachments per email, up to the 40MB total message size limit.

> **Note**
>
> `SendEmail` SES API v2 with `Raw` content type, SMTP interface, and SES API v1 continue to handle attachments through raw email MIME message construction.

How attachments work in SES

There are two different types of encoding that happen at different stages when sending an email with attachments:

Stage 1 – Sending data to SES:

- When you want to send an attachment to SES, the binary data (like a PDF or image) needs to be converted into a format that can be transmitted safely.
- This is where base64-encoding comes in—it's required because you can't send raw binary data in a JSON request.
- If you're using the AWS SDK, it handles this encoding automatically.
- If you're using the AWS CLI, you need to base64-encode the attachment yourself before sending it.

Stage 2 – SES creating the email:

- Once SES receives your data, it needs to create an actual email with the attachment.
- This is where the ContentTransferEncoding setting comes into play.
- SES will use whatever encoding method you specify in ContentTransferEncoding to automatically format the attachment in the final email.

Think of it like this—it's similar to sending a package through the mail. First, you need to get the package to the post office (Stage 1 - Base64-encoding required), then the post office will package it appropriately for final delivery (Stage 2 - ContentTransferEncoding).

Attachment object structure

When you send an email with attachments through SES, the service handles the complex MIME message construction automatically. You simply need to provide the attachment content and metadata through the following the SES API v2 `Attachment` object structure:

- `FileName` (Required) – The file name displayed to recipients (must include file extension). If not provided, SES will derive a `ContentType` from the extension of the `FileName`.
- `ContentType` (Optional) – IANA-compliant media type identifier.
- `ContentDisposition` (Optional) – Specifies how the attachment should be rendered: `ATTACHMENT` *(default)* or `INLINE`.
- `ContentDescription` (Optional) – Short description of the content.
- `RawContent` (Required) – The actual content of the attachment.
- `ContentTransferEncoding` (Optional) – Specifies how the attachment payload is encoded when it's assembled into the email's mime message: `SEVEN_BIT` *(default)*, `BASE64` or `QUOTED_PRINTABLE`.

All attached content must be encoded to base64 before transferring to the SES endpoint for sending. If you're using the AWS SDK client to make API calls, this is automatically handled for you. If you're using the AWS CLI, or have implemented your own client, you will have to do the encoding yourself, such as:

- Plain text content: `Text attachment sample content.`
- Base64 encoded: `VGV4dCBhdHRhY2htZW50IHNhbXBsZSBjb250ZW50Lg==`

The following examples show how to use the attachment object structure when specifying attachments with the SES API v2 `SendEmail` and `SendBulkEmail` operations using the AWS CLI referencing a JSON file containing attachment object elements.

Example– SendEmail with simple content

```
aws sesv2 send-email --cli-input-json file://request-send-email-simple.json
```

request-send-email-simple.json

```
{
    "FromEmailAddress": "sender@example.com",
    "Destination": {
        "ToAddresses": [
            "recipient@example.com"
        ]
    },
    "Content": {
        "Simple": {
            "Subject": {
                "Data": "Email with attachment"
            },
            "Body": {
                "Text": {
                    "Data": "Please see attached document."
                },
                "Html": {
                    "Data": "Please see attached <b>document</b>."
                }
            },
            "Attachments": [
                {
                    "RawContent": "<base64-encoded-content>",
                    "ContentDisposition": "ATTACHMENT",
                    "FileName": "document.pdf",
                    "ContentDescription": "PDF Document Attachment",
                    "ContentTransferEncoding": "BASE64"
                }
            ]
        }
    }
}
```

Example– SendEmail with simple content and inline attachment

```
aws sesv2 send-email --cli-input-json file://request-send-email-simple-inline-
attachment.json
```

request-send-email-simple-inline-attachment.json

```
{
```

```
    "FromEmailAddress": "sender@example.com",
    "Destination": {
        "ToAddresses": [
            "recipient@example.com"
        ]
    },
    "Content": {
        "Simple": {
            "Subject": {
                "Data": "Email with attachment"
            },
            "Body": {
                "Html": {
                    "Data": "<html><body>Our logo:<br><img src=\"cid:logo123\" alt=
\"Company Logo\"></body></html>"
                }
            },
            "Attachments": [
                {
                    "RawContent": "<base64-encoded-content>",
                    "ContentDisposition": "INLINE",
                    "FileName": "logo.png",
                    "ContentId": "logo123",
                    "ContentTransferEncoding": "BASE64"
                }
            ]
        }
    }
}
```

Example– SendEmail with template content

```
aws sesv2 send-email --cli-input-json file://request-send-email-template.json
```

request-send-email-template.json

```
{
    "FromEmailAddress": "sender@example.com",
    "Destination": {
        "ToAddresses": [
            "recipient@example.com"
        ]
    },
```

```
    "Content": {
        "Template": {
            "TemplateName": "MyTemplate",
            "TemplateData": "{\"name\":\"John\"}",
            "Attachments": [
                {
                    "RawContent": "<base64-encoded-content>",
                    "ContentDisposition": "ATTACHMENT",
                    "FileName": "document.pdf",
                    "ContentDescription": "PDF Document Attachment",
                    "ContentTransferEncoding": "BASE64"
                }
            ]
        }
    }
}
```

Example– SendBulkEmail with attachment content

```
aws sesv2 send-bulk-email --cli-input-json file://request-send-bulk-email.json
```

request-send-bulk-email.json

```
{
    "FromEmailAddress": "sender@example.com",
    "DefaultContent": {
        "Template": {
            "TemplateName": "MyTemplate",
            "TemplateData": "{}",
            "Attachments": [
                {
                    "RawContent": "<base64-encoded-content>",
                    "ContentDisposition": "ATTACHMENT",
                    "FileName": "document.pdf",
                    "ContentDescription": "PDF Document Attachment",
                    "ContentTransferEncoding": "BASE64"
                }
            ]
        }
    },
    "BulkEmailEntries": [
        {
            "Destination": {
```

```
                "ToAddresses": [
                    "recipient@example.com"
                ]
            },
            "ReplacementEmailContent": {
                "ReplacementTemplate": {
                    "ReplacementTemplateData": "{\"name\":\"John\"}"
                }
            }
        }
    ]
}
```

Best practices

- Keep total message size (including attachments) under 40MB.
- Let SES auto-detect content types based on file extensions when possible.
- Explicitly specify content types only when they fall outside of the common MIME types.
- Consider using inline images for better email rendering.
- SES supports a wide range of MIME types for attachments, except for those listed in Unsupported attachment types.

SES unsupported attachment types

You can send messages with attachments through Amazon SES by using the Multipurpose Internet Mail Extensions (MIME) standard. Amazon SES accepts all file attachment types *except* for attachments with the file extensions in the following list.

.ade	.hta	.mau	.mst	.psc1
.adp	.inf	.mav	.ops	.psc2
.app	.ins	.maw	.pcd	.tmp
.asp	.isp	.mda	.pif	.url
.bas	.its	.mdb	.plg	.vb
.bat	.js	.mde	.prf	.vbe

.cer	.jse	.mdt	.prg	.vbs
.chm	.ksh	.mdw	.reg	.vps
.cmd	.lib	.mdz	.scf	.vsmacros
.com	.lnk	.msc	.scr	.vss
.cpl	.mad	.msh	.sct	.vst
.crt	.maf	.msh1	.shb	.vsw
.csh	.mag	.msh2	.shs	.vxd
.der	.mam	.mshxml	.sys	.ws
.exe	.maq	.msh1xml	.ps1	.wsc
.fxp	.mar	.msh2xml	.ps1xml	.wsf
.gadget	.mas	.msi	.ps2	.wsh
.hlp	.mat	.msp	.ps2xml	.xnk

Some ISPs have further restrictions (such as restrictions regarding archived attachments), so we recommend testing your email sending through major ISPs before you send your production email.

Email receiving with Amazon SES

Besides using Amazon SES to manage your email sending, you can also configure SES to receive email on behalf of one or more of your domains. As the email receiver, SES handles underlying mail-receiving operations, such as communicating with other mail servers, scanning for spam and viruses, blocking mail from untrusted sources (addresses on the block lists of either Spamhaus or SES), and accepting mail for recipients in your domain.

The extent of processing on your received email is determined by the custom instructions you specify. These instructions come in two forms:

- **Receipt rules** *(recipient-based control)* provide the finest granularity of control over incoming email. Receipt rules can do advanced processing such as deliver incoming mail to an Amazon S3 bucket, publish it to an Amazon SNS topic, send it to Amazon WorkMail, or automatically send bounce messages when messages are to specific email addresses, and more.
- **IP address filters** *(IP-based control)* provide a broad level of control and are simple to setup. These filters allow you to explicitly block or allow all messages from specific IP addresses or IP address ranges.

To get started with learning about email receiving, setting it up, and implementation using either *receipt rules* or *IP address filters*, first read through Email receiving concepts & use cases to get an overview of how it works and the different ways you can use it. Next, Setting up email receiving will guide you through the email receiving set up prerequisites. Then, the Email receiving console walkthroughs will guide you through the wizards used for configuring *receipt rules* and *IP address filters*.

> **Note**
>
> Email receiving can only be used if your account is in an AWS Region where SES supports email receiving. The Email Receiving endpoints table in the AWS General Reference lists all of the AWS Regions where SES supports email receiving.

Topics in this section:

- Amazon SES email receiving concepts and use cases
- Setting up Amazon SES email receiving

- Amazon SES email receiving console walkthroughs
- Viewing metrics for Amazon SES email receiving

Amazon SES email receiving concepts and use cases

When you use Amazon SES as your email receiver, you tell the service what to do with your mail. The primary method, receipt rules, gives you fine-grained control over your email receiving by utilizing *recipient-based control* to specify a set of actions to take based on the recipient. The other method, IP address filters, provides a broad level of *IP-based control* to block or allow mail based on the originating IP address or range of addresses.

Both of these methods are described in this section along with an overview of how Amazon SES processes received email, and use cases to help you consider how you want to receive, filter, and process your email when setting up rules and filters.

Topics in this section:

- Recipient-based control using receipt rules
- IP-based control using IP address filters
- Email-receiving process
- Use cases and restrictions for Amazon SES email receiving
- Email-receiving authentication and malware scanning

Recipient-based control using receipt rules

The primary way to control your incoming mail is to specify how mail is handled through an ordered list of actions for any of your verified identities which includes domains, sub-domains, or email addresses - note that email addresses have to belong to one of your verified domain identities. These actions are defined and ordered in *receipt rules* that you create within a *rule set*.

As an option, you can also add recipient conditions as a way to specify that the actions only be taken if the recipient to whom the incoming mail is addressed matches a recipient identity specified in the condition. For example, if you own *example.com*, you can specify that mail for *user@example.com* should bounce, and that all other mail for *example.com* and its subdomains should be delivered.

Note

Recipient conditions are evaluated against the *SMTP envelope recipients* (the addresses specified in the `RCPT TO` command during the SMTP transaction), not against the `To:` or `Cc:` headers in the email content. The `To:` and `Cc:` headers are set by the sender and may not reflect the actual envelope recipients. For example, with a blind carbon copy (BCC), a message can be delivered to one of your recipients even though that address does not appear in the `To:` or `Cc:` header. As a result, Amazon SES can correctly match a receipt rule's recipient condition against an envelope recipient that is not visible in the email's headers.

When you process received mail downstream (for example, in a Lambda function), reference the `recipients` field provided in the Amazon SES event notification to determine who the message was actually delivered to, rather than parsing the `To:` or `Cc:` header from the raw email.

Otherwise, if you do not add any recipient conditions, the actions will be applied to everything - all email addresses, domains, and sub-domains that belong to your verified domains. The following actions are available to be applied to your receipt rules:

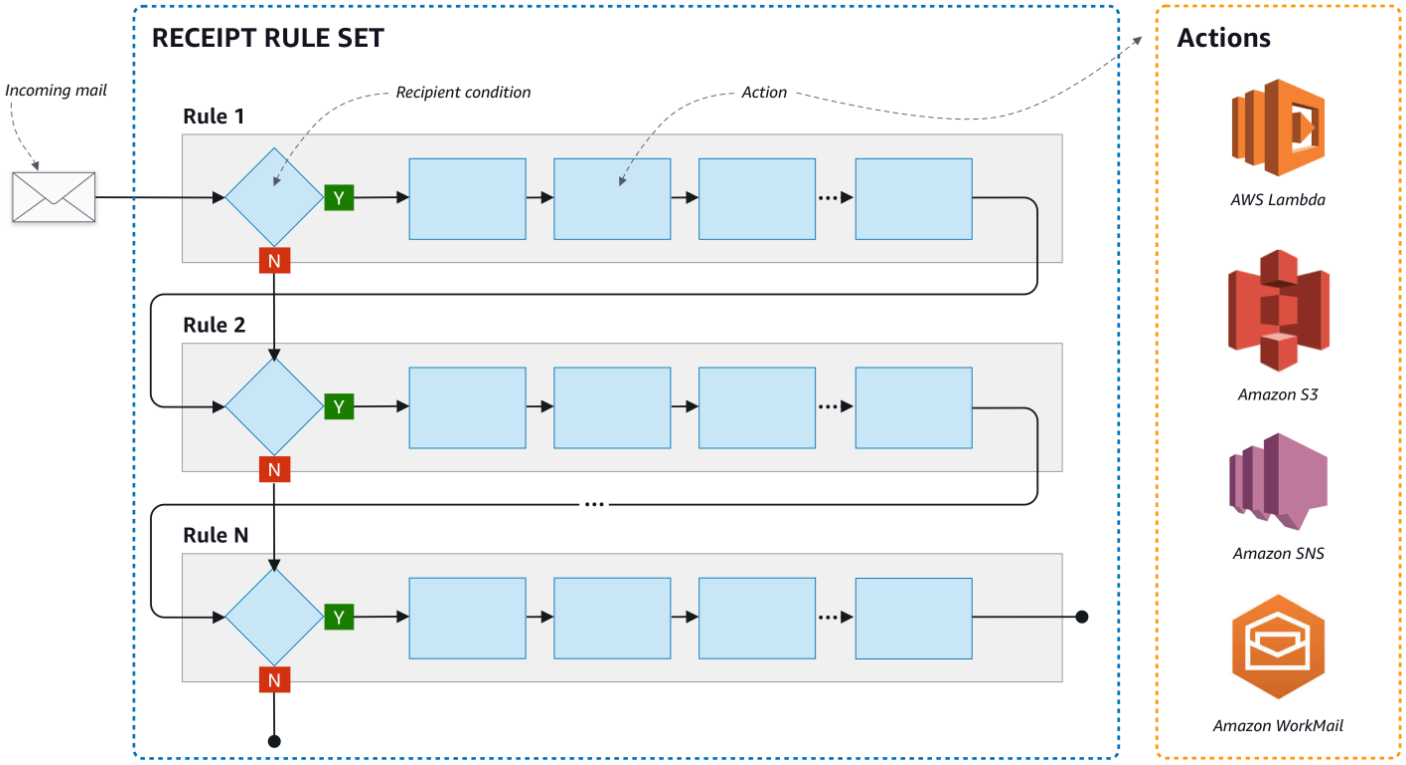
- **Add header action—**Adds a header to the received email. You typically use this action only in combination with other actions.
- **Return bounce response action—**blocks the email by returning a bounce response to the sender and, optionally, notifies you through Amazon SNS.
- **Invoke AWS Lambda function action—**Calls your code through a Lambda function and, optionally, notifies you through Amazon SNS.
- **Deliver to S3 bucket action—**Delivers the mail to an Amazon S3 bucket and, optionally, notifies you through Amazon SNS.
- **Publish to Amazon SNS topic action—**Publishes the complete email to an Amazon SNS topic.

Note

The SNS action includes a complete copy of the email content in the Amazon SNS notifications. The other Amazon SNS notification options mentioned here simply notify you of email delivery; they contain information about the email, not the email content itself.

- **Stop rule set action—**Terminates the evaluation of the receipt rule set and, optionally, notifies you through Amazon SNS.
- **Integrate with Amazon WorkMail action—**Handles the mail with Amazon WorkMail. You will typically not use this action directly because Amazon WorkMail takes care of the setup.

Receipt rules are grouped together into *rule sets*. If you don't have an existing rule set, you'll first have to create a rule set before you start creating receipt rules. You can define multiple rule sets for your AWS account, but only one rule set is active at any time. The following figure shows how receipt rules, rule sets, and actions relate to each other.



IP-based control using IP address filters

You can control your mail flow by setting up **IP address filters**. IP address filters are optional and enable you to specify whether to accept or block mail originating from an IP address or range of IP addresses. Your IP address filters can include *block lists* (IP addresses from which you want to block incoming mail) and *allow lists* (IP addresses from which you want to always accept mail).

IP address filters are useful for blocking spam. Amazon SES maintains its own block list of IP addresses known to send spam including those listed in Spamhaus. However, you can choose to receive mail from those IP addresses by adding them to your allow list. Since there are no logs that

show which IP addresses are being blocked, the sender who is being blocked will need to inform you. This is also a good opportunity to help the sender determine if their IP address is on a block list, such as Spamhaus, and recommend they request to be unlisted. Doing so will be beneficial to both you and the sender in that you won't have to maintain an IP address filter for them and they will improve their email deliverability.

Note

- Independent of your IP address filter configuration, Amazon EC2 will block outbound traffic on port 25 (mail sending) unless allowlisted. Refer to this AWS re:Post article for more information.
- If you only want to receive mail from a finite list of known IP addresses, then set up a block list that contains `0.0.0.0/0`, and set up an allow list that contains the IP addresses that you trust. This configuration blocks all IP addresses by default, and only allows mail from the IP addresses that you explicitly specify.

Email-receiving process

When Amazon SES receives an email for your domain, the following events occur:

1. Amazon SES first looks at the IP address of the sender. Amazon SES allows the mail to pass this stage unless:
   - The IP address is in your block list.
   - The IP address is in the Amazon SES block list, but not on your allow list.
2. Amazon SES examines your active rule set to determine whether any of your receipt rules contain a recipient condition:
   - If there's a recipient condition and it matches any of the incoming email's recipients, Amazon SES accepts the email. Otherwise, if there aren't any matches, Amazon SES blocks the email. The recipients used for matching are the SMTP envelope recipients (`RCPT TO`), not the `To:` or `Cc:` headers in the email content.
   - If the receipt rule does not contain a recipient condition, Amazon SES accepts the mail - all of the rule's actions will apply to all the verified identities you own.
3. Amazon SES authenticates the email and scans its content for spam and malware:

- The IP address of the remote host that delivered the email to Amazon SES is checked against the SPF policy specified under the MAIL FROM's domain used during the SMTP transaction.
- The DKIM signatures present in the email's header section are checked.
- If content scanning is enabled, the email content is scanned for spam and malware.
- The email authentication and content scanning results are made available to you during the receipt rules evaluation.

See Email authentication and malware detection for more information.

4. For the email that Amazon SES accepts, all of the receipt rules within your active rule set are applied in the order you've defined; and within each receipt rule, the actions are executed in the order you've defined.

Use cases and restrictions for Amazon SES email receiving

This section goes over some general considerations and use cases for Amazon SES email receiving. Presented in question and answer format, are commonly asked questions and facts to help determine if it would be beneficial for using Amazon SES to receive and manage email on behalf of one or more of the verified domains that you own.

Regional availability

Does Amazon SES support email receiving in your Region?

Amazon SES only supports email receiving in certain AWS Regions. For a complete list of Regions where email receiving is supported, see Amazon Simple Email Service endpoints and quotas in the AWS General Reference.

POP or IMAP based email clients

Can Microsoft Outlook be used to receive incoming email?

Amazon SES doesn't include POP or IMAP servers for receiving incoming email. This means that you can't use an email client such as Microsoft Outlook to receive incoming email. If you need a solution that can both send and receive email by using an email client, consider using Amazon WorkMail.

Using other AWS services

Have you set up the appropriate permissions?

If you want your mail to be delivered to an S3 bucket, published to an Amazon SNS topic you don't own, trigger a Lambda function, or use a customer managed key, you need to give Amazon SES permission to access those resources. To give Amazon SES access, you create policies on resources from the consoles or APIs for those AWS services. For more information Giving permission.

Email content

How do you want Amazon SES to pass you the email content?

Amazon SES can provide you the email content in two ways: it can store the emails in an S3 bucket that you specify, or it can send you an Amazon SNS notification that contains a copy of the email. Amazon SES delivers you the raw, unmodified email in Multipurpose Internet Mail Extensions (MIME) format. For more information about MIME format, see RFC 2045.

How large are the emails that you'll be receiving?

If you store emails in an S3 bucket, the maximum email size (including headers) is 40 MB. If you receive your emails through Amazon SNS notifications, the maximum email size (including headers) is 150 KB.

How do you want to trigger the processing of your mail?

After your mail is delivered, you will want to process it with your own code. For example, your application might convert the base 64-encoded email into a displayable format and then make it available to an end user through an email client. There are a couple of ways you can start the process:

- If your emails are delivered to Amazon S3, your application can listen for Amazon SNS notifications generated by S3 actions, extract the message ID of the email from the notifications, and then use the message ID to retrieve the email from Amazon S3.

  Alternatively, you can incorporate email processing into your receipt rules by writing a Lambda function. In this case, your receipt rule should first write the email to Amazon S3, and then trigger the Lambda function. Lambda actions can be executed synchronously or asynchronously from within your receipt rules, depending on whether the Lambda function needs to return a result that influences how other actions are executed. We recommend that you use asynchronous execution unless synchronous is absolutely necessary for your use case. For more information about AWS Lambda, see the AWS Lambda Developer Guide.

- If your emails are delivered through an Amazon SNS notification by using the SNS action, your application can listen for Amazon SNS notifications, and then extract the email messages from the notifications.

Do you want the emails to be encrypted?

Amazon SES integrates with AWS Key Management Service (AWS KMS) to optionally encrypt the mail it writes to your S3 bucket. Amazon SES uses client-side encryption to encrypt your mail before writing it to Amazon S3. This means that you must decrypt the content on your side after retrieving the mail from Amazon S3. The AWS SDK for Java and AWS SDK for Ruby provide a client that can handle the decryption for you. Amazon SES can encrypt the emails for you only if you choose for your emails to be delivered to an S3 bucket.

Unwanted mail

At what point in the email-receiving process do you want to block unwanted mail?

When a sender tries to send an email to a recipient, the sender's email server exchanges a sequence of commands with the recipient's server. This sequence is called the *SMTP conversation*.

You can block incoming email at two points in the email receiving process: during the SMTP conversation, and after the SMTP conversation. You use *IP address filters* to block messages during the SMTP conversation, and *receipt rules* to block emails after the SMTP conversation.

You can use IP address filters to block email that originates from specific IP addresses. The benefit of using IP address filters to block unwanted mail is that we don't charge you for messages that are blocked during the SMTP conversation. The drawback to using IP address filters is that they block email from the IP addresses you specify without performing any analysis on the actual content of the messages. For more information about IP address filters, see Create IP address filters console walkthrough.

You can use receipt rules to send a bounce notification to the sender of an email based on the address (or domain, or subdomain) that the message was sent to. The benefit of using receipt rules is that you can perform additional analysis on incoming messages before you send a bounce notification to the sender. For example, you can use AWS Lambda to send bounce notifications only when messages fail DKIM authentication or are identified as spam. The drawback to using receipt rules is that, because receipt rules are processed after the SMTP conversation, we bill you for each message that you receive. You might also be charged if you use Lambda to analyze the content of incoming messages. For more information about receipt rules, see Creating receipt rules console

walkthrough. For more information about using Lambda to analyze incoming email, see Lambda function examples.

Mail streams

How do you want to divide your mail stream?

Your domain most likely receives different classes of mail. For example, some of your domain's mail, such as an email to *user@example.com*, might be intended for a personal inbox. Other mail, such as an email to *unsubscribe@example.com*, might be better directed to automated systems instead. You can use receipt rules to divide your incoming mail so that it can be processed differently. For information about how to set up receipt rules, see Creating receipt rules.

Email-receiving authentication and malware scanning

Amazon SES authenticates each received email and optionally scans the email's content for spam and malware. SES doesn't take any actions on received email based on the results of the email authentication or content scanning; however, the results of these operations are provided to you as attributes that you can use in SES receipt rule actions such as Amazon SNS notifications or as headers in a message delivered to Amazon S3.

Email authentication

Amazon SES authenticates each received email using SPF, DKIM and DMARC. The results of each authentication mechanism is provided in the Amazon SNS notifications that SES dispatches as part of evaluating the rules in the active receipt rule set. In addition, if you chose to receive a copy of the email in Amazon S3, the result of the email authentication is captured in the `Authentication-Results` header that SES adds to the email's header section:

```
Authentication-Results: example.com;
spf=pass (spfCheck: 10.0.0.1 is permitted by domain of example.com) client-ip=10.0.0.1;
 envelope-from=example@example.com; helo=10.0.0.1;
dkim=pass header.i=example.com;
dkim=permerror header.i=some-example.com;
dmarc=pass header.from=example@example.com;
```

The `Authentication-Results` header is described in RFC 8601

Email content scanning for spam and malware detection

Amazon SES scans received email content for malware depending of the value of the *ScanEnabled* (API) or *Spam and virus scanning* (console) attribute of the receipt rule that matched the email. By default SES scans received email content for malware. To disable content scanning for received emails that match a specific receipt rule, you would need to set the receipt rule's *ScanEnabled* flag to false if using the API, or clear the *Spam and virus scanning* checkbox if using the console. If the receipt rule that matched an email is scan enabled, the result of the content scanning is provided in the Amazon SNS notifications that SES dispatches as part of evaluating the rules in the active receipt rule set. In addition, if you chose to receive a copy of the email in Amazon S3, the result of the content scanning is captured in the `X-SES-Spam-Verdict` and the `X-SES-Virus-Verdict` headers that SES adds to the email's header section.

```
X-SES-Spam-Verdict: PASS
X-SES-Virus-Verdict: FAIL
```

The possible values for the headers above are listed in:

- spam
- virus

Now that you have an understanding of the email receiving concepts, how it works, and it's use cases, you can get started by going to Setting up email receiving.

Setting up Amazon SES email receiving

This section describes the prerequisites that are required before you can begin to configure Amazon SES to receive your mail. It's important that you've read Email receiving concepts & use cases to understand the concepts of how Amazon SES works and to consider how you want to receive, filter, and process your email.

Before you can configure email receiving by creating a *rule set*, *receipt rules*, and *IP address filters*, you must first complete the following set up prerequisites:

- Verify your domain with Amazon SES by publishing DNS records to prove that you own it.
- Permit Amazon SES to receive email for your domain by publishing an MX record.
- Give Amazon SES permission to access other AWS resources in order to execute receipt rule actions.

When you create and verify a domain identity, you're publishing records to your DNS settings to complete the verification process, but this alone is not enough to use email receiving. Specific to email receiving, it's also required to publish an MX record for specifying a custom mail-from domain. This record is used in your domain's DNS settings to permit SES to receive email for your domain. Giving permissions is required because the actions you choose in your receipt rules won't work unless Amazon SES has permission to use the respective AWS service required for those actions.

These three prerequisites required to use email receiving are explained in the following topics:

- Verifying your domain for Amazon SES email receiving
- Publishing an MX record for Amazon SES email receiving
- Giving permissions to Amazon SES for email receiving

Verifying your domain for Amazon SES email receiving

As with any domain you want to use for sending or receiving email with Amazon SES, you must first prove that you own it. The verification procedure includes initiating domain verification with SES and then publishing the DNS records, either CNAME or TXT, to your DNS provider depending on which verification method you use.

Through the console, you can verify your domains with either Easy DKIM or Bring Your Own DKIM (BYODKIM) and easily copy their DNS records to publish to your DNS provider - how to do this is explained in Creating a domain identity. Optionally, you can use either the SES `VerifyDomainDkim` or `VerifyDomainIdentity` APIs.

You can easily confirm that your domain or email address is verified by looking at its status in the Verified identities table in the SES console or by using either the SES `GetIdentityVerificationAttributes` or `GetEmailIdentity` APIs.

Publishing an MX record for Amazon SES email receiving

A *mail exchanger* record (*MX record*) is a configuration that specifies which mail servers can accept email that's sent to your domain.

To have Amazon SES manage your incoming email, you need to add an MX record to your domain's DNS configuration. The MX record that you create refers to the endpoint that receives email for the AWS Region where you use Amazon SES. For example, the endpoint for the US West (Oregon)

Region is *inbound-smtp.us-west-2.amazonaws.com*. For a complete list of endpoints, see SES regions and endpoints.

> **Note**
>
> The endpoints that receive email in Amazon SES aren't IMAP or POP3 email servers. You can't use these URLs as incoming mail servers in email clients. If you need a solution that can both send and receive email by using an email client, consider using Amazon WorkMail.

The following procedure includes general steps for creating an MX record. *The specific procedures for creating an MX record depend on your DNS or hosting provider.* See your provider's documentation for information about adding an MX record to the DNS configuration for your domain.

> **Note**

To add an MX record to the DNS configuration for your domain

1. (Prerequisite) To complete these procedures, you will need to modify the DNS records for your domain. If you can't access the DNS records, or you're not comfortable doing so, contact your system administrator for assistance.
2. Sign in to the management console for your DNS provider.
3. Create a new MX record.
4. For the MX record **Name**, enter your domain. For example, if you want Amazon SES to manage email that's sent to the domain *example.com*, enter the following:

   ```
   example.com.
   ```

   > **Note**
   >
   > Depending on your DNS provider: 1) The trailing `.` at the end of the domain extension may not be required. 2) The **Name** field may be referred to as the **Host**, **Domain**, or **Mail Domain**.

5. For **Type**, choose **MX**.

> **Note**
>
> Some DNS providers refer to the **Type** field as the **Record Type** or a similar name.

6. For **Value**, enter the following:

```
10 inbound-smtp.region.amazonaws.com
```

In the preceding example, replace *region* with the address of the endpoint that receives email for the AWS Region you use with Amazon SES. For example, if you're using the US East (N. Virginia) Region, replace *region* with `us-east-1`. For a complete list of email receiving endpoints, see SES regions and endpoints.

> **Note**
>
> The management consoles of some DNS providers include separate fields for the record **Value** and the record **Priority**. If this is the case for your DNS provider, enter `10` for the **Priority** value, and enter the incoming mail endpoint URL for the **Value**.

> **Important**
>
> The specific procedures for creating an MX record depend on your DNS or hosting provider. See your provider's documentation or contact them for information about adding an MX record to the DNS configuration for your domain.

Instructions for creating MX records for various providers

The procedures for creating an MX record for your domain depend on which DNS provider you use. This section includes links to the documentation for several common DNS providers. This list isn't a complete list of providers. If your provider isn't listed below, you can probably still use it with Amazon SES. Inclusion on this list isn't an endorsement or recommendation of any company's products or services.

DNS/Hosting Provider Name	Documentation Link
Amazon Route 53	Creating Records by Using the Amazon Route 53 Console
GoDaddy	Add an MX record (external link)
DreamHost	How do I change my MX records? (external link)
Cloudflare	Set up email records (external link)
HostGator	Changing MX records - Windows (external link)
Namecheap	How can I set up MX records required for mail service? (external link)
Names.co.uk	Changing your domain's DNS settings (external link)
Wix	Adding or Updating MX Records in Your Wix Account (external link)

Giving permissions to Amazon SES for email receiving

Some of the tasks that you can perform when you receive email in SES, such as sending email to an Amazon Simple Storage Service (Amazon S3) bucket or calling a AWS Lambda function, require special permissions. This section includes example policies for several common use cases.

Topics in this section:

- Setting up IAM role permissions for Deliver to S3 bucket action
- Give SES permission to write to an S3 bucket
- Give SES permission to use your AWS KMS key
- Give SES permission to invoke a AWS Lambda function
- Give SES permission to publish to an Amazon SNS topic that belongs to a different AWS account

Setting up IAM role permissions for Deliver to S3 bucket action

The following points are applicable to this IAM role:

- It can only be used for Deliver to S3 bucket action.
- It must be used if want to write to an S3 bucket that exists in a region where SES the section called “Email receiving” isn't available.

If want to write to an S3 bucket, you can provide an IAM role with permissions to access the relevant resources for the Deliver to S3 bucket action. You would also need to give SES permission to assume that role to perform the action through an IAM trust policy as explained in the next section.

This permission policy must be pasted into the IAM role's inline policy editor—see Deliver to S3 bucket action and follow the steps given in the **IAM role** item. (The following example also includes optional permissions in case you want to use SNS topic notification, or a customer managed key in the S3 action.)

> **Note**
>
> - You have the option to set up the S3 action without specifying an IAM role by allowing just the SES service in the S3 bucket policy as shown the section called “Give SES permission to write to an S3 bucket”. This will work for cross-account scenarios as well.
> - If you specify an IAM role for the S3 action, SES assumes that role for 'PutObject' operation, and the IAM permissions specified here will be sufficient for same account usage. However, for cross-account usage, you'll need an additional bucket policy that allows the IAM role to 'PutObject' in the bucket. This is specified by the bucket owner granting cross-account bucket permissions as explained in Bucket owner granting cross-account bucket permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
        },
        {
            "Sid": "SNSAccess",
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "arn:aws-eusc:sns:eusc-de-east-1:111122223333:my-topic"
        },
        {
            "Sid": "KMSAccess",
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey*",
            "Resource": "arn:aws-eusc:kms:eusc-de-east-1::111122223333:key/key-
id"
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *amzn-s3-demo-bucket* with the name of the S3 bucket that you want to write to.
- Replace *region* with the AWS Region where you created the receipt rule.
- Replace *111122223333* with your AWS account ID.
- Replace *my-topic* with the name of the SNS topic that you want to publish notifications to.
- Replace *key-id* with the ID of your KMS key.

Trust policy for S3 action IAM role

The following trust policy should be added into the *Trust relationships* of the IAM role to allow SES to assume that role.

> **Note**
>
> The manual addition of this trust policy is only required if you did not create your IAM role from the SES console using the steps given in the **IAM role** item of the Deliver to S3

bucket action workflow. *When you create the IAM role from the console, this trust policy is automatically generated and applied to the role for you making this step unnecessary.*

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSESAssume",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                  "AWS:SourceAccount":"111122223333",
                  "AWS:SourceArn": "arn:aws-eusc:ses:region:111122223333:receipt-rule-set/rule_set_name:receipt-rule/receipt_rule_name"
                }
            }
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *region* with the AWS Region where you created the receipt rule.
- Replace *111122223333* with your AWS account ID.
- Replace *rule_set_name* with the name of the rule set that contains the receipt rule that contains the deliver to Amazon S3 bucket action.
- Replace *receipt_rule_name* with the name of the receipt rule that contains the deliver to Amazon S3 bucket action.

Give SES permission to write to an S3 bucket

When you apply the following policy to an S3 bucket, it gives SES permission to write to that bucket as long as it exists in a region where SES Email receiving is available—if you want to write to a bucket outside of an *Email receiving* region, see Setting up IAM role permissions for Deliver to S3 bucket action. For more information about creating receipt rules that transfer incoming email to Amazon S3, see Deliver to S3 bucket action.

For more information about attaching policies to S3 buckets, see Using Bucket Policies and User Policies in the *Amazon Simple Storage Service User Guide*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowSESPuts",
      "Effect":"Allow",
      "Principal":{
        "Service":"ses.amazonaws.com"
      },
      "Action":"s3:PutObject",
      "Resource":"arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
      "Condition":{
        "StringEquals":{
          "AWS:SourceAccount":"111122223333",
          "AWS:SourceArn": "arn:aws-eusc:ses:region:111122223333:receipt-rule-
set/rule_set_name:receipt-rule/receipt_rule_name"
        }
      }
    }
  ]
}
```

Make the following changes to the preceding policy example:

- Replace *amzn-s3-demo-bucket* with the name of the S3 bucket that you want to write to.
- Replace *region* with the AWS Region where you created the receipt rule.
- Replace *111122223333* with your AWS account ID.

- Replace *rule_set_name* with the name of the rule set that contains the receipt rule that contains the deliver to Amazon S3 bucket action.
- Replace *receipt_rule_name* with the name of the receipt rule that contains the deliver to Amazon S3 bucket action.

Give SES permission to use your AWS KMS key

In order for SES to encrypt your emails, it must have permission to use the AWS KMS key that you specified when you set up your receipt rule. You can either use the default KMS key (**aws/ses**) in your account, or use a customer managed key that you create. If you use the default KMS key, you don't need to perform any additional steps to give SES permission to use it. If you use a customer managed key, you need to give SES permission to use it by adding a statement to the key's policy.

Use the following policy statement as the key policy to allow SES to use your customer managed key when it receives email on your domain.

```
{
  "Sid": "AllowSESToEncryptMessagesBelongingToThisAccount",
  "Effect": "Allow",
  "Principal": {
    "Service":"ses.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition":{
        "StringEquals":{
          "AWS:SourceAccount":"111122223333",
          "AWS:SourceArn": "arn:aws-eusc:ses:region:111122223333:receipt-rule-
set/rule_set_name:receipt-rule/receipt_rule_name"
        }
      }
}
```

Make the following changes to the preceding policy example:

- Replace *region* with the AWS Region where you created the receipt rule.
- Replace *111122223333* with your AWS account ID.

- Replace *rule_set_name* with the name of the rule set that contains the receipt rule that you've associated with email receiving.
- Replace *receipt_rule_name* with the name of the receipt rule that you've associated with email receiving.

If you're using AWS KMS to send encrypted messages to an S3 bucket with server-side encryption enabled, then you need to add the policy action, `"kms:Decrypt"`. Using the preceding example, adding this action to your policy would appear as follows:

```
{
  "Sid": "AllowSESToEncryptMessagesBelongingToThisAccount",
  "Effect": "Allow",
  "Principal": {
    "Service":"ses.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition":{
        "StringEquals":{
          "AWS:SourceAccount":"111122223333",
          "AWS:SourceArn": "arn:aws-eusc:ses:region:111122223333:receipt-rule-
set/rule_set_name:receipt-rule/receipt_rule_name"
        }
      }
}
```

For more information about attaching policies to AWS KMS keys, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Give SES permission to invoke a AWS Lambda function

To enable SES to call a AWS Lambda function, you can choose the function when you create a receipt rule in the SES console. When you do, SES automatically adds the necessary permissions to the function.

Alternatively, you can use the `AddPermission` operation in the AWS Lambda API to attach a policy to a function. The following call to the `AddPermission` API gives SES permission to invoke

your Lambda function. For more information about attaching policies to Lambda functions, see AWS Lambda Permissions in the *AWS Lambda Developer Guide*.

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "ses.amazonaws.com",
  "SourceAccount": "111122223333",
  "SourceArn": "arn:aws-eusc:ses:region:111122223333:receipt-rule-
set/rule_set_name:receipt-rule/receipt_rule_name",
  "StatementId": "GiveSESPermissionToInvokeFunction"
}
```

Make the following changes to the preceding policy example:

- Replace *region* with the AWS Region where you created the receipt rule.
- Replace *111122223333* with your AWS account ID.
- Replace *rule_set_name* with the name of the rule set that contains the receipt rule where you created your Lambda function.
- Replace *receipt_rule_name* with the name of the receipt rule containing your Lambda function.

Give SES permission to publish to an Amazon SNS topic that belongs to a different AWS account

To publish notifications to a topic in a separate AWS account, you must attach a policy to the Amazon SNS topic. The SNS topic must be in the same Region as the domain and receipt rule set.

The following policy gives SES permission to publish to an Amazon SNS topic in a separate AWS account.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "ses.amazonaws.com"
            },
            "Action": "SNS:Publish",
            "Resource": "arn:aws-eusc:sns:eusc-de-
east-1:111122223333:topic_name",
            "Condition": {
                "StringEquals": {
                "AWS:SourceAccount": "444455556666",
                "AWS:SourceArn": "arn:aws-eusc:ses:eusc-de-
east-1:777788889999:receipt-rule-set/rule_set_name:receipt-rule/rule_name"
                }
            }
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *topic_region* with the AWS Region that the Amazon SNS topic was created in.
- Replace *sns_topic_account_id* with the ID of the AWS account that owns the Amazon SNS topic.
- Replace *topic_name* with the name of the Amazon SNS topic that you want to publish notifications to.
- Replace *aws_account_id* with the ID of the AWS account that is configured to receive email.
- Replace *receipt_region* with the AWS Region where you created the receipt rule.
- Replace *rule_set_name* with the name of the rule set that contains the receipt rule where you created your publish to Amazon SNS topic action.
- Replace *receipt_rule_name* with the name of the receipt rule containing the publish to Amazon SNS topic action.

If your Amazon SNS topic uses AWS KMS for server-side encryption, you have to add permissions to the AWS KMS key policy. You can add permissions by attaching the following policy to the AWS KMS key policy:

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSESToUseKMSKey",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon SES email receiving console walkthroughs

This section describes the email receiving console wizards that are used for configuring *receipt rules* and *IP address filters* to manage your email receiving. Before using the console wizards, it's important that you've read both Email receiving concepts & use cases to understand the concepts of how email receiving works and Setting up email receiving to make sure you've completed the set up prerequisites.

The console wizards for configuring receipt rules and IP address filters are explained in the following:

- Creating receipt rules console walkthrough
- Create IP address filters console walkthrough

Creating receipt rules console walkthrough

This section will walk you through creating and defining receipt rules using the Amazon SES console. The key points to understanding how receipt rules work are:

- *Rule sets* contain an ordered set of receipt rules; *Receipt rules* contain an ordered set of actions.
- Receipt rules tell Amazon SES how to handle incoming mail by executing an ordered list of actions you specify.

- This ordered list of actions can optionally be made dependent on first matching a recipient condition; if not specified, the actions will be applied to all identities that belong to your verified domains.
- Receipt rules are created and defined in a container called a rule set - while you can create multiple rule sets, only one can be active at a time.
- Receipt rules within the active rule set are executed in the order that you specify.
- Before you create your receipt rules, you must first create a *rule set* to contain them.

Optionally, you can use the `CreateReceiptRuleSet` API to create an empty receipt rule set, as described in the Amazon Simple Email Service API Reference. Then, you can use the Amazon SES console or the `CreateReceiptRule` API to add receipt rules to it.

Before proceeding with the walkthrough, please ensure you have met all of the necessary prerequisites that are required in order to use recipient-based email receiving.

Prerequisites

The following prerequisites must be met before proceeding with setting up recipient based email control using receipt rules:

1. Ensure your endpoint is in an AWS Region where Amazon SES supports email receiving. The Email Receiving endpoints table in the AWS General Reference lists the email receiving endpoints for all of the AWS Regions where SES supports email receiving.
2. You first need to create and verify a domain identity in Amazon SES.
3. Next, you need to specify which mail servers can accept mail for your domain by publishing an MX record to your domain's DNS settings. (The MX record should refer to the Amazon SES endpoint that receives mail for the AWS Region where you use Amazon SES.)
4. Lastly, you need to give Amazon SES permission to access other AWS resources in order to execute receipt rule actions.

Creating rule sets and receipt rules

This walkthrough begins by first creating a rule set to contain your rules and progresses into the **Create rule** wizard to create, define, and order your receipt rules. The wizard contains four screens to define rule settings, add recipient conditions, add actions, and to review all your settings.

To create a rule set and receipt rules using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Email Receiving**.

   > **Note**
   >
   > *Email receiving* will not be visible in the left navigation pane of the SES console if your account is in an AWS Region where SES doesn't support email receiving. See the first item listed in the section called “Prerequisites”.

3. Under the **Receipt rule sets** tab in the **Email receiving** pane, choose **Create rule set**.
4. Enter an unique name for your rule set and choose **Create rule set**.
5. Choose **Create rule** and this will open the **Create rule** wizard.
6. On the **Define rule settings** page, under **Receipt rule details**, enter a **Rule name**.
7. For **Status**, only clear the **Enabled** checkbox if you don't want Amazon SES to run this rule after creation; otherwise, leave this option selected.
8. (Optional) Under **Security and protection options**, for **Transport Layer Security (TLS)**, select **Required** if you want Amazon SES to reject incoming messages that aren't sent over a secure connection.
9. (Optional) For **Spam and virus scanning**, select **Enabled** if you want Amazon SES to scan incoming messages for spam and viruses.
10. To proceed to the next step, choose **Next**.
11. (Optional) On the **Add recipient conditions** page, use the following procedure to specify one or more recipient conditions. You can have a maximum of 100 recipient conditions per receipt rule.

    a. Under **Recipient conditions**, choose **Add new recipient condition** to specify the receiving email address or domain to which you want to apply the receipt rule. The following table uses the address *user@example.com* to show how to specify recipient conditions.

If you want to...	Specify the following recipient...	Notes
Match a specific email address.	*user@example.com*	Also matches variation s of the address that contain labels (such as *user +123@example.com* and *user+xyz@example.com*). However, if you specify an address that contains a label, only that specific address is matched.
Match all addresses within a domain, but not those within its subdomains.	*example.com*	
Match all addresses within a specific subdomain, but not those within the parent domain.	*subdomain.example.com*	
Match all addresses within all subdomains, but not those within the parent domain.	*.example.com*	Note the period (.) before the domain name.
Match all addresses within a domain, and all addresses within all of its subdomains.	*example.com* *.example.com*	Create two separate recipients: one with the domain name, and one with a period followed by the domain name.
Match all recipients in all verified domains	[None]	Leave the recipient field blank.

> **Important**
>
> If multiple Amazon SES accounts receive email on a common domain (for example, if multiple teams in the same company each have separate Amazon SES accounts), Amazon SES processes all matching receipt rules simultaneously for each of those accounts. This behavior may result in a situation where one account generates a bounce, while another account accepts the email.
> We recommend that you coordinate with other teams in your organization that use Amazon SES to ensure that each account uses unique receipt rules, and that those rules do not overlap. In these situations, it is best to configure your receipt rules to use only email addresses or subdomains that are unique to your group or team.

b. Repeat this step for each recipient condition you want to add. When you finish adding recipient conditions, choose **Next**.

12. On the **Add actions** page, use the following procedure to add one or more actions to the receipt rule.

    a. Open the **Add new action** menu, and then choose one of the following types of actions:

    - **Add header** - This action adds a custom header to the received email.
    - **Return bounce response** - This action rejects the received email by returning a bounce response to the sender.
    - **Invoke Lambda function** - This action calls your code via an AWS Lambda function.
    - **Deliver to S3 bucket** - This action stores the received email in an Amazon Simple Storage Service (S3) bucket.
    - **Publish to Amazon SNS topic** - This action publishes the complete email to an Amazon Simple Notification Service (SNS) topic.
    - **Stop rule set** - This action terminates the evaluation of the receipt rule set.
    - **Integrate with Amazon WorkMail** - This action integrates with Amazon WorkMail.

    For more information about each of these actions, see Action options.

b. Repeat this step for each action that you want to define. If you have multiple actions defined, you can reorder them by using the up/down arrows within the action containers. Choose **Next** to proceed to the **Review** page.

13. On the **Review** page, review the settings and actions of the rule. If you need to make changes, choose the **Edit** option, or use the navigation section on the left side of the page to go directly to the step that contains the content you want to edit. You can optionally make changes to the order of the actions listed in the **Actions** table of the **Review** page by using the up/down arrows in the **Reorder** column.
14. When you're ready to proceed, choose **Create rule**.
15. On the confirmation page for the rule set, choose **Set as active** if you want to enforce the rule set immediately.

Rule modifications after creation

After you've created a rule set, you can edit both the rule set and the receipt rules it contains. Not only can they be edited, but there's also the option to duplicate either the rule set or its rules so that new ones can be created quickly. The following list shows the available modifications for the rule set and the receipt rules:

- **Rule set** is listed with its name, status and creation date. Modification options for the rule set are:
  - **Set as active/inactive** toggle button will toggle between setting the status.
  - **Duplicate** button will copy the rule set. You will be prompted to supply a unique name.
  - **Delete** button will delete the rule set. You will be prompted to confirm this irreversible action.
- **Receipt rules** are listed with their name, status, security, and order. Modification options for the receipt rules are:
  - **Up/down arrows** to reorder rule execution within the rule set.
  - **Duplicate** button will create a copy of the selected rule. You will be prompted to supply a unique name.
  - **Edit** button will open the selected rule so that any of its parameters such as rule settings, recipient conditions, and actions can be edited.
  - **Delete** button will delete the selected rule. You will be prompted to confirm this irreversible action.
  - **Create rule** button will allow you to create and add a new rule to the current rule set.

Action options

Each receipt rule for Amazon SES email receiving contains an ordered list of actions. This section describes the specific options for each action type.

The action types are the following:

- Add header action
- Return bounce response action
- Invoke Lambda function action
- Deliver to S3 bucket action
- Publish to Amazon SNS topic action
- Stop rule set action
- Integrate with Amazon WorkMail action

Add header action

The **Add Header** action adds a custom header to the received email. You typically use this action only in combination with another action. This action has the following options.

- **Header name—**The name of the header to add. It must be between 1 and 50 characters, inclusive, and consist of alphanumeric (a-z, A-Z, 0-9) characters and dashes only.
- **Header value—**The value of the header to add. It must be less than 2048 characters, and must not contain newline characters ("\r" or "\n").

Return bounce response action

The **Bounce** action rejects the email by returning a bounce response to the sender and, optionally, notifies you through Amazon SNS. This action has the following options.

- **SMTP Reply Code—**The SMTP reply code, as defined by RFC 5321.
- **SMTP Status Code—**The SMTP enhanced status code, as defined by RFC 3463.
- **Message—**Human-readable text to include in the bounce email.
- **Reply Sender—**The email address of the sender of the bounced email. This is the address from which the bounce email will be sent. It must be verified with Amazon SES.

> **Note**
>
> Bounce messages are not sent through your custom MAIL FROM domain, but are generated internally by SES and signed solely with the `amazonses.com` DKIM signature. As a workaround, use the **Reply Sender** option to set an email address for your bounce messages. Refer to this AWS re:Post article for more information.

- **SNS Topic—**The name or ARN of the Amazon SNS topic to optionally notify when a bounce email is sent. An example of an Amazon SNS topic ARN is *arn:aws-eusc:sns:eusc-de-east-1:123456789012:MyTopic*. You can also create an Amazon SNS topic when you set up your action by choosing **Create SNS Topic**. For more information about Amazon SNS topics, see the Amazon Simple Notification Service Developer Guide.

> **Note**
>
> The Amazon SNS topic you choose must be in the same AWS Region as the Amazon SES endpoint you use to receive email.

You can type in your own values for these fields, or you can choose a template that fills in the SMTP Reply Code, SMTP Status Code, and Message fields with values based on the bounce reason. The following templates are available:

- **Mailbox Does Not Exist—** SMTP Reply Code = 550, SMTP Status Code = 5.1.1
- **Message Too Large—** SMTP Reply Code = 552, SMTP Status Code = 5.3.4
- **Mailbox Full—** SMTP Reply Code = 552, SMTP Status Code = 5.2.2
- **Message Content Rejected—** SMTP Reply Code = 500, SMTP Status Code = 5.6.1
- **Unknown Failure—** SMTP Reply Code = 554, SMTP Status Code = 5.0.0
- **Temporary Failure—** SMTP Reply Code = 450, SMTP Status Code = 4.0.0

For additional bounce codes that you might use by typing custom values in the fields, see RFC 3463.

Invoke Lambda function action

The Lambda action calls your code through a Lambda function and, optionally, notifies you through Amazon SNS. This action has the following options and requirements.

Options

- **Lambda function—**The ARN of the Lambda function. An example of a Lambda function ARN is *arn:aws-eusc:lambda:eusc-de-east-1:account-id:function:MyFunction*.
- **Invocation type—**The invocation type of the Lambda function. An invocation type of **RequestResponse** means that the execution of the function results in an immediate response. An invocation type of **Event** means that the function is invoked asynchronously. We recommend that you use **Event** invocation type unless synchronous execution is required for your use case.

  There is a 30-second timeout on **RequestResponse** invocations.

  For more information, see Invoking Lambda functions in the *AWS Lambda Developer Guide*.
- **SNS topic—**The name or ARN of the Amazon SNS topic to notify when the specified Lambda function is triggered. An example of an Amazon SNS topic ARN is *arn:aws:sns:eusc-de-east-1:123456789012:MyTopic*. For more information, see Creating an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Requirements

- The Lambda function that you choose must be in the same AWS Region as the Amazon SES endpoint that you use to receive email.
- The Amazon SNS topic that you choose must be in the same AWS Region as the Amazon SES endpoint that you use to receive email.

Writing your Lambda function

To process your email, your Lambda function can be invoked asynchronously (that is, using the `Event` invocation type). The event object passed to your Lambda function will contain metadata pertaining to the inbound email event. You can also use the metadata to access the message content from your Amazon S3 bucket.

If you want to actually control the mail flow, your Lambda function must be invoked synchronously (that is, using the `RequestResponse` invocation type) and your Lambda function must call the

`callback` method with two arguments: the first argument is `null`, and the second argument is a `disposition` property that is set to either `STOP_RULE`, `STOP_RULE_SET`, or `CONTINUE`. If the second argument is `null` or does not have a valid `disposition` property, the mail flow continues and further actions and rules are processed, which is the same as with `CONTINUE`.

For example, you can stop the receipt rule set by writing the following line at the end of your Lambda function code:

```
callback( null, { "disposition" : "STOP_RULE_SET" });
```

For AWS Lambda code samples, see Lambda function examples. For examples of high-level use cases, see Use case examples.

Input format

Amazon SES passes information to the Lambda function in JSON format. The top-level object contains a `Records` array, which is populated with properties `eventSource`, `eventVersion`, and `ses`. The `ses` object contains `receipt` and `mail` objects, which are in exactly the same format as in the Amazon SNS notifications described in Notification contents.

The data that Amazon SES passes to Lambda includes metadata about the message, as well as several email headers. However, it doesn't contain the body of the message.

The following is a high-level view of the structure of the input that Amazon SES provides to the Lambda function.

```
{
   "Records": [
      {
        "eventSource": "aws:ses",
        "eventVersion": "1.0",
        "ses": {
           "receipt": {
               <same contents as SNS notification>
            },
           "mail": {
               <same contents as SNS notification>
           }
         }
      }
   ]
```

```
}
```

Return values

Your Lambda function can control mail flow by returning one of the following values:

- `STOP_RULE`—No further actions in the current receipt rule will be processed, but further receipt rules can be processed.
- `STOP_RULE_SET`—No further actions or receipt rules will be processed.
- `CONTINUE` or any other invalid value—This means that further actions and receipt rules can be processed.

The following topics cover samples of incoming mail events, examples of high-level use cases, and AWS Lambda code examples:

- Use case examples
- Lambda function examples

Use case examples

The following examples outline some rules that you might set up to use Lambda function outcomes to control your mail flow. For demonstration purposes, many of these examples use the S3 action as the outcome.

Use case 1: Drop spam across all domains

This example demonstrates a global rule that drops spam across all of your domains. Rules 2 and 3 are included to show that you can apply domain-specific rules after the spam is dropped over all the domains.

Rule 1

Recipient list: Empty. This rule will therefore apply to all recipients under all of your verified domains.

Actions

1. Lambda action (synchronous) that returns `STOP_RULE_SET` if the email is spam. Otherwise, it returns `CONTINUE`. See the example Lambda function for dropping spam in Lambda function examples.

Rule 2

Recipient list: example1.com

Actions

1. Any action.

Rule 3

Recipient list: example2.com

Actions

1. Any action.

Use case 2: Bounce spam across all domains

This example demonstrates a global rule that bounces spam across all of your domains. Rules 2 and 3 are included to show that you can apply domain-specific rules after the spam is bounced over all the domains.

Rule 1

Recipient list: Empty. This rule will therefore apply to all recipients under all of your verified domains.

Actions

1. Lambda action (synchronous) that returns `CONTINUE` if the email is spam. Otherwise, it returns `STOP_RULE`.
2. Bounce action ("500 5.6.1. Message content rejected").
3. Stop action.

Rule 2

Recipient list: example1.com

Actions

1. Any action

Rule 3

Recipient list: example2.com

Actions

1. Any action

Use case 3: Apply the most specific rule

This example demonstrates how you can use the Stop action to prevent emails from being processed by multiple rules. In this example, you have one rule for a specific address, and another rule for all email addresses under the domain. By using the Stop action, messages that match the rule for the specific email address are not processed by the more generic rule that applies to the domain.

Rule 1

Recipient list: user@example.com

Actions

1. Lambda action (asynchronous).
2. Stop action.

Rule 2

Recipient list: example.com

Actions

1. Any action.

Use case 4: Log mail events to CloudWatch

This example demonstrates how to keep an audit log of all mail going through your system before saving the mail to Amazon SES.

Rule 1

Recipient list: example.com

Actions

1. Lambda action (asynchronous) that writes the event object to a CloudWatch log. The example Lambda functions in Lambda function examples log to CloudWatch.
2. S3 action.

Use case 5: Drops mail that fails DKIM

This example demonstrates how you can save all incoming email to an Amazon S3 bucket, but only send email that goes to a specific email address, and passes DKIM, to your automated email application.

Rule 1

Recipient list: example.com

Actions

1. S3 action.
2. Lambda action (synchronous) that returns `STOP_RULE_SET` if the message fails DKIM. Otherwise, it returns `CONTINUE`.

Rule 2

Recipient list: support@example.com

Actions

1. Lambda action (asynchronous) that triggers the automated application.

Use case 6: Filters mail based on subject line

This example demonstrates how you can drop all of a domain's incoming mail that contains the word "discount" in the subject line, and then process mail intended for an automated system one way, and process mail addressed to all other recipients in the domain a different way.

Rule 1

Recipient list: example.com

Actions

1. Lambda action (synchronous) that returns `STOP_RULE_SET` if the subject line contains the word "discount". Otherwise, it returns `CONTINUE`.

Rule 2

Recipient list: support@example.com

Actions

1. S3 action with bucket 1.
2. Lambda action (asynchronous) that triggers the automated application.
3. Stop action.

Rule 3

Recipient list: example.com

Actions

1. S3 action with bucket 2.
2. Lambda action (asynchronous) that processes email for the rest of the domain.

Lambda function examples

This topic contains examples of Lambda functions that control mail flow.

Example 1: Drop spam

This example stops processing messages that have at least one spam indicator.

```
export const handler = async (event, context, callback) => {
    console.log('Spam filter');
```

```
    const sesNotification = event.Records[0].ses;
    console.log("SES Notification:\n", JSON.stringify(sesNotification, null, 2));

    // Check if any spam check failed
    if (sesNotification.receipt.spfVerdict.status === 'FAIL'
            || sesNotification.receipt.dkimVerdict.status === 'FAIL'
            || sesNotification.receipt.spamVerdict.status === 'FAIL'
            || sesNotification.receipt.virusVerdict.status === 'FAIL') {

        console.log('Dropping spam');

        // Stop processing rule set, dropping message
        callback(null, {'disposition':'STOP_RULE_SET'});
    } else {
        callback(null, {'disposition':'CONTINUE'});
    }
};
```

Example 2: Continue if a particular header is found

This example continues processing the current rule only if the email contains a specific header value.

```
export const handler = async (event, context, callback) => {
    console.log('Header matcher');

    const sesNotification = event.Records[0].ses;
    console.log("SES Notification:\n", JSON.stringify(sesNotification, null, 2));

    // Iterate over the headers
    for (let index in sesNotification.mail.headers) {
        const header = sesNotification.mail.headers[index];

        // Examine the header values
        if (header.name === 'X-Header' && header.value === 'X-Value') {
            console.log('Found header with value.');
            callback(null, {'disposition':'CONTINUE'});
            return;
        }
    }

    // Stop processing the rule if the header value wasn't found
    callback(null, {'disposition':'STOP_RULE'});
```

```
};
```

Example 3: Retrieve email from Amazon S3

This example gets the raw email from Amazon S3 and processes it.

> **Note**
>
> - You must first write the email to Amazon S3 using an S3 Action.
> - Ensure that the Lambda function has IAM permissions to fetch objects from the S3 bucket—refer to this AWS re:Post article for more information.
> - It's possible that the default Lambda execution timeouts are too short for your workflow, consider increasing them.

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";
const bucketName = '<Your Bucket Name>';

export const handler = async (event, context, callback) => {
    const client = new S3Client();
    console.log('Process email');

    var sesNotification = event.Records[0].ses;
    console.log("SES Notification:\n", JSON.stringify(sesNotification, null, 2));
    console.log("MessageId: " + sesNotification.mail.messageId)

    const getObjectCommand = new GetObjectCommand({
        Bucket: bucketName,
        Key: sesNotification.mail.messageId
    });

    try {
        const response = await client.send(getObjectCommand);
        const receivedMail = await response.Body.transformToString();
        console.log(receivedMail);
        callback(null, {'disposition':'CONTINUE'})
    } catch (e) {
        // Perform error handling here
        console.log("Encountered S3 client error: "+ e, e.stack);
        callback(null, {'disposition':'STOP_RULE_SET'})
    }
```

```
};
```

Example 4: Bounce messages that fail DMARC authentication

This examples sends a bounce message if an incoming email fails DMARC authentication.

> **Note**
>
> - When using this example, set the value of the `emailDomain` environment variable to your email receiving domain.
> - Ensure that the Lambda function has the `ses:SendBounce` permissions for the SES identity that is sending the bounce messages.

```
import { SESClient, SendBounceCommand } from "@aws-sdk/client-ses";
const sesClient = new SESClient();
// Assign the emailDomain environment variable to a constant.
const emailDomain = process.env.emailDomain;

export const handler = async (event, context, callback) => {
    console.log('Spam filter starting');

    const sesNotification = event.Records[0].ses;
    const messageId = sesNotification.mail.messageId;
    const receipt = sesNotification.receipt;

    console.log('Processing message:', messageId);

    // If DMARC verdict is FAIL and the sending domain's policy is REJECT
    // (p=reject), bounce the email.
    if (receipt.dmarcVerdict.status === 'FAIL'
        && receipt.dmarcPolicy.status === 'REJECT') {
        // The values that make up the body of the bounce message.
        const sendBounceParams = {
            BounceSender: `mailer-daemon@${emailDomain}`,
            OriginalMessageId: messageId,
            MessageDsn: {
                ReportingMta: `dns; ${emailDomain}`,
                ArrivalDate: new Date(),
                ExtensionFields: [],
            },
```

```
            // Include custom text explaining why the email was bounced.
            Explanation: "Unauthenticated email is not accepted due to the sending
 domain's DMARC policy.",
            BouncedRecipientInfoList: receipt.recipients.map((recipient) => ({
                Recipient: recipient,
                // Bounce with 550 5.6.1 Message content rejected
                BounceType: 'ContentRejected',
            })),
        };

        console.log('Bouncing message with parameters:');
        console.log(JSON.stringify(sendBounceParams, null, 2));

        const sendBounceCommand = new SendBounceCommand(sendBounceParams);

        // Try to send the bounce.
        try {
          const response = await sesClient.send(sendBounceCommand);
          console.log(response);
          console.log(`Bounce for message ${messageId} sent, bounce message ID:
 ${response.MessageId}`);
          // Stop processing additional receipt rules in the rule set.
          callback(null, {disposition: 'STOP_RULE_SET'});
        } catch (e) {
          // If something goes wrong, log the issue.
          console.log(`An error occurred while sending bounce for message:
 ${messageId}`, e);
          // Perform any additional error handling here
          callback(e)
        }

    // If the DMARC verdict is anything else (PASS, QUARANTINE or GRAY), accept
    // the message and process remaining receipt rules in the rule set.
    } else {
        console.log('Accepting message:', messageId);
        callback(null, {disposition: 'CONTINUE'});
    }
};
```

Deliver to S3 bucket action

The **Deliver to S3 bucket** action delivers the mail to an S3 bucket and can optionally notify you through SNS and more. This action has the following options.

- **S3 bucket** – The name of the S3 bucket to which to save received emails. You can also create a new S3 bucket when you set up your action by choosing **Create S3 Bucket**. Amazon SES provides you the raw, unmodified email, which is typically in Multipurpose Internet Mail Extensions (MIME) format. For more information about MIME format, see RFC 2045.

  > **Important**
  >
  > - The Amazon S3 bucket must exist in a region where SES the section called "Email receiving" is available; otherwise, you must use the IAM role option explained below.
  > - When you save your emails to an S3 bucket, the default maximum email size (including headers) is 40 MB.
  > - SES does not support receipt rules that upload to S3 buckets enabled with object lock configured with a default retention period.
  > - If applying encryption on your S3 bucket by specifying your own KMS key, be sure to use the fully qualified KMS key ARN, and not the KMS key alias; using the alias can result in data encrypted with a KMS key that belongs to the requester, and not the bucket administrator. See Using encryption for cross-account operations.

- **Object key prefix** – An optional key name prefix to use within the S3 bucket. Key name prefixes enable you to organize your S3 bucket in a folder structure. For example, if you use *Email* as your **Object key prefix**, your emails will appear in your S3 bucket in a folder named *Email*.
- **Message encryption** – The option to encrypt received email messages before delivering them to your S3 bucket.
- **KMS encryption key** – (Available if *Message encryption* is selected.) The AWS KMS key that SES should use to encrypt your emails before saving them to the S3 bucket. You can use the default KMS key or a customer managed key that you created in KMS.

  > **Note**
  >
  > The KMS key you choose must be in the same AWS region as the SES endpoint you use to receive email.

  - To use the default KMS key, choose **aws/ses** when you set up the receipt rule in the SES console. If you use the SES API, you can specify the default KMS key by providing an ARN in the form of `arn:aws:kms:REGION:AWSACCOUNTID:alias/aws/ses`. For example, if your

AWS account ID is 123456789012 and you want to use the default KMS key in the eusc-de-east-1 region, the ARN of the default KMS key would be `arn:aws-eusc:kms:eusc-de-east-1:123456789012:alias/aws/ses`. If you use the default KMS key, you don't need to perform any extra steps to give SES permission to use the key.

- To use a customer managed key that you created in KMS, provide the ARN of the KMS key and ensure that you add a statement to your key's policy to give SES permission to use it. For more information about giving permissions, see Giving permissions to Amazon SES for email receiving.

For more information about using KMS with SES, see the AWS Key Management Service Developer Guide. If you do not specify a KMS key in the console or API, SES will not encrypt your emails.

> **Important**
>
> Your mail is encrypted by SES using the S3 encryption client before the mail is submitted to S3 for storage. It is not encrypted using S3 server-side encryption. This means that you must use the S3 encryption client to decrypt the email after retrieving it from S3, as the service has no access to use your KMS keys for decryption. This encryption client is available in the AWS SDK for Java and the AWS SDK for Ruby. For more information, see the Amazon Simple Storage Service User Guide.

- **IAM role** – An IAM role used by SES to access the resources in the *Deliver to S3* action (Amazon S3 bucket, SNS topic, and KMS key). If not provided, you'll need to explicitly give permissions to SES to access each resource individually—see Giving permissions to Amazon SES for email receiving.

  If you want to write to an S3 bucket that exists in a region where SES *Email receiving* isn't available, you must use an IAM role that has the write to S3 permission policy as an inline policy of the role. You can apply the permission policy for this action directly from the console:

  1. Choose **Create new role** in the **IAM role** field and enter a name followed by **Create role**. (The IAM trust policy for this role will automatically be generated in the background.)
  2. Because the IAM trust policy was automatically generated, you'll only need to add the action's permission policy to the role—select **View role** under the **IAM role** field to open the IAM console.
  3. Under the **Permissions** tab, choose **Add permissions** and select **Create inline policy**.

4. On the **Specify permissions** page, select **JSON** in the **Policy editor**.
5. Copy and paste the permission policy from IAM role permissions for S3 action into the **Policy editor** and replace the data in red text with your own. (Be sure to delete any example code in the editor.)
6. Choose **Next**.
7. Review and create your permission policy for the IAM role by choosing **Create policy**.
8. Select your browser's tab where you have the SES **Create rule–Add actions** page open and continue with the remaining steps for creating rules.

- **SNS topic** – The name or ARN of the Amazon SNS topic to notify when an email is saved to the S3 bucket. An example of an SNS topic ARN is *arn:aws-eusc:sns:eusc-de-east-1:123456789012:MyTopic*. You can also create an SNS topic when you set up your action by choosing **Create SNS Topic**. For more information about SNS topics, see the Amazon Simple Notification Service Developer Guide.

> **Note**
>
> - The SNS topic you choose must be in the same AWS region as the SES endpoint you use to receive email.
> - Only use *customer managed* KMS key encryption with SNS topics you associate with SES receipt rules as you will be required to edit the KMS key policy to allow SES to publish to SNS. This is in contrast with *AWS managed* KMS key policies which cannot be edited by design.

Publish to Amazon SNS topic action

The **SNS** action publishes the mail using an Amazon SNS notification. The notification includes the complete email content. This action has the following options.

- **SNS Topic—**The name or ARN of the Amazon SNS topic to which to publish the emails. The Amazon SNS notifications will contain a raw, unmodified copy of the email, which is typically in Multipurpose Internet Mail Extensions (MIME) format. For more information about MIME format, see RFC 2045.

> **Important**
>
> If you choose to receive your emails through Amazon SNS notifications, the maximum email size (including headers) is 150 KB. Larger emails will bounce. If you anticipate emails larger than this size, save the emails to an Amazon S3 bucket instead.

An example of an Amazon SNS topic ARN is *arn:aws-eusc:sns:eusc-de-east-1:123456789012:MyTopic*. You can also create an Amazon SNS topic when you set up your action by choosing **Create SNS Topic**. For more information about Amazon SNS topics, see the Amazon Simple Notification Service Developer Guide.

> **Note**
>
> - The Amazon SNS topic you choose must be in the same AWS region as the Amazon SES endpoint you use to receive email.
> - Only use *customer managed* KMS key encryption with SNS topics you associate with SES receipt rules as you will be required to edit the KMS key policy to allow SES to publish to SNS. This is in contrast with *AWS managed* KMS key policies which cannot be edited by design.

- **Encoding—**The encoding to use for the email within the Amazon SNS notification. UTF-8 is easier to use, but may not preserve all special characters when a message was encoded with a different encoding format. Base64 preserves all special characters. For information about UTF-8 and Base64, see RFC 3629 and RFC 4648, respectively.

When you receive an email, Amazon SES executes the rules in the active receipt rule set. You can configure receipt rules to send you notifications using Amazon SNS. Your receipt rules can send two different types of notifications:

- **Notifications sent from SNS actions** – When you add an SNS action to a receipt rule, it sends information about the email as well as the email's content. If the message is 150KB or smaller, this notification type also includes the complete MIME body of the email.
- **Notifications sent from other action types** – When you add any other action type (including Bounce, Lambda, Stop Rule Set, or WorkMail actions) to a receipt rule, you can optionally specify an Amazon SNS topic. If you do, you will receive notifications when these actions are performed.

These notifications contain information about the email, but do not contain the content of the email.

The following topics describe the contents of these notifications and provide an example of each type of notification:

- Contents of notifications for Amazon SES email receiving
- Examples of notifications for Amazon SES email receiving

Contents of notifications for Amazon SES email receiving

All notifications for email receiving are published to Amazon Simple Notification Service (Amazon SNS) topics in JavaScript Object Notation (JSON) format.

For example notifications, see Notification examples.

Contents

- Top-level JSON object
- receipt object
  - action object
  - dkimVerdict object
  - dmarcVerdict object
  - spamVerdict object
  - spfVerdict object
  - virusVerdict object
- mail object
  - commonHeaders object

Top-level JSON object

The top-level JSON object contains the following fields.

Field Name	Description
`notificationType`	The notification type. For this type of notification, the value is always `Received`.

Field Name	Description
receipt	Object that contains information about the email delivery.
mail	Object that contains information about the email associated with the notification.
content	String that contains the raw, unmodified email, which is typically in Multipurpose Internet Mail Extensions (MIME) format. For more information about MIME format, see RFC 2045. **Note** This field is present only if the notification was triggered by an SNS action. Notifications triggered by all other actions do not contain this field.

receipt object

The `receipt` object has the following fields.

Field Name	Description
action	Object that encapsulates information about the action that was executed. For a list of possible values, see action object.
dkimVerdict	Object that indicates whether the DomainKeys Identified Mail (DKIM) check passed. For a list of possible values, see dkimVerdict object.
dmarcPolicy	Indicates the Domain-based Message Authentication, Reporting & Conformance

Field Name	Description
	(DMARC) settings for the sending domain. This field only appears if the message fails DMARC authentication. Possible values for this field are: • none: The owner of the sending domain requests that no specific action be taken on messages that fail DMARC authentication. • `quarantine` : The owner of the sending domain requests that messages that fail DMARC authentication be treated by receivers as suspicious. • `reject`: The owner of the sending domain requests that messages that fail DMARC authentication be rejected.
`dmarcVerdict`	Object that indicates whether the Domain-based Message Authentication, Reporting & Conformance (DMARC) check passed. For a list of possible values, see dmarcVerdict object.
`processingTimeMillis`	String that specifies the period, in milliseconds, from the time Amazon SES received the message to the time it triggered the action.
`recipients`	The recipients (specifically, the envelope RCPT TO addresses) that were matched by the active receipt rule. The addresses listed here may differ from those listed by the `destination` field in the the section called “mail object”.
`spamVerdict`	Object that indicates whether the message is spam. For a list of possible values, see spamVerdict object.

Field Name	Description
spfVerdict	Object that indicates whether the Sender Policy Framework (SPF) check passed. For a list of possible values, see spfVerdict object.
timestamp	String that specifies the qualified date and time at which the action was triggered, in ISO 8601 format.
virusVerdict	Object that indicates whether the message contains a virus. For a list of possible values, see virusVerdict object.

action object

The action object has the following fields.

Field Name	Description
type	String that indicates the type of action that was executed. Possible values are S3, SNS, Bounce, Lambda, Stop, and WorkMail.
topicArn	String that contains the Amazon Resource Name (ARN) of the Amazon SNS topic to which the notification was published.
bucketName	String that contains the name of the Amazon S3 bucket to which the message was published. Present only for the S3 action type.
objectKey	String that contains a name that uniquely identifies the email in the Amazon S3 bucket. This is the same as the messageId in the the section called “mail object”. Present only for the S3 action type.

Field Name	Description
`smtpReplyCode`	String that contains the SMTP reply code, as defined by RFC 5321. Present only for the bounce action type.
`statusCode`	String that contains the SMTP enhanced status code, as defined by RFC 3463. Present only for the bounce action type.
`message`	String that contains the human-readable text to include in the bounce message. Present only for the bounce action type.
`sender`	String that contains the email address of the sender of the email that bounced. This is the address from which the bounce message was sent. Present only for the bounce action type.
`functionArn`	String that contains the ARN of the Lambda function that was triggered. Present only for the Lambda action type.
`invocationType`	String that contains the invocation type of the Lambda function. Possible values are `RequestResponse` and `Event`. Present only for the Lambda action type.
`organizationArn`	String that contains the ARN of the Amazon WorkMail organization. Present only for the WorkMail action type.

dkimVerdict object

The `dkimVerdict` object has the following fields.

Field Name	Description
`status`	String that contains the DKIM verdict. Possible values are: • `PASS`: The message passed DKIM authentication. • `FAIL`: The message failed DKIM authentication. • `GRAY`: The message is not DKIM-signed or the from domain and DKIM-signature domain do not match. • `PROCESSING_FAILED` : There is an issue that prevents Amazon SES from checking the DKIM signature. For example, DNS queries are failing or the DKIM signature header is not formatted properly.

dmarcVerdict object

The `dmarcVerdict` object has the following fields.

Field Name	Description
`status`	String that contains the DMARC verdict. Possible values are: • `PASS`: The message passed DMARC authentication. • `FAIL`: The message failed DMARC authentication. • `GRAY`: At least one of SPF or DKIM passed authentication, but the sending domain does not have a DMARC policy or uses the p=none policy.

Field Name	Description
	• PROCESSING_FAILED : There is an issue that prevents Amazon SES from providing a DMARC verdict.

spamVerdict object

The spamVerdict object has the following fields.

Field Name	Description
status	String that contains the result of spam scanning. Possible values are: • PASS: The spam scan determined that the message is unlikely to contain spam. • FAIL: The spam scan determined that the message is likely to contain spam. • GRAY: Amazon SES scanned the email but could not determine with confidence whether it is spam. • PROCESSING_FAILED : Amazon SES was unable to scan the email. For example, the email is not a valid MIME message.

spfVerdict object

The spfVerdict object has the following fields.

Field Name	Description
status	String that contains the SPF verdict. Possible values are: • PASS: The message passed SPF authentic ation.

Field Name	Description
	• FAIL: The message failed SPF authentic ation. • GRAY: The SPF result is none, softfail, or neutral. • PROCESSING_FAILED : There is an issue that prevents Amazon SES from checking the SPF record. For example, DNS queries are failing.

virusVerdict object

The virusVerdict object has the following fields.

Field Name	Description
status	String that contains the result of virus scanning. Possible values are: • PASS: The message does not contain a virus. • FAIL: The message contains a virus. • GRAY: Amazon SES scanned the email but could not determine with confidence whether it contains a virus. • PROCESSING_FAILED : Amazon SES is unable to scan the content of the email. For example, the email is not a valid MIME message.

mail object

The mail object has the following fields.

Field Name	Description
destination	A complete list of all recipient addresses (including To: and CC: recipients) from the MIME headers of the incoming email.
messageId	String that contains the unique ID assigned to the email by Amazon SES. If the email was delivered to Amazon S3, the message ID is also the Amazon S3 object key that was used to write the message to your Amazon S3 bucket.
source	String that contains the email address (specifically, the envelope MAIL FROM address) that the email was sent from.
timestamp	String that contains the time at which the email was received, in ISO8601 format.
headers	The Amazon SES headers and your custom headers. Each header has the following fields: name and value.
commonHeaders	The headers common to all emails. Each header has the following fields: name and value.
headersTruncated	Specifies whether the headers were truncated in the notification, which happens if the headers are larger than 10 KB. Possible values are true and false.

commonHeaders object

The commonHeaders object can have the fields shown in the following table. The fields present in this object vary depending on which fields were present in the incoming email.

Field Name	Description
`messageId`	The ID of the original message.
`date`	The date and time when Amazon SES received the message.
`to`	The To header of the email.
`cc`	The CC header of the email.
`bcc`	The BCC header of the email.
`from`	The From header of the email.
`sender`	The Sender header of the email.
`returnPath`	The Return-Path header of the email.
`replyTo`	The Reply-To header of the email.
`subject`	The Subject header of the email.

Examples of notifications for Amazon SES email receiving

This section includes examples of the following types of notifications:

- A notification sent as a result of an SNS action.
- A notification sent as a result of another type of action (an *alert notification*).

Notification of an SNS action

This section contains an example of an SNS action notification. Unlike the alert notification shown previously, it includes a `content` section that contains the email, which is typically in Multipurpose Internet Mail Extensions (MIME) format.

```
{
  "notificationType":"Received",
  "receipt":{
    "timestamp":"2015-09-11T20:32:33.936Z",
```

```
    "processingTimeMillis":222,
    "recipients":[
      "recipient@example.com"
    ],
    "spamVerdict":{
      "status":"PASS"
    },
    "virusVerdict":{
      "status":"PASS"
    },
    "spfVerdict":{
      "status":"PASS"
    },
    "dkimVerdict":{
      "status":"PASS"
    },
    "action":{
      "type":"SNS",
      "topicArn":"arn:aws-eusc:sns:us-east-1:012345678912:example-topic"
    }
  },
  "mail":{
    "timestamp":"2015-09-11T20:32:33.936Z",
    "source":"61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com",
    "messageId":"d6iitobk75ur44p8kdnnp7g2n800",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "headers":[
      {
        "name":"Return-Path",

"value":"<0000014fbe1c09cf-7cb9f704-7531-4e53-89a1-5fa9744f5eb6-000000@amazonses.com>"
      },
      {
        "name":"Received",
        "value":"from a9-183.smtp-out.amazonses.com (a9-183.smtp-out.amazonses.com
[54.240.9.183]) by inbound-smtp.us-east-1.amazonaws.com with SMTP id
d6iitobk75ur44p8kdnnp7g2n800 for recipient@example.com; Fri, 11 Sep 2015 20:32:33
+0000 (UTC)"
      },
      {
        "name":"DKIM-Signature",
```

```
        "value":"v=1; a=rsa-sha256; q=dns/txt; c=relaxed/simple;
 s=ug7nbtf4gccmlpwj322ax3p6ow6yfsug; d=amazonses.com; t=1442003552;
 h=From:To:Subject:MIME-Version:Content-Type:Content-Transfer-Encoding:Date:Message-
ID:Feedback-ID; bh=DWr3IOmYWoXCA9ARqGC/UaODfghffiwFNRIb2Mckyt4=;
 b=p4ukUDSFqhqiub+zPR0DW1kp7oJZakrzupr6LBe6sUuvqpBkig56UzUwc29rFbJF
 hlX3Ov7DeYVNoN38stqwsF8ivcajXpQsXRC1cW9z8x875J041rClAjV7EGbLmudVpPX
 4hHst1XPyX5wmgdHIhmUuh8oZKpVqGi6bHGzzf7g="
      },
      {
        "name":"From",
        "value":"sender@example.com"
      },
      {
        "name":"To",
        "value":"recipient@example.com"
      },
      {
        "name":"Subject",
        "value":"Example subject"
      },
      {
        "name":"MIME-Version",
        "value":"1.0"
      },
      {
        "name":"Content-Type",
        "value":"text/plain; charset=UTF-8"
      },
      {
        "name":"Content-Transfer-Encoding",
        "value":"7bit"
      },
      {
        "name":"Date",
        "value":"Fri, 11 Sep 2015 20:32:32 +0000"
      },
      {
        "name":"Message-ID",
        "value":"<61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>"
      },
      {
        "name":"X-SES-Outgoing",
        "value":"2015.09.11-54.240.9.183"
      },
```

```
      {
        "name":"Feedback-ID",
        "value":"1.us-east-1.Krv2FKpFdWV+KUYw3Qd6wcpPJ4Sv/pOPpEPSHn2u2o4=:AmazonSES"
      }
    ],
    "commonHeaders":{

 "returnPath":"0000014fbe1c09cf-7cb9f704-7531-4e53-89a1-5fa9744f5eb6-000000@amazonses.com",
      "from":[
        "sender@example.com"
      ],
      "date":"Fri, 11 Sep 2015 20:32:32 +0000",
      "to":[
        "recipient@example.com"
      ],
      "messageId":"<61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>",
      "subject":"Example subject"
    }
  },
  "content":"Return-Path: <61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>\r
\nReceived: from a9-183.smtp-out.amazonses.com (a9-183.smtp-out.amazonses.com
 [54.240.9.183])\r\n by inbound-smtp.us-east-1.amazonaws.com with SMTP id
 d6iitobk75ur44p8kdnnp7g2n800\r\n for recipient@example.com;\r\n Fri, 11 Sep 2015
 20:32:33 +0000 (UTC)\r\nDKIM-Signature: v=1; a=rsa-sha256; q=dns/txt; c=relaxed/
simple;\r\n\ts=ug7nbtf4gccmlpwj322ax3p6ow6yfsug; d=amazonses.com; t=1442003552;\r\n
\th=From:To:Subject:MIME-Version:Content-Type:Content-Transfer-Encoding:Date:Message-
ID:Feedback-ID;\r\n\tbh=DWr3IOmYWoXCA9ARqGC/UaODfghffiwFNRIb2Mckyt4=;\r\n
\tb=p4ukUDSFqhqiub+zPR0DW1kp7oJZakrzupr6LBe6sUuvqpBkig56UzUwc29rFbJF\r\n
\thlX3Ov7DeYVNoN38stqwsF8ivcajXpQsXRC1cW9z8x875J041rClAjV7EGbLmudVpPX\r\n
\t4hHst1XPyX5wmgdHIhmUuh8oZKpVqGi6bHGzzf7g=\r\nFrom: sender@example.com\r\nTo:
 recipient@example.com\r\nSubject: Example subject\r\nMIME-Version: 1.0\r\nContent-
Type: text/plain; charset=UTF-8\r\nContent-Transfer-Encoding: 7bit\r\nDate: Fri, 11 Sep
 2015 20:32:32 +0000\r\nMessage-ID: <61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>
\r\nX-SES-Outgoing: 2015.09.11-54.240.9.183\r\nFeedback-ID: 1.us-east-1.Krv2FKpFdWV
+KUYw3Qd6wcpPJ4Sv/pOPpEPSHn2u2o4=:AmazonSES\r\n\r\nExample content\r\n"
}
```

Alert notification

This section contains an example of an Amazon SNS notification that can be triggered by an S3 action. Notifications triggered by Lambda actions, bounce actions, stop actions, and WorkMail actions are similar. Although the notification contains information about the email, it does not contain the content of the email itself.

```
{
     "notificationType": "Received",
    "receipt": {
      "timestamp": "2015-09-11T20:32:33.936Z",
      "processingTimeMillis": 406,
      "recipients": [
        "recipient@example.com"
      ],
      "spamVerdict": {
        "status": "PASS"
      },
      "virusVerdict": {
        "status": "PASS"
      },
      "spfVerdict": {
        "status": "PASS"
      },
      "dkimVerdict": {
        "status": "PASS"
      },
      "action": {
        "type": "S3",
        "topicArn": "arn:aws-eusc:sns:us-east-1:012345678912:example-topic",
        "bucketName": "my-S3-bucket",
        "objectKey": "\email"
      }
    },
    "mail": {
      "timestamp": "2015-09-11T20:32:33.936Z",
      "source":
 "0000014fbe1c09cf-7cb9f704-7531-4e53-89a1-5fa9744f5eb6-000000@amazonses.com",
      "messageId": "d6iitobk75ur44p8kdnnp7g2n800",
      "destination": [
        "recipient@example.com"
      ],
      "headersTruncated": false,
      "headers": [
        {
          "name": "Return-Path",
          "value":
 "<0000014fbe1c09cf-7cb9f704-7531-4e53-89a1-5fa9744f5eb6-000000@amazonses.com>"
        },
        {
```

```
          "name": "Received",
          "value": "from a9-183.smtp-out.amazonses.com (a9-183.smtp-out.amazonses.com
 [54.240.9.183]) by inbound-smtp.us-east-1.amazonaws.com with SMTP id
 d6iitobk75ur44p8kdnnp7g2n800 for recipient@example.com; Fri, 11 Sep 2015 20:32:33
 +0000 (UTC)"
        },
        {
          "name": "DKIM-Signature",
          "value": "v=1; a=rsa-sha256; q=dns/txt; c=relaxed/simple;
 s=ug7nbtf4gccmlpwj322ax3p6ow6yfsug; d=amazonses.com; t=1442003552;
 h=From:To:Subject:MIME-Version:Content-Type:Content-Transfer-Encoding:Date:Message-
ID:Feedback-ID; bh=DWr3IOmYWoXCA9ARqGC/UaODfghffiwFNRIb2Mckyt4=;
 b=p4ukUDSFqhqiub+zPR0DW1kp7oJZakrzupr6LBe6sUuvqpBkig56UzUwc29rFbJF
 hlX3Ov7DeYVNoN38stqwsF8ivcajXpQsXRC1cW9z8x875J041rClAjV7EGbLmudVpPX
 4hHst1XPyX5wmgdHIhmUuh8oZKpVqGi6bHGzzf7g="
        },
        {
          "name": "From",
          "value": "sender@example.com"
        },
        {
          "name": "To",
          "value": "recipient@example.com"
        },
        {
          "name": "Subject",
          "value": "Example subject"
        },
        {
          "name": "MIME-Version",
          "value": "1.0"
        },
        {
          "name": "Content-Type",
          "value": "text/plain; charset=UTF-8"
        },
        {
          "name": "Content-Transfer-Encoding",
          "value": "7bit"
        },
        {
          "name": "Date",
          "value": "Fri, 11 Sep 2015 20:32:32 +0000"
        },
```

```
        {
          "name": "Message-ID",
          "value": "<61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>"
        },
        {
          "name": "X-SES-Outgoing",
          "value": "2015.09.11-54.240.9.183"
        },
        {
          "name": "Feedback-ID",
          "value": "1.us-east-1.Krv2FKpFdWV+KUYw3Qd6wcpPJ4Sv/pOPpEPSHn2u2o4=:AmazonSES"
        }
      ],
      "commonHeaders": {
        "returnPath":
"0000014fbe1c09cf-7cb9f704-7531-4e53-89a1-5fa9744f5eb6-000000@amazonses.com",
        "from": [
          "sender@example.com"
        ],
        "date": "Fri, 11 Sep 2015 20:32:32 +0000",
        "to": [
          "recipient@example.com"
        ],
        "messageId": "<61967230-7A45-4A9D-BEC9-87CBCF2211C9@example.com>",
        "subject": "Example subject"
      }
    }
  }
```

Stop rule set action

The **Stop** action terminates the evaluation of the receipt rule set and, optionally, notifies you through Amazon SNS. This action has the following options.

- **SNS Topic—**The name or ARN of the Amazon SNS topic to notify when the Stop action is performed. An example of an Amazon SNS topic ARN is *arn:aws-eusc:sns:eusc-de-east-1:123456789012:MyTopic*. You can also create an Amazon SNS topic when you set up your action by choosing **Create SNS Topic**. For more information about Amazon SNS topics, see the Amazon Simple Notification Service Developer Guide.

> **Note**
>
> The Amazon SNS topic you choose must be in the same AWS Region as the Amazon SES endpoint you use to receive email.

Integrate with Amazon WorkMail action

The **WorkMail** action integrates with Amazon WorkMail. If Amazon WorkMail performs all of your email processing, you will typically not use this action directly because Amazon WorkMail takes care of the setup. This action has the following options.

- **Organization ARN—**The ARN of the Amazon WorkMail organization. Amazon WorkMail organization ARNs are in the form `arn:aws:workmail:`*`region`*`:`*`account_ID`*`:organization/`*`organization_ID`*, where:
  - `region` is the region in which you are using Amazon SES and Amazon WorkMail. (You must use them from the same Region.) An example is eusc-de-east-1.
  - `account_ID` is the AWS account ID. You can find your AWS account ID on the Account page of the AWS Management Console.
  - `organization_ID` is a unique identifier that Amazon WorkMail generates when you create an organization. You can find the organization ID in the Amazon WorkMail console on the Organization Settings page of your organization.

  An example of a complete Amazon WorkMail organization ARN is *arn:aws-eusc:workmail:eusc-de-east-1:123456789012:organization/m-68755160c4cb4e29a2b2f8fb58f359d7*. For information about Amazon WorkMail organizations, see the Amazon WorkMail Administrator Guide.
- **SNS Topic—**The name or ARN of the Amazon SNS topic to notify when the Amazon WorkMail action is taken. An example of an Amazon SNS topic ARN is *arn:aws-eusc:sns:eusc-de-east-1:123456789012:MyTopic*. You can also create an Amazon SNS topic when you set up your action by choosing **Create SNS Topic**. For more information about Amazon SNS topics, see the Amazon Simple Notification Service Developer Guide.

> **Note**
>
> The Amazon SNS topic you choose must be in the same AWS Region as the Amazon SES endpoint you use to receive email.

> **Note**
>
> Amazon SES only supports WorkMail actions in regions where WorkMail is available. See Amazon WorkMail endpoints and quotas in the AWS General Reference.

Create IP address filters console walkthrough

This section will walk you through setting up IP address filters using the Amazon SES console. IP address filtering allows you to provide a broad level of control. These IP filters allow you to explicitly block or allow all messages from specific IP addresses or IP address ranges.

Optionally, you can use the `CreateReceiptFilter` API to create an IP address filter as described in the Amazon Simple Email Service API Reference.

> **Note**
>
> If you only want to receive mail from a finite list of known IP addresses, then set up a block list that contains `0.0.0.0/0`, and set up an allow list that contains the IP addresses that you trust. This configuration blocks all IP addresses by default, and only allows mail from the IP addresses that you explicitly specify.

Prerequisites

The following prerequisites must be met before proceeding with setting up recipient based email control using IP address filters:

1. You first need to create and verify a domain identity in Amazon SES.
2. Next, you need to specify which mail servers can accept mail for your domain by publishing an MX record to your domain's DNS settings. (The MX record should refer to the Amazon SES endpoint that receives mail for the AWS Region where you use Amazon SES.)

Create IP address filters

To create IP address filters using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Email receiving**.
3. Select the **IP address filters** tab.
4. Choose **Create Filter**.
5. Enter an unique name for your filter - the field's legend will indicate syntax requirements. (The name must contain less than 64 alphanumeric, hyphen (-), underscore (_), and period (.) characters. The name must start and end with a letter or number.)
6. Enter an IP address or a range of IP addresses - the field's legend will give examples specified in Classless Inter-Domain Routing (CIDR) syntax. (An example of a single IP address is 10.0.0.1. An example of a range of IP addresses is 10.0.0.1/24. For more information about CIDR notation, see RFC 2317.)
7. Choose the **Policy type** by selecting either the **Block** or **Allow** radio button.
8. Choose **Create filter**.
9. If you want to add another IP filter, choose **Create filter** and repeat the previous steps for each additional filter you wish to add.
10. If you want to remove an IP address filter, select it and choose the **Delete** button.

Viewing metrics for Amazon SES email receiving

If you've enabled email receiving in Amazon SES and you've created receipt rules for your email, you can view the metrics for those receipt rule sets and rules using Amazon CloudWatch.

In the CloudWatch console, you'll find the metrics under **Metrics** > **All metrics** > **SES** > **Receipt Rule Set Metrics** and **Receipt Rule Metrics**.

> **Note**
>
> **Receipt Rule Set Metrics** and **Receipt Rule Metrics** will not appear under **SES** if you have not yet:
>
> - enabled email receiving

- created any receipt rules
- received any mail that would match any of your rules.

The following message metrics are available:

- **Message receiving**

Scope	Metric	Description	Dimension
Receipt Rule Se	Received	SES successfully received a message that has at least one rule that applies. This metric can only have a value of 1.	RuleSetName
Receipt Rule M	Received	SES successfully received a message and will try to process the applied rule. This metric can only have a value of 1.	RuleName

- **Message publishing**

Scope	Metric	Description	Dimension
Receipt Rule Se	PublishSuccess	SES successfully executed all rules that apply within a rule set.	RuleSetName
Receipt Rule M	PublishSuccess	SES successfully executed a rule that applies to the receiving message.	RuleName
Receipt Rule Se	PublishFailure	SES encountered an error when it tried to execute rules within a rule set, execution will be retried.	RuleSetName
Receipt Rule M	PublishFailure	SES encountered an error when it tried to execute the actions in a rule—depending on the error, execution may be retried.	RuleName

Scope	Metric	Description	Dimension
Receipt Rule Se	PublishExpired	SES will no longer retry to execute the rules because they didn't succeed within 36 hours, or encountered non-retriable error.	RuleSetName
Receipt Rule M	PublishExpired	SES will no longer retry to execute the rule's actions because they didn't succeed within 36 hours.	RuleName

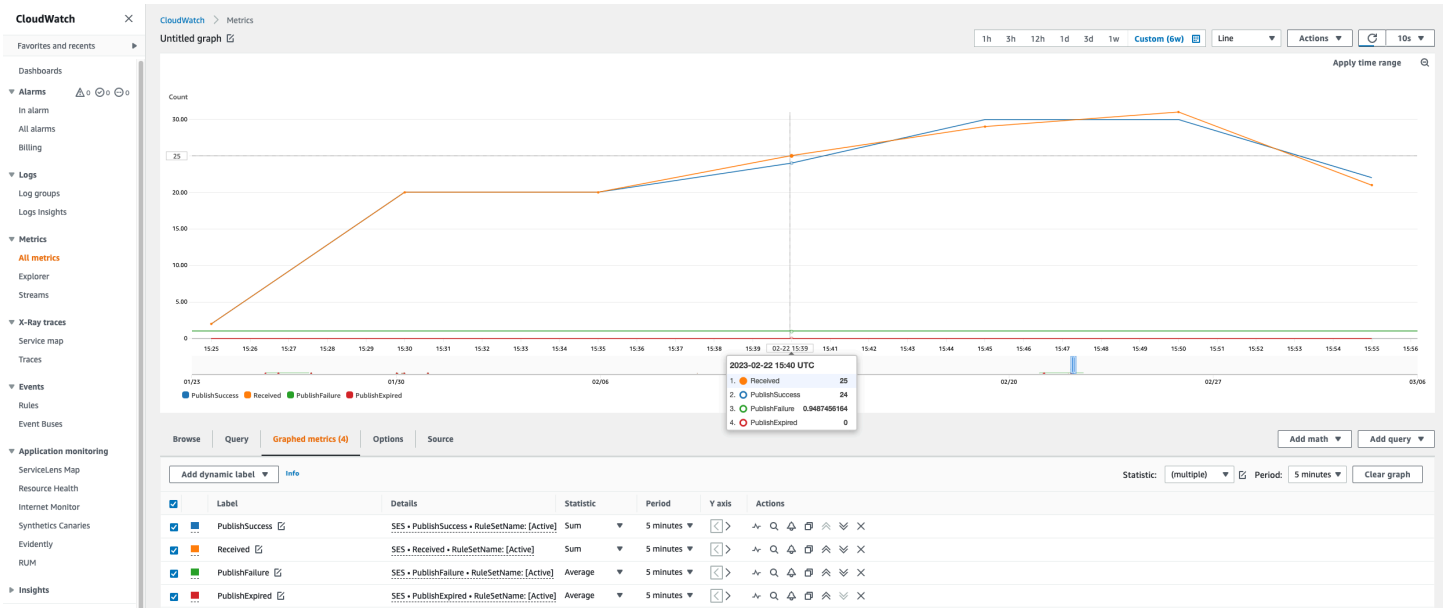
Note

- In the preceding tables, the term *applies* means that the sender is not blocklisted by IP Filters or is on SES's internal blocklist, and the rule has matching recipient conditions and matching TLS policy.
- Publish failure errors can occur, for example, if you deleted or revoked permissions to an Amazon S3 bucket, Amazon SNS topic, or Lambda function that an action in one of your receipt rules was configured to use.
- Because only one rule set can be active at a time, SES publishes an aggregate metric displayed as *RuleSetName:[Active]* for all rules sets that were active for the time range you select in CloudWatch. This has the advantage of letting you freely change rule sets without any change to your alarming setup.

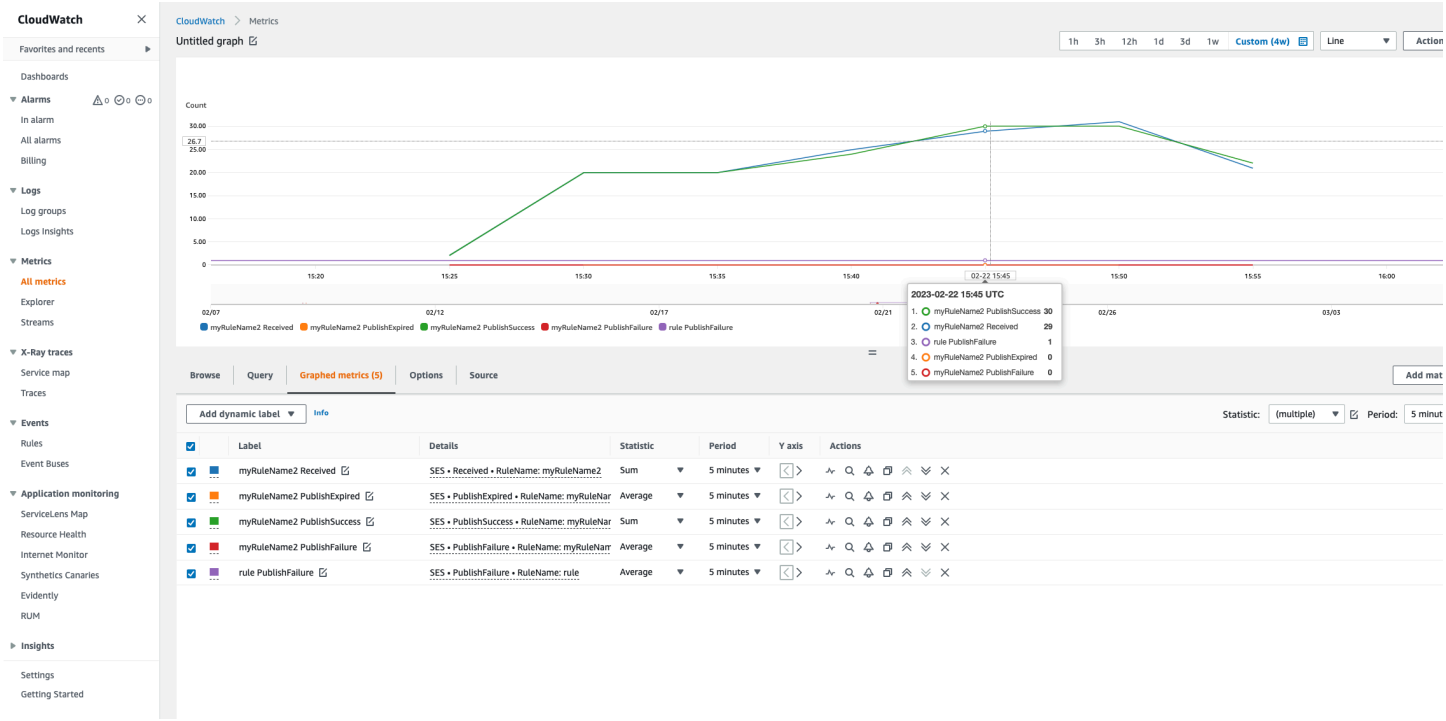
Important

Changes you make to fix your receipt rule set will apply only to emails that Amazon SES receives after the update. Emails are always evaluated against the receipt rule set that was in place at the time the email was received.

Metrics for an SES *receipt rule set* displayed in the CloudWatch console.



Metrics for an SES *receipt rule* displayed in the CloudWatch console.



Tenants

This chapter explains how to use Amazon SES tenant management to isolate, monitor, and manage email sending across multiple tenants within your SES account. This feature helps Independent Software Vendors (ISVs), enterprises, and organizations sending emails on behalf of multiple downstream entities maintain separate reputation profiles and prevent issues with one tenant from affecting others.

What is tenant management?

Tenant management is a feature that allows you to create isolated containers called "tenants" within your SES account. Each tenant can have its own email identities, configuration sets, templates, and reputation metrics, ensuring that email activities are completely separated between different customers or business units.

Tenant management addresses the challenge where one customer's poor email practices could previously pause an entire SES account, affecting all other customers. With tenant isolation, you can manage multiple email streams independently while maintaining centralized oversight and control.

A tenant serves as a logical container that groups related SES resources together. When you send emails on behalf of a specific tenant, SES tracks reputation metrics, and enforces policies at the tenant level. This isolation ensures that a high bounce rate or complaint rate from one tenant doesn't impact the deliverability of emails sent by other tenants.

Tenant management is particularly valuable for:

- **Independent Software Vendors (ISVs)** sending emails on behalf of multiple customers.
- **Enterprises** managing email communications across different business units.
- **Service providers** who need to isolate email reputation by client or application.
- **Organizations** requiring compliance with different regulatory requirements per tenant.

How tenant management works

Tenant reputation management works by creating a logical container for your email sending resources. You assign specific resources to each tenant, including verified identities (domains and

email addresses), configuration sets, and templates. When sending email on behalf of a tenant, you specify the tenant in your API call or SMTP header, and SES verifies that the resources being used are properly associated with that tenant.

Resource association

Resources in your SES account can be associated with tenants in two ways:

- **Dedicated assignment** – Resources used exclusively by specific tenants.
- **Shared assignment** – Resources available to multiple tenants.

When you associate a resource with a tenant, that tenant gains permission to use that resource for email sending. With each send request, SES validates that the specified tenant has permission to use the identity, configuration set, and template in the request. If the resources aren't properly associated, the send request fails.

Reputation monitoring and enforcement

SES continuously monitors key reputation metrics for each tenant, including bounce rates, complaint rates (including those from the mailbox provider Feedback Loop (FBL) system), and third-party feedback signals. When these metrics exceed defined thresholds, SES creates "reputation findings" categorized as:

- **Low severity warnings** – Minor issues that could affect deliverability if not addressed.
- **High severity warnings** – Serious issues likely affecting deliverability that may trigger enforcement.

Based on these findings, SES can automatically pause problematic tenants through reputation policies that you configure. Three enforcement levels are available:

- **Standard** (recommended) – Automatically pauses tenant sending when high-severity reputation findings are detected. This provides balanced protection while minimizing disruption.
- **Strict** – Automatically pauses tenant sending when any reputation finding is detected, including low-severity issues. Provides maximum protection but may result in more frequent pausing.
- **None** – Disables automated pausing for the tenant. All reputation findings are still recorded and visible, but no automated enforcement actions are taken.

When a tenant's metrics trigger a reputation finding that meets the threshold for enforcement under your selected policy, the system automatically updates the tenant's sending status to "Paused" without affecting the sending capability of other tenants. While a tenant is paused, any attempt to send email using that tenant will fail until you review the issue and manually re-enable sending. Additionally, you can manually pause a tenant's sending capabilities when needed, and unpause it when ready to resume sending.

Setting up tenants

Topics

- Creating tenants
- Assigning resources to a tenant
- Configuring reputation policies

Creating tenants

Using the console:

1. Sign in to the AWS Management Console and open the SES console at https://console.aws.amazon.com/ses/.
2. In the navigation pane, choose **Tenants**.
3. Choose **Create tenant**.
4. For **Tenant name**, enter a unique name for your tenant.
5. Choose **Create tenant**.

Using the AWS CLI:

```
aws sesv2 create-tenant \
    --tenant-name "MyTenant" \
    --region us-east-1
```

Assigning resources to a tenant

After creating a tenant, you must assign at least one verified identity and one configuration set before the tenant can send email.

Using the console:

1. In the SES console, navigate to the **Tenants** page.
2. Select the tenant you want to configure.
3. In the **Tenant setup** section, you can use the **Identities** and **Configuration sets** cards to assign these resources. Optionally, you can scroll down to the tabbed section and use the **Identities** and **Configuration sets** tabs to do the same.

   > **Note**
   >
   > Any attempt to delete an identity or configuration set that is associated with a tenant will fail. You must first remove these associations from the tenant before you can delete the associated resources.

4. (Optional) You can assign one or more tags to your tenant by selecting the **Tags** tab.

Using the AWS CLI:

```
# Assign an identity to a tenant
aws sesv2 create-tenant-resource-association \
    --tenant-name "MyTenant" \
    --resource-arn "arn:aws:ses:us-east-1:123456789012:identity/example.com" \
    --region us-east-1

# Assign a configuration set to a tenant
aws sesv2 create-tenant-resource-association \
    --tenant-name "MyTenant" \
    --resource-arn "arn:aws:ses:us-east-1:123456789012:configuration-set/MyConfigSet" \
    --region us-east-1
```

Configuring reputation policies

Reputation policies determine when a tenant's email sending is automatically paused based on reputation metrics. When you create a tenant, SES automatically assigns the *Standard* reputation policy. Use the following steps if you want to change to another policy.

Using the console:

1. In the SES console, navigate to the **Tenants** page.

2. Select the tenant you want to configure.
3. In the **Tenant setup** section, you can use the **Reputation policy** card to assign a policy. Optionally, you can scroll down to the tabbed section and use the **Reputation policy** tab to do the same.
4. Select one of the following policies:

   - **Standard (recommended)** – Pause sending when high-severity findings are detected.
   - **Strict** – Pause sending when any findings (including low-severity) are detected.
   - **None** – Do not automatically pause sending regardless of findings. *There are risks associated with this level as explained in Trust & Safety.*

Using the AWS CLI:

```
update-reputation-entity-policy \
    --reputation-entity-type "RESOURCE" \
    --reputation-entity-reference "arn:aws:ses:us-east-1:123456789012:tenant/tenantId"
 \
    --reputation-entity-policy "arn:aws:ses:us-east-1:aws:reputation-policy/standard"
```

Sending email with tenants

When sending email through a tenant, you must specify the tenant in your API calls or SMTP headers and ensure that all resources used are associated with that tenant.

Topics

- Using the SendEmail API with tenants
- Using SMTP with tenants

Using the SendEmail API with tenants

AWS CLI Example:

```
aws sesv2 send-email \
  --tenant-name "MyTenant" \
  --from-email-address "sender@example.com" \
  --destination "ToAddresses=recipient@example.com" \
```

```
  --content "Simple={Subject={Data='Test Subject',Charset=utf-8},Body={Text={Data='Test
 email body',Charset=utf-8}}}" \
  --configuration-set-name "MyConfigSet"
```

AWS SDK for Python Example:

```
import boto3

client = boto3.client('sesv2')
response = client.send_email(
    FromEmailAddress='sender@example.com',
    Destination={
        'ToAddresses': ['recipient@example.com']
    },
    Content={
        'Simple': {
            'Subject': {
                'Data': 'Test email'
            },
            'Body': {
                'Text': {
                    'Data': 'This is a test email sent using a tenant.'
                }
            }
        }
    },
    ConfigurationSetName='MyConfigurationSet',
     TenantName='MyTenant'
)
```

Using SMTP with tenants

When using SMTP to send email through a tenant, include the tenant information in an email header:

```
X-SES-TENANT: MyTenant
```

This header tells SES which tenant should be used for the email sending operation, allowing SES to apply the appropriate resource validation and reputation tracking.

Managing tenant status

Topics

- Viewing tenant status and metrics
- Pausing and unpausing tenants

Viewing tenant status and metrics

Using the console:

1. In the SES console, navigate to the **Tenants** page.
2. Select a tenant to view its details.
3. The **Tenant status** box displays:

   Sending Status:.

   - **Enabled** – The tenant can send emails.
   - **Paused** – You or SES automated policies have paused sending for this tenant.
   - **Enforced** – SES has paused sending due to serious reputation issues.
   - **Reinstated** – Sending was reactivated after being paused.

   Reputation Status:

   - **No findings detected** – No issues affecting deliverability.
   - **Low severity warning** – Minor issues that could affect deliverability if not addressed.
   - **High severity warning** – Serious issues likely affecting deliverability.
4. Scroll down to **Sending statistics** under the **Resources** tab to view delivery, bounce, and complaint metrics for the date range selected.

Using the AWS CLI:

```
# Get tenant details
aws sesv2 get-tenant --tenant-name "MyTenant"
```

If you've enabled Virtual Deliverability Manager for your account, the Virtual Deliverability Manager dashboard provides per-tenant deliverability metrics including send volume, bounces, complaints, opens, and clicks, with drill-down by ISP. Each tenant's detail page shows associated sending identities, configuration sets, and reputation findings. For more information, see Using the dashboard (console). To get started with Virtual Deliverability Manager, see Getting started.

Pausing and unpausing tenants

You can manually pause a tenant's sending capabilities when needed, and unpause it when ready to resume sending.

Using the console:

1. In the SES console, navigate to the **Tenants** page.
2. Select the checkbox next to the tenant you want to pause or unpause.
3. Choose **Pause sending** or **Resume sending**.
4. Confirm the action.

Using the AWS CLI:

To pause a tenant:

```
# Pause a tenant
aws sesv2 update-reputation-entity-customer-managed-status \
  --reputation-entity-type RESOURCE
  --reputation-entity-reference "arn:aws:ses:us-east-1:593442965613:tenant/tenantId"
  --sending-status DISABLED
```

To unpause a tenant:

```
# Unpause a tenant
aws sesv2 update-reputation-entity-customer-managed-status \
  --reputation-entity-type RESOURCE
  --reputation-entity-reference "arn:aws:ses:us-east-1:593442965613:tenant/tenantId"
  --sending-status ENABLED
```

Working with reputation findings

Topics

- Viewing reputation findings
- Understanding reputation findings
- Resolving reputation issues

Viewing reputation findings

Using the console:

1. In the SES console, navigate to the **Tenants** page.
2. Select the tenant you want to examine.
3. Navigate to the **Reputation findings** table.
4. Review any active findings, including their type, severity, detection date, and guidance to resolve the issue.

Using the AWS CLI:

```
aws sesv2 list-recommendations \
   --filter='{ "RESOURCE_ARN":"arn:aws:ses:us-east-1:012345678901:tenant/tenantId"}'
```

The response includes details about all active findings:

```
{
    "Recommendations": [
        {
            "ResourceArn": "arn:aws:ses:us-east-1:012345678901:tenant/{tenant-name}/
{tenant-id}",
            "Type": "BOUNCE",
            "Description": "The bounce rate exceeded 15.0% based on a representative
 volume of 664 emails from July 11, 2025 at 14:41 (UTC) to July 11, 2025 at 16:26
 (UTC).",
            "Status": "OPEN",
            "CreatedTimestamp": "2025-07-11T16:16:14.029000+00:00",
            "LastUpdatedTimestamp": "2025-07-11T16:37:14.145000+00:00",
            "Impact": "HIGH"
        }
    ]
}
```

Understanding reputation findings

Reputation findings provide insights into potential issues with your tenants' email sending practices. Each finding includes:

- **Impact** – Severity level (high or low).
- **Finding type** – Bounce rates, complaint rates, third-party feedback from mailbox providers, and IP blocklist listings.
- **Age** – Time since first detected date.
- **Description** – Context about the problem (such as the specific rate that triggered the finding).
- **Last checked** – Date of most recent status update.
- **Resolve issue** – Links to the relevant section in the SES Developer Guide with guidance to help resolve the issue.

Common findings include:

- **High bounce rates** – When bounce rates exceed configured thresholds.
- **Complaint activity** – When spam reports are received from mailbox providers that exceed thresholds.
- **Third-party feedback** – Negative signals from mailbox providers.
- **Blocklist appearances** – When sending IPs appear on a reputation blocklist.

Resolving reputation issues

When you receive a reputation finding, it's important to investigate the root cause and take corrective action promptly:

1. **Address the root cause** – Advise your tenant to improve list hygiene, update email content, or fix technical issues.
2. **Review sending practices** – Ensure your tenant complies with email best practices.
3. **Monitor metrics** – Watch for improvement in bounce and complaint rates.
4. **Communicate with affected parties** – Inform relevant stakeholders about the issue and resolution steps.

Monitoring and analytics

Topics

- Setting up CloudWatch metrics
- Setting up EventBridge notifications

Setting up CloudWatch metrics

SES publishes tenant-specific metrics to Amazon CloudWatch. The following metrics are available for each tenant:

- **Sends** – Total number of emails sent by the tenant.
- **Bounces** – Number of bounced emails for the tenant.
- **Complaints** – Number of bounced emails for the tenant.

Accessing tenant metrics in CloudWatch:

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Select the **AWS/SES** namespace.
4. Choose **By TenantId** and **TenantName** to view tenant-specific metrics.

Setting up EventBridge notifications

By default, SES sends events to the EventBridge default event bus when tenant reputation findings are detected or when tenant status changes occur.

You can create rules on the default event bus to identify specific events for EventBridge to send to one or more specified targets.

The following detail-types are available:

For tenant sending status changes:

- `Sending Status Enabled` – The new sending status is `ENABLED`.
- `Sending Status Disabled` – The new sending status is `DISABLED` or `REINSTATED`.

Note

SES sends `Sending Status Disabled` for both a pause and a reinstatement. Read `detail.data.record.status` to tell the two apart. A status of `DISABLED` means the origin cannot send email. A status of `REINSTATED` means the origin can send email again. Do not use the detail-type alone to decide if a tenant can send email.
The status in the event applies only to the origin named in `detail.data.origin`. It does not describe the tenant as a whole. If either origin is `DISABLED`, the tenant cannot send email. To read the combined status, call GetReputationEntity and use the `SendingStatusAggregate` value.

For reputation findings:

- `Advisor Recommendation Status Open`
- `Advisor Recommendation Status Resolved`

Setting up EventBridge rules

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. Choose **Create rule**.
3. Enter a name and description for your rule.
4. For **Define pattern**, choose **Event pattern**.
5. Choose **Pre-defined pattern by service**.
6. Choose **SES** as the service name.
7. For **Event type**, choose **Sending Status Enabled** or **Sending Status Disabled**.
8. Configure the event pattern to match specific events you're interested in.
9. Choose your target (such as an AWS Lambda function, Amazon SNS topic, or Amazon SQS queue).
10. Configure any additional settings as needed.
11. Choose **Create**.

Example EventBridge rule pattern for reputation findings:

```
{
  "detail-type": "Advisor Recommendation Status Open",
  "source": "aws.ses",
  "account": "012345678901",
  "time": "2023-11-15T17:00:59Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ses:us-east-1:012345678901:tenant/{tenant-name}/{tenant-id}"
  ],
  "detail": {
       "version": "1.0.0",
       "data": "The bounce rate exceeded 15.0% based on a representative volume of 197
 emails from July 11, 2025 at 14:43 (UTC) to July 11, 2025 at 16:13 (UTC).",
       "metadata":{"impact":"HIGH","type":"BOUNCE"}

    }
}
```

Example EventBridge rule pattern for sending status changes:

```
{
        "detail-type": "Sending Status Disabled",
        "source": "aws.ses",
        "account": "012345678901",
        "time": "2025-07-24T12:44:28Z",
        "region": "us-east-1",
        "resources": [
            "arn:aws:ses:us-east-1:012345678901:tenant/{tenant-name}/{tenant-id}"
        ],
        "detail": {
            "version": "1.0.0",
            "data": {
                "origin": "CUSTOMER_MANAGED",
                "record": {
                    "status": "DISABLED",
                    "cause": "Status manually updated.",
                    "lastUpdatedTimestamp": [2025, 7, 24, 12, 44, 28, 995000000]
                }
            }
        }
    }
```

Field descriptions:

- **source** – Identifies the service that generated the event. For SES events, this value is `aws.ses`.
- **detail-type** – The type of status change event (see Event Types above).
- **resources** – Array containing the ARN of the affected tenant.
- **detail.data.origin** – Source of the status change: `CUSTOMER_MANAGED` (you or your application changed the status) or `AWS_SES` (SES changed it). These are the only two values.
- **detail.data.record.status** – The new status for the origin in `detail.data.origin` (`ENABLED`, `DISABLED`, or `REINSTATED`).
- **detail.data.record.cause** – Description of why the status changed.
- **detail.data.record.lastUpdatedTimestamp** – Timestamp when the status was updated.

You can use these events to monitor tenant status changes and automate responses in your applications. For example, you might want to trigger alerts when tenants are disabled or track tenant health metrics over time.

Integration with AWS Trust & Safety

When AWS Trust & Safety detects issues that would normally result in account-level enforcement, the tenant system enables more targeted responses. Instead of pausing your entire account, Trust & Safety can pause only the problematic tenants while allowing compliant tenants to continue sending.

This tenant-level enforcement reduces the impact of reputation issues and helps maintain business continuity for your other email streams.

You'll receive notifications when SES has disabled sending due to high-severity issues affecting tenant reputation and has placed your account under review. A case will be opened for you in the AWS Support Center so that you can work with Trust & Safety to resolve issues and restore sending for the affected tenant.

> **Note**
>
> As the account owner, it's your responsibility to monitor the reputation metrics of all your tenants. While the tenant feature isolates reputation tracking at the tenant level, their combined sending activity still affects your overall account reputation—tenants that develop poor sending practices could put your entire account at risk. Therefore, it's

essential to ensure all tenants maintain good email sending practices to protect your account's standing.

Best practices

Follow these best practices to effectively manage tenant reputation in Amazon SES:

- **Start with the Standard policy** – For most tenants, the Standard reputation policy provides a good balance between protection and operational stability.
- **Monitor before enforcing** – When onboarding new tenants, consider temporarily using the "None" policy while monitoring their sending patterns (such as with EventBridge) before enabling automated enforcement.
- **Set up EventBridge notifications** – Create alerts for reputation findings to take proactive action before automated pausing occurs.
- **Review tenant metrics regularly** – Monitor tenant-level sending statistics even in the absence of reputation findings to identify emerging patterns.
- **Educate your tenants** – Provide guidelines on Email best practices to help tenants maintain good sending reputation.
- **Apply appropriate policies** – Apply the Strict policy to high-risk tenants or those with a history of reputation issues.
- **React quickly to findings** – When a finding is detected, investigate the root cause immediately and take appropriate corrective action.
- **Resource planning** – Design your tenant structure to match your business needs. Tenants can represent multiple customers (ISVs), business units, client/application types, and regulatory requirements.

Tenant-level suppression lists

By default, all tenants in your account share the account-level suppression list. This means that when a recipient bounces or complains about one tenant's email, that address is suppressed for every tenant in the account. Tenant-level suppression lists solve this by giving each tenant its own isolated suppression list.

When you enable tenant-level suppression for a tenant, SES maintains a separate suppression list for that tenant. Bounces and complaints only affect the tenant that sent the email. Other tenants

can continue sending to the same recipients. You configure tenant-level suppression by setting the `SuppressionScope` to `TENANT` and choosing which `SuppressedReasons` (bounces, complaints, or both) to track.

For complete information about configuring and managing tenant-level suppression lists, see Using tenant-level suppression lists in Amazon SES.

Limitations

When using tenant management, be aware of these limitations:

- **Regional scope** – Tenants are region-specific and are not automatically replicated across AWS Regions. If you send email from multiple regions, you'll need to configure and monitor tenant reputation separately in each region.
- **Quota limits** – By default, accounts can create up to 10,000 tenants. You can request increases through the AWS Service Quota Console, with automatic approval available up to 300,000 tenants for qualifying accounts.
- **Configuration set requirement** – When sending on behalf of a tenant, you must specify a configuration set that is associated with that tenant, or use an identity that has a default configuration set associated with the tenant.
- **Metric calculation periods** – Reputation metrics are calculated based on recent sending activity, typically over a rolling 24-hour to 7-day period depending on the metric type.
- **Minimum sending volume** – Some reputation findings require a minimum representative volume of emails before they can be triggered.
- **Re-enabling grace period** – After re-enabling a paused tenant, they enter into a 'reinstated' state where active reputation findings are temporarily ignored to allow the tenant to recover. The tenant will remain in this state until all active findings are resolved.
- **Cross-account limitations** – Tenants cannot span multiple AWS accounts. Each account manages its own set of tenants independently.
- **No tenant nesting** – Tenants cannot contain other tenants - they are flat structures.

Pricing

There's an additional charge per tenant per month based on the number of emails. For detailed pricing information, see the SES pricing page.

When using CloudWatch with tenant metrics, the standard CloudWatch metrics for each tenant are provided at no additional cost as part of the basic monitoring. Additional CloudWatch features like custom dashboards, alarms, or detailed monitoring may incur standard CloudWatch charges.

Verified identities in Amazon SES

In Amazon SES, a *verified identity* is a domain or email address that you use to send or receive email. Before you can send an email using Amazon SES, you must create and verify each identity that you're going to use as a "From", "Source", "Sender", or "Return-Path" address. Verifying an identity with Amazon SES confirms that you own it and helps prevent unauthorized use.

If your account is still in the Amazon SES sandbox, you also need to verify any email addresses which you plan on sending email to, unless you're sending to test inboxes provided by the Amazon SES mailbox simulator. For more information, see the section called “Using the mailbox simulator manually”.

You can create an identity by using the Amazon SES console or the Amazon SES API. The identity verification process depends on which type of identity you choose to create.

> **Tip**
>
> If you're a first time user of SES, you can use the Get started wizard to create and verify your first identity (email address or domain).

Contents

- Creating and verifying identities in Amazon SES
- Managing identities in Amazon SES
- Configuring identities in Amazon SES
- Sending test emails in Amazon SES with the simulator

Creating and verifying identities in Amazon SES

In Amazon SES, you can create an identity at the domain level or you can create an email address identity. These identity types aren’t mutually exclusive. In most cases, creating a domain identity eliminates the need for creating and verifying individual email address identities, unless you want to apply custom configurations to a specific email address. Whether you create a domain and utilize email addresses based on the domain, or create individual email addresses, there are benefits to both approaches. Which method you choose is dependent on your specific needs as discussed below.

Creating and verifying an email address identity is the fastest way to get started in SES, but there are benefits to verifying an identity at the domain level. When you verify an email address identity, only that email address can be used to send mail, but when you verify a domain identity, you can send email from any subdomain or email address of the verified domain without having to verify each one individually. For example, if you create and verify a domain identity called example.com, you don't need to create separate subdomain identities for a.example.com, a.b.example.com, nor separate email address identities for user@example.com, user@a.example.com, and so on.

However, keep in mind that an email address identity that's using the inherited verification from its domain is limited to straightforward email sending. If you want to do more advanced sending, you'll have to also explicitly verify it as an email address identity. Advanced sending includes using the email address with configuration sets, policy authorizations for delegate sending, and configurations that override the domain settings.

To help clarify the verification inheritance and email sending capabilities discussed above, the following table categorizes each combination of domain/email address verification and lists the inheritance, sending level, and display status for each:

	Only domain verified	**Only email address verified**	**Both domain & email address verified**
Inheritance level	Subdomains and email addresses inherit verification from the parent domain.	Email address explicitly verified.	• Subdomains inherit verification from the parent domain. • Email address explicitly verified.
Sending level	Email addresses limited to straightforward email sending.	Email address can be used in *advanced sending*[*].	Email address can be used in *advanced sending*[*].
Displayed status	Console/API status: • Domain/Subdomains = Verified	Console/API status: • Email address = Verified	Console/API status: • Domain/Subdomains = Verified

	Only domain verified	Only email address verified	Both domain & email address verified
	• Email address = Unverified.		• Email address = Verified.

**Advanced sending includes using the email address with configuration sets, policy authorizations for delegate sending, and configurations that override the domain settings.*

To send email from the same domain or email address in more than one AWS Region, you must create and verify a separate identity for each Region. You can verify as many as 10,000 identities in each Region.

When you create and verify domain and email address identities, consider the following:

- You can send email from any subdomain or email address of the verified domain without having to verify each one individually. For example, if you create and verify an identity for example.com, you don't need to create separate identities for a.example.com, a.b.example.com, user@example.com, user@a.example.com, and so on.
- As specified in RFC 1034, each DNS label can have up to 63 characters, and the whole domain name must not exceed a total length of 255 characters.
- If you verify a domain, subdomain, or email address that shares a root domain, the identity settings (such as feedback notifications) apply at the most granular level you verified.
  - Verified email address identity settings override verified domain identity settings.
  - Verified subdomain identity settings override verified domain identity settings, with lower-level subdomain settings overriding higher-level subdomain settings.

    For example, assume you verify user@a.b.example.com, a.b.example.com, b.example.com, and example.com. These are the verified identity settings that will be used in the following scenarios:

    - Emails sent from user@example.com (an email address that isn't specifically verified) will use the settings for example.com.
    - Emails sent from user@a.b.example.com (an email address that is specifically verified) will use the settings for user@a.b.example.com.

  - Emails sent from user@b.example.com (an email address that isn't specifically verified) will use the settings for b.example.com.
- You can add labels to verified email addresses without performing additional verification steps. To add a label to an email address, add a plus sign (+) between the account name and the "at" sign (@), followed by a text label. For example, if you already verified sender@example.com, you can use sender+myLabel@example.com as the "From" or "Return-Path" address for your emails. You can use this feature to implement Variable Envelope Return Path (VERP). Then you can use VERP to detect and remove undeliverable email addresses from your mailing lists.
- Domain names are case-insensitive. If you verify example.com, you can send from EXAMPLE.com also.
- Email addresses *are* case sensitive. If you verify sender@EXAMPLE.com, you can't send email from sender@example.com unless you verify sender@example.com as well.
- In each AWS Region, you can verify as many as 10,000 identities (domains and email addresses, in any combination).

Tip

If you're a first time user of SES, you can use the Get started wizard to create and verify your first identity (email address or domain).

Contents

- Creating a domain identity
- Verifying a DKIM domain identity with your DNS provider
- Creating an email address identity
- Verifying an email address identity
- Create and verify an identity and assign a default configuration set at the same time
- Using custom verification email templates

Creating a domain identity

Part of creating a domain identity is configuring its DKIM-based verification. DomainKeys Identified Mail (DKIM) is an email authentication method that Amazon SES uses to verify domain ownership,

and receiving mail servers use to validate email authenticity. You can choose to configure DKIM by using either Easy DKIM or Bring Your Own DKIM (BYODKIM), and depending on your choice, you'll have to configure the signing key length of the private key as follows:

- **Easy DKIM** - either accept the Amazon SES default of 2048 bits, or override it by selecting 1024 bits.
- **BYODKIM** - private key length must be at least 1024 bits and up to 2048-bits.

See the section called “DKIM signing key length” to learn more about DKIM signing key lengths and how to change them.

The following procedure shows you how to create a domain identity using the Amazon SES console.

- If you've already created your domain and just need to verify it, skip to the procedure the section called “Verifying a domain identity” on this page.

To create a domain identity

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. Choose **Create identity**.
4. Under **Identity details**, select **Domain** as the type of identity you want to create. You must have access to the domain’s DNS settings to complete the domain verification process.
5. Enter the name of the domain or subdomain in the **Domain** field.

   > **Tip**
   >
   > If your domain is *www.example.com*, enter *example.com* as your domain. Don't include the "www." part, because the domain verification process won't succeed if you do.

6. (Optional) If you want to **Assign a default configuration set**, select the check box.

   1. For **Default configuration set**, select the existing configuration set that you want to assign to your identity. If you haven’t created any configuration sets yet, see *Configuration sets*.

> **Note**
>
> Amazon SES only defaults to the assigned configuration set when no other set is specified at the time of sending. If a configuration set is specified, Amazon SES applies the specified set in place of the default set.

7. (Optional) If you want to **Use a custom MAIL FROM domain**, select the check box and complete the following steps. For more information, see the section called "Using a custom MAIL FROM domain".

   1. For **MAIL FROM domain**, enter the subdomain that you want to use as the MAIL FROM domain. This must be a subdomain of the domain identity that you're verifying. The MAIL FROM domain shouldn't be a domain from which you send email.
   2. For **Behavior on MX failure**, indicate which action Amazon SES should take if it can't find the required MX record at the time of sending. Choose one of the following options:
      - **Use default MAIL FROM domain** - If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES will use a subdomain of amazonses.com. The subdomain varies based on the AWS Region in which you use Amazon SES.
      - **Reject message** - If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES will return a `MailFromDomainNotVerified` error. If you choose this option, emails that you attempt to send from this domain are automatically rejected.
   3. For **Publish DNS records to Route53**, if your domain is hosted through Amazon Route 53, you have the option to let SES publish the associated TXT and MX records at the time of creation by leaving **Enabled** checked. If you'd rather publish these records later, clear the **Enabled** checkbox. (You can come back at a later time to publish the records to Route 53 by editing the identity - see the section called "Edit an identity using the console".)

8. (Optional) To configure customized DKIM-based verification, other than the SES default setting which uses Easy DKIM *with a 2048 bit signing length*, under **Verifying your domain**, expand **Advanced DKIM settings** and choose the type of DKIM you want to configure:

   a. **Easy DKIM**:

      i. Under **Identity type**, choose **Easy DKIM**.

      ii. In the **DKIM signing key length** field, choose either **RSA_2048_BIT** or **RSA_1024_BIT**.

      iii. For **Publish DNS records to Route53**, if your domain is hosted through Amazon Route 53, you have the option to let SES publish the associated CNAME records at

the time of creation by leaving **Enabled** checked. If you'd rather publish these records later, clear the **Enabled** checkbox. (You can come back at a later time to publish the records to Route 53 by editing the identity - see the section called "Edit an identity using the console".)

b. **Deterministic Easy DKIM (DEED)**:

> **Tip**
>
> This form of DKIM is to be used if you're creating a Global (replica) identity. DEED will utilize the Easy DKIM setup of an existing identity of the same name from a parent region and sign the new identity without requiring you to perform additional DNS setup. For more information, see DEED.

i. Under **Identity type**, choose **Deterministic Easy DKIM**.

ii. From the **Parent region** dropdown menu, select a parent region where an Easy DKIM-signed identity with the same name as you entered for your Global (replica) identity resides. (Your replica region defaults to the region you signed into the SES console with.)

c. **Provide DKIM authentication token (BYODKIM)**:

i. Ensure you've already generated a public-private key pair and have added the public key to your DNS host provider. For more information, see the section called "BYODKIM - Bring Your Own DKIM".

ii. Under **Identity type**, choose **Provide DKIM authentication token (BYODKIM)**.

iii. For **Private key**, paste the private key you generated from your public-private key pair. The private key must use at least 1024-bit RSA encryption and up to 2048-bit, and must be encoded using base64 (PEM) encoding.

> **Note**
>
> You have to delete the first and last lines (`-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----`, respectively) of the generated private key. Additionally, you have to remove the line breaks in the generated private key. The resulting value is a string of characters with no spaces or line breaks.

iv. For **Selector name**, enter the name of the selector to be specified in your domain's DNS settings.

9. Ensure that the **Enabled** box is checked in the **DKIM signatures** field.
10. (Optional) Add one or more **Tags** to your domain identity by including a tag key and an optional value for the key:
    1. Choose **Add new tag** and enter the **Key**. You can optionally add a **Value** for the tag.
    2. Repeat for additional tags not to exceed 50, or choose **Remove** to remove tags.
11. Choose **Create identity**.

Now that you've created and configured your domain identity with DKIM, you must complete the verification process with your DNS provider - proceed to the section called "Verifying a domain identity" and follow the DNS authentication procedures for the type of DKIM you configured your identity with.

Note

Verifying a DKIM domain identity with your DNS provider

After you've created your domain identity configured with DKIM, you must complete the verification process with your DNS provider by following the respective authentication procedures for the type of DKIM you chose.

If you haven't created a domain identity, see the section called "Creating a domain identity".

Note

- Verifying a domain identity requires access to the domain's DNS settings. Changes to these settings can take up to 72 hours to propagate.
- If you created a Global (replica) identity using Deterministic Easy DKIM (DEED), no additional DNS setup is required—you can skip this step. For more information, see DEED.

To verify a DKIM domain identity with your DNS provider

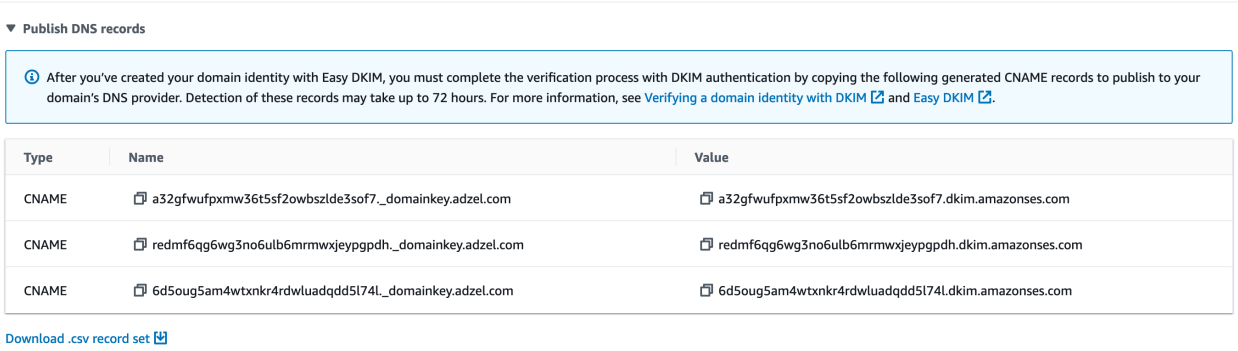
1. From the **Loaded identities** table, select the domain you want to verify.
2. On the **Authentication** tab of the identity details page, expand **Publish DNS records**.
3. Depending on which flavor of DKIM you configured your domain with, **Easy DKIM** or **BYODKIM**, follow the respective instructions:

   Easy DKIM

   #### To verify a domain configured with Easy DKIM

   1. From the **Publish DNS records** table, copy the three CNAME records that appear in this section to be published (added) to your DNS provider. Alternatively, you can choose **Download .csv record set** to save a copy of the records to your computer.

      The following image shows an example of the CNAME records to publish to your DNS provider.



> **Retrieving the hosted zone programmatically**
>
> Each CNAME record's value is composed of the DKIM token followed by a hosted zone domain (for example, *`token`*`.dkim.`*`us-west-2`*`.amazonses.com`). The hosted zone portion varies by AWS Region and identity.
>
> You can retrieve the hosted zone directly from the API response. Both the CreateEmailIdentity and GetEmailIdentity responses include a `SigningHostedZone` field in the `DkimAttributes` object. Use this value to construct the full CNAME record value: *`token`*`.`*`SigningHostedZone`*.

2. Add the CNAME records to your domain's DNS settings respective of your DNS host provider:

   - **All DNS host providers** *(excluding Route 53)* – Login to your domain's DNS or web hosting provider, and then add the CNAME records containing the values that you copied or saved previously. Different providers have different procedures for updating DNS records. See the DNS/Hosting provider table following these procedures.

     > **Note**
     >
     > A small number of DNS providers don't allow you to include underscores (_) in record names. However, the underscore in the DKIM record name is required. If your DNS provider doesn't allow you to enter an underscore in the record name, contact the provider's customer support team for assistance.

   - **Route 53 as your DNS host provider** – If you use Route 53 on the same account that you use when you send email using SES, and the domain is registered, SES automatically updates the DNS settings for your domain if you enabled SES to publish them at the time of creation. Otherwise, you can easily publish them to Route 53 with a button click after creation - see the section called "Edit an identity using the console". If your DNS settings don't update automatically, or you want to add CNAME records to Route 53 that aren't on the same account you use when you send email using SES, complete the procedures in Editing records.
   - **If you're not sure who your DNS provider is** – Ask your system administrator for more information.

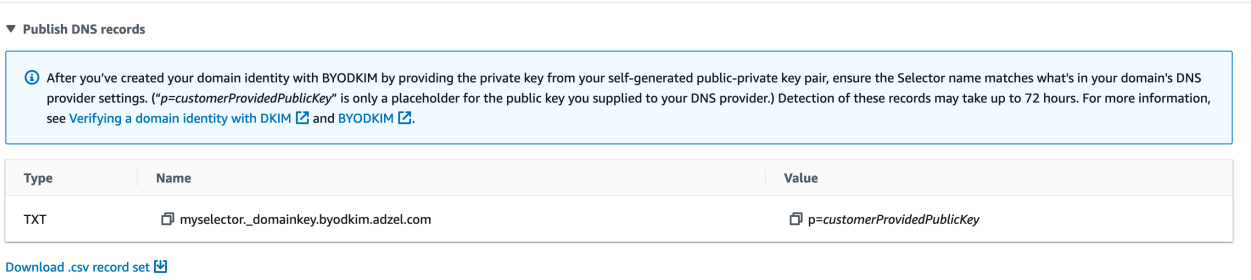
BYODKIM

To verify a domain configured with BYODKIM

1. To recap, when you created your domain with BYODKIM, or you configured an existing domain with BYODKIM, you added the private key (from your self-generated public-private key pair) and selector name prefix into their respective fields on the SES console's Advance DKIM Settings page. Now you must complete the verification process by updating the following records for your DNS host provider.

2. From the **Publish DNS records** table, copy the selector name record that appears in the **Name** column to be published (added) to your DNS provider. Alternatively, you can choose **Download .csv record set** to save a copy of it to your computer.

   The following image shows an example of the selector name record to publish to your DNS provider.



3. Login to your domain's DNS or web hosting provider, and then add the selector name record you copied or saved previously. Different providers have different procedures for updating DNS records. See the DNS/Hosting provider table following these procedures.

   > **Note**
   >
   > A small number of DNS providers don't allow you to include underscores (_) in record names. However, the underscore in the DKIM record name is required. If your DNS provider doesn't allow you to enter an underscore in the record name, contact the provider's customer support team for assistance.

4. If you haven't done so already, be sure to add the public key from your from your self-generated public-private key pair to your domain's DNS or web hosting provider.

   Note that in the **Publish DNS records** table, the public key record that appears in the **Value** column only displays, "p=*customerProvidedPublicKey*", as a placeholder for the public key value you saved to your computer or supplied to your DNS provider.

   > **Note**
   >
   > When you publish (add) your public key to your DNS provider, it must be formatted as follows:
   >
   > - You have to delete the first and last lines (`-----BEGIN PUBLIC KEY-----` and `-----END PUBLIC KEY-----`, respectively) of the generated public

key. Additionally, you have to remove the line breaks in the generated public key. The resulting value is a string of characters with no spaces or line breaks.

- You must include the p= prefix as shown in the *Value* column in the **Publish DNS records** table.

Tip

When adding CNAME records to your DNS configuration:

- Use the exact record names as provided in the SES console.
- Do not add any additional underscores (_) at the beginning of the CNAME record names.
- The `_domainkey` portion is part of the record name and should be used exactly as shown.

Example showing correct vs incorrect CNAME Record implementation:

- *Correct:* `abc123._domainkey.domain.com`
- *Incorrect:* `_abc123._domainkey.domain.com`

4. It can take up to 72 hours for changes to DNS settings to propagate. As soon as Amazon SES detects all of the required DKIM records in your domain's DNS settings, the verification process is complete. Your domain's **DKIM configuration** appears as **Successful** and the **Identity status** appears as **Verified**.
5. If want to configure and verify a custom MAIL FROM domain, follow the procedures in Configuring your custom MAIL FROM domain.

The following table includes links to the documentation for a few widely used DNS providers. This list isn't exhaustive and doesn't signify endorsement; likewise, if your DNS provider isn't listed, it doesn't imply you can't use the domain with Amazon SES.

DNS/Hosting provider	Documentation link
GoDaddy	Add a CNAME record (external link)

DNS/Hosting provider	Documentation link
DreamHost	How do I add custom DNS records? (external link)
Cloudflare	Managing DNS records in Cloudflare (external link)
HostGator	Manage DNS Records with HostGator/eNom (external link)
Namecheap	How do I add TXT/SPF/DKIM/DMARC records for my domain? (external link)
Names.co.uk	Changing your domains DNS Settings (external link)
Wix	Adding or Updating CNAME Records in Your Wix Account (external link)

Troubleshooting domain verification

If you completed the steps above, but your domain isn't verified after 72 hours, check the following:

- Make sure that you entered the values for the DNS records in the correct fields. Some DNS providers refer to the **Name/host** field as **Host** or **Hostname**. In addition, some providers refer to the **Record value** field as **Points to** or **Result**.
- Make sure that your provider didn't automatically append your domain name to the **Name/host** value that you entered in the DNS record. Some providers append the domain name without indicating that they've done so. If your provider appended your domain name to the **Name/host** value, remove the domain name from the end of the value. You can also try adding a period to the end of the value in the DNS record. This period indicates to the provider that the domain name is fully qualified.
- The underscore character (_) is required in the **Name/host** value of each DNS record. If your provider doesn't allow underscores in DNS record names, contact the provider's customer support department for additional assistance.

- The validation records that you have to add to your domain's DNS settings are different for each AWS Region. If you want to use a domain to send email from multiple AWS Regions, you have to create and verify a separate domain identity for each of those Regions.

Creating an email address identity

Complete the following procedure to create an email address identity by using the Amazon SES console.

To create an email address identity (console)

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. Choose **Create identity**.
4. Under **Identity details**, choose **Email address** as the identity type you want to create.
5. For **Email address**, enter the email address that you want to use. The email address must be an address that's able to receive mail and that you have access to.
6. (Optional) If you want to **Assign a default configuration set**, select the check box.
   1. For **Default configuration set**, select the existing configuration set that you want to assign to your identity. If you haven't created any configuration sets yet, see *Configuration sets*.

      > **Note**
      >
      > Amazon SES only defaults to the assigned configuration set when no other set is specified at the time of sending. If a configuration set is specified, Amazon SES applies the specified set in place of the default set.

7. (Optional) Add one or more **Tags** to your domain identity by including a tag key and an optional value for the key:
   1. Choose **Add new tag** and enter the **Key**. You can optionally add a **Value** for the tag.
   2. Repeat for additional tags not to exceed 50, or choose **Remove** to remove tags.

8. To create your email address identity, choose **Create identity**. After it's created, you should receive a verification email within five minutes. The next step is to verify your email address by following the verification procedure in the next section.

> **Note**
>
> You can customize the messages that are sent to the email addresses you attempt to verify. For more information, see the section called “Using custom verification email templates”.

Now that you’ve created your email address identity, you must complete the verification process - proceed to the section called “Verifying an email address identity”.

Verifying an email address identity

After you’ve created your email address identity, you must complete the verification process.

If you haven't created an email address identity, see the section called “Creating an email address identity”.

To verify an email address identity

1. Check the inbox of the email address used to create your identity and look for an email from no-reply-aws@amazon.com.
2. Open the email and click the link to complete the verification process for the email address. After it's complete, the **Identity status** updates to **Verified**.

Troubleshooting email address verification

If you don't receive the verification email within five minutes of creating your identity, try the following troubleshooting steps:

- Make sure you entered the email address correctly.
- Make sure the email address that you're attempting to verify can receive email. You can test this by using another email address to send a test email to the address that you want to verify.
- Check your junk mail folder.

- The link in the verification email expires after 24 hours. To send a new verification email, choose **Resend** at the top of the identity details page.

Create and verify an identity and assign a default configuration set at the same time

You can use the CreateEmailIdentity operation in the Amazon SES API v2 to create a new email identity and set its default configuration set at the same time.

> **Note**
>
> Before you complete the procedure in this section, you have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

To set a default configuration set using the AWS CLI

- At the command line, enter the following command to use the CreateEmailIdentity operation.

```
aws sesv2 create-email-identity --email-identity ADDRESS-OR-DOMAIN --configuration-set-name CONFIG-SET
```

In the preceding commands, replace *ADDRESS-OR-DOMAIN* with the email identity that you want to verify. Replace *CONFIG-SET* with the name of the configuration set you want to set as the default configuration set for the identity.

If the command executes successfully, it exits without providing any output.

To verify your email address

1. Check the inbox for the email address that you're verifying. You'll receive a message with the following subject line: "Amazon Web Services - Email Address Verification Request in region *RegionName*," where *RegionName* is the name of the AWS Region that you attempted to verify the email address in.

   Open the message, and then click the link in it.

> **Note**
>
> The link in the verification message expires 24 hours after the message was sent. If 24 hours have passed since you received the verification email, repeat steps 1–5 to receive a verification email with a valid link.

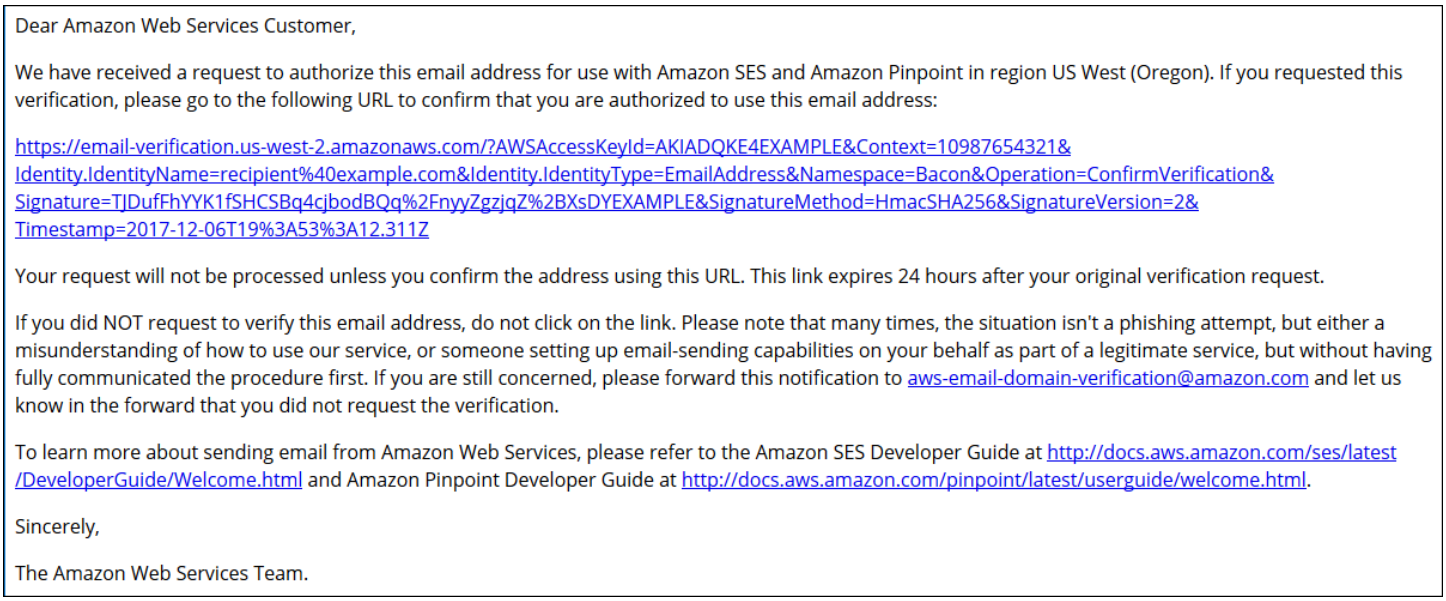
2. In the Amazon SES console, under **Identity Management**, choose **Email Addresses**. In the list of email addresses, locate the email address you're verifying. If the email address was verified, the value in the **Status** column is "verified".

To verify your domain

If you entered a domain name for the `--email-identity` parameter in the above command line procedure, see Verifying a domain identity for more information.

Using custom verification email templates

When you attempt to verify an email address, Amazon SES sends an email to that address that resembles the example shown in the following image.



Dear Amazon Web Services Customer,

We have received a request to authorize this email address for use with Amazon SES and Amazon Pinpoint in region US West (Oregon). If you requested this verification, please go to the following URL to confirm that you are authorized to use this email address:

https://email-verification.us-west-2.amazonaws.com/?AWSAccessKeyId=AKIADQKE4EXAMPLE&Context=10987654321&Identity.IdentityName=recipient%40example.com&Identity.IdentityType=EmailAddress&Namespace=Bacon&Operation=ConfirmVerification&Signature=TJDufFhYYK1fSHCSBq4cjbodBQq%2FnyyZgzjqZ%2BXsDYEXAMPLE&SignatureMethod=HmacSHA256&SignatureVersion=2&Timestamp=2017-12-06T19%3A53%3A12.311Z

Your request will not be processed unless you confirm the address using this URL. This link expires 24 hours after your original verification request.

If you did NOT request to verify this email address, do not click on the link. Please note that many times, the situation isn't a phishing attempt, but either a misunderstanding of how to use our service, or someone setting up email-sending capabilities on your behalf as part of a legitimate service, but without having fully communicated the procedure first. If you are still concerned, please forward this notification to aws-email-domain-verification@amazon.com and let us know in the forward that you did not request the verification.

To learn more about sending email from Amazon Web Services, please refer to the Amazon SES Developer Guide at http://docs.aws.amazon.com/ses/latest/DeveloperGuide/Welcome.html and Amazon Pinpoint Developer Guide at http://docs.aws.amazon.com/pinpoint/latest/userguide/welcome.html.

Sincerely,

The Amazon Web Services Team.

Several Amazon SES customers build applications (such as email marketing suites or ticketing systems) that send email through Amazon SES on behalf of their own customers. For the end users of these applications, the email verification process can be confusing: the verification email uses

Amazon SES branding, rather than the branding of the application, and those end users never signed up to use Amazon SES directly.

If your Amazon SES use case requires your customers to have their email addresses verified for use with Amazon SES, you can create customized verification emails. These customized emails help reduce customer confusion and increase the rates at which your customers complete the registration process.

> **Note**
>
> To use this feature, your Amazon SES account has to be out of the sandbox. For more information, see Request production access (Moving out of the Amazon SES sandbox).

Topics in this section:

- Creating a custom verification email template
- Editing a custom verification email template
- Sending verification emails using custom templates
- Custom verification email frequently asked questions

Creating a custom verification email template

To create a custom verification email, use the `CreateCustomVerificationEmailTemplate` API operation. This operation takes the following inputs:

Attribute	Description
`TemplateName`	The name of the template. The name you specify must be unique.
`FromEmailAddress`	The email address that the verification email is sent from. The address or domain you specify must be verified for use with your Amazon SES account.

Attribute	Description
	Note The `FromEmailAddress` attribute doesn't support display names (also known as "friendly from" names).
`TemplateSubject`	The subject line of the verification email.
`TemplateContent`	The body of the email. The email body can contain HTML, with certain restrictions. For more information, see Custom verification email frequently asked questions.
`SuccessRedirectionURL`	The URL that users are sent to, if their email addresses are successfully verified.
`FailureRedirectionURL`	The URL that users are sent to, if their email addresses are not successfully verified.

You can use the AWS SDKs or the AWS CLI to create a custom verification email template with the `CreateCustomVerificationEmailTemplate` operation. To learn more about the AWS SDKs, see Tools for Amazon Web Services. For more information about the AWS CLI, see AWS Command Line Interface.

The following section includes procedures for creating a custom verification email using the AWS CLI. These procedures assume that you have installed and configured the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

Note

To complete the procedure in this section, you must use version 1.14.6 or later of the AWS CLI. For best results, upgrade to the latest version of the AWS CLI. For more information about updating the AWS CLI, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

1. In a text editor, create a new file. Paste the following content into the editor:

```
{
  "TemplateName": "SampleTemplate",
  "FromEmailAddress": "sender@example.com",
  "TemplateSubject": "Please confirm your email address",
  "TemplateContent": "<html>
                      <head></head>
                      <body style='font-family:sans-serif;'>
                        <h1 style='text-align:center'>Ready to start sending
                        email with ProductName?</h1>
                        <p>We here at Example Corp are happy to have you on
                          board! There's just one last step to complete before
                          you can start sending email. Just click the following
                          link to verify your email address. Once we confirm that
                          you're really you, we'll give you some additional
                          information to help you get started with ProductName.</p>
                      </body>
                      </html>",
  "SuccessRedirectionURL": "https://www.example.com/verifysuccess",
  "FailureRedirectionURL": "https://www.example.com/verifyfailure"
}
```

Important

To make the preceding example easier to read, the `TemplateContent` attribute contains line breaks. If you paste the preceding example into your text file, remove the line breaks before proceeding.

Replace the values of `TemplateName`, `FromEmailAddress`, `TemplateSubject`, `TemplateContent`, `SuccessRedirectionURL`, and `FailureRedirectionURL` with your own values.

Note

The email address that you specify for the `FromEmailAddress` parameter has to be verified, or has to be an address on a verified domain. For more information, see Verified identities in Amazon SES.

When you finish, save the file as `customverificationemail.json`.

2. At the command line, type the following command to create the custom verification email template:

```
aws sesv2 create-custom-verification-email-template --cli-input-json file://
customverificationemail.json
```

3. (Optional) You can confirm that the template was created by typing the following command:

```
aws sesv2 list-custom-verification-email-templates
```

Editing a custom verification email template

You can edit a custom verification email template by using the `UpdateCustomVerificationEmailTemplate` operation. This operation accepts the same inputs as the `CreateCustomVerificationEmailTemplate` operation (that is, the `TemplateName`, `FromEmailAddress`, `TemplateSubject`, `TemplateContent`, `SuccessRedirectionURL`, and `FailureRedirectionURL` attributes). However, with the `UpdateCustomVerificationEmailTemplate` operation, none of these attributes are required. When you pass a value for `TemplateName` that is the same as the name of an existing custom verification email template, the attributes you specify overwrite the attributes that were originally in the template.

Sending verification emails using custom templates

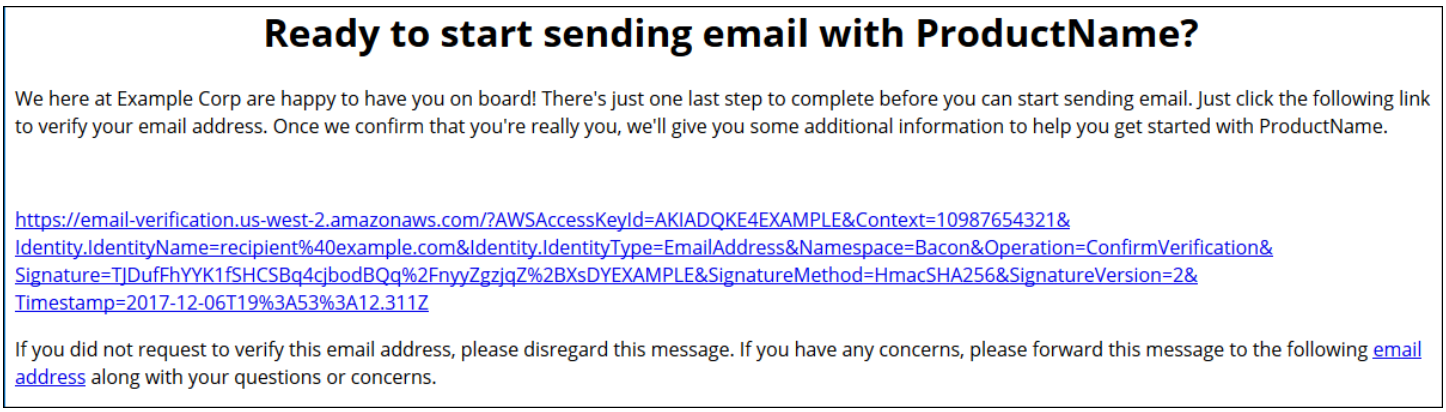
After you create at least one custom verification email template, you can send it to your customers by calling the SendCustomVerificationEmail API operation. You can call the `SendCustomVerificationEmail` operation by using any of the AWS SDKs or the AWS CLI. The `SendCustomVerificationEmail` operation takes the following inputs:

Attribute	Description
`EmailAddress`	The email address that is being verified.
`TemplateName`	The name of the custom verification email template that is sent to email address that is being verified.

Attribute	Description
`ConfigurationSetName`	(Optional) The name of a configuration set to use when sending the verification email.

For example, assume your customers register for your service using a form in your application. When the customer completes the form and submits it, your application calls the `SendCustomVerificationEmail` operation, passing the customer's email address and the name of the template you want to use.

Your customer receives an email that uses the customized email template you created. Amazon SES automatically adds a unique link to the recipient, and also a brief disclaimer. The following image shows a sample verification email that uses the template created in Creating a custom verification email template.



Ready to start sending email with ProductName?

We here at Example Corp are happy to have you on board! There's just one last step to complete before you can start sending email. Just click the following link to verify your email address. Once we confirm that you're really you, we'll give you some additional information to help you get started with ProductName.

https://email-verification.us-west-2.amazonaws.com/?AWSAccessKeyId=AKIADQKE4EXAMPLE&Context=10987654321&Identity.IdentityName=recipient%40example.com&Identity.IdentityType=EmailAddress&Namespace=Bacon&Operation=ConfirmVerification&Signature=TJDufFhYYK1fSHCSBq4cjbodBQq%2FnyyZgzjqZ%2BXsDYEXAMPLE&SignatureMethod=HmacSHA256&SignatureVersion=2&Timestamp=2017-12-06T19%3A53%3A12.311Z

If you did not request to verify this email address, please disregard this message. If you have any concerns, please forward this message to the following email address along with your questions or concerns.

Custom verification email frequently asked questions

This section contains answers to frequently asked questions about the custom verification email template feature.

Q1. How many custom verification email templates can I create?

You can create up to 50 custom verification email templates per Amazon SES account.

Q2. How do custom verification emails appear to recipients?

Custom verification emails include the content you specified when you created the template, followed by a link that recipients must click to verify their email addresses.

Q3. Can I preview the custom verification email?

To preview a custom verification email, use the `SendCustomVerificationEmail` operation to send a verification email to an address you own. If you don't click the verification link, Amazon SES does not create a new identity. If you do click the verification link, you can optionally delete the newly created identity by using the `DeleteIdentity` operation.

Q4. Can I include images in my custom verification email templates?

You can embed images in the HTML for your templates by using base64 encoding. When you embed images in this way, Amazon SES automatically converts them into attachments. You can encode an image at the command line by issuing one of the following commands:

Linux, macOS, or Unix

```
base64 -i imagefile.png | tr -d '\n' > output.txt
```

Windows

```
certutil -encodehex -f imagefile.png output.txt 0x40000001
```

Replace *`imagefile.png`* with the name of the file you want to encode. In both of the commands above, the base64 encoded image is saved to `output.txt`.

You can embed the base64-encoded image by including the following in the HTML for the template: `<img src="`*`base64EncodedImage`*`"/>`

In the previous example, replace *`png`* with the file type of the encoded image (such as jpg or gif), and replace *`base64EncodedImage`* with the base64 encoded image (that is, the contents of `output.txt` from one of the preceding commands).

Q5. Are there any limits to the content that I can include in custom verification email templates?

Custom verification email templates can't exceed 10 MB in size. Additionally, custom verification email templates that contain HTML can only use the tags and attributes listed in the following table.

HTML tag	Allowed attributes
abbr	class, id, style, title
acronym	class, id, style, title
address	class, id, style, title
area	class, id, style, title
b	class, id, style, title
bdo	class, id, style, title
big	class, id, style, title
blockquote	cite, class, id, style, title
body	class, id, style, title
br	class, id, style, title
button	class, id, style, title
caption	class, id, style, title
center	class, id, style, title
cite	class, id, style, title
code	class, id, style, title
col	class, id, span, style, title, width
colgroup	class, id, span, style, title, width
dd	class, id, style, title
del	class, id, style, title

HTML tag	Allowed attributes
dfn	class, id, style, title
dir	class, id, style, title
div	class, id, style, title
dl	class, id, style, title
dt	class, id, style, title
em	class, id, style, title
fieldset	class, id, style, title
font	class, id, style, title
form	class, id, style, title
h1	class, id, style, title
h2	class, id, style, title
h3	class, id, style, title
h4	class, id, style, title
h5	class, id, style, title
h6	class, id, style, title
head	class, id, style, title
hr	class, id, style, title
html	class, id, style, title
i	class, id, style, title
img	align, alt, class, height, id, src, style, title, width

HTML tag	Allowed attributes
input	class, id, style, title
ins	class, id, style, title
kbd	class, id, style, title
label	class, id, style, title
legend	class, id, style, title
li	class, id, style, title
map	class, id, style, title
menu	class, id, style, title
ol	class, id, start, style, title, type
optgroup	class, id, style, title
option	class, id, style, title
p	class, id, style, title
pre	class, id, style, title
q	cite, class, id, style, title
s	class, id, style, title
samp	class, id, style, title
select	class, id, style, title
small	class, id, style, title
span	class, id, style, title
strike	class, id, style, title

HTML tag	Allowed attributes
strong	class, id, style, title
sub	class, id, style, title
sup	class, id, style, title
table	class, id, style, summary, title, width
tbody	class, id, style, title
td	abbr, axis, class, colspan, id, rowspan, style, title, width
textarea	class, id, style, title
tfoot	class, id, style, title
th	abbr, axis, class, colspan, id, rowspan, scope, style, title, width
thead	class, id, style, title
tr	class, id, style, title
tt	class, id, style, title
u	class, id, style, title
ul	class, id, style, title, type
var	class, id, style, title

Note

Custom verification email templates can't include comment tags.

Q6. How many verified email addresses can exist in my account?

Your Amazon SES account can have up to 10,000 verified identities in each AWS Region. In Amazon SES, *identities* include both verified domains and email addresses.

Q7. Can I create custom verification email templates using the Amazon SES console?

Currently, it's only possible to create, edit, and delete custom verification emails using the Amazon SES API.

Q8. Can I track open and click events that occur when customers receive custom verification emails?

Custom verification emails can't include open or click tracking.

Q9. Can custom verification emails include custom headers?

Custom verification emails can't include custom headers.

Q10. Can I remove the text that appears at the bottom of custom verification emails?

The following text is automatically added to the end of every custom verification email and can't be removed:

If you did not request to verify this email address, please disregard this message.

Q11. Are custom verification emails DKIM-signed?

In order for verification emails to be DKIM-signed, the email address that you specify in the `FromEmailAddress` attribute when you create the verification email template must be configured to generate a DKIM signature. For more information about setting up DKIM for domains and email addresses, see the section called “Authenticating Email with DKIM”.

Q12. Why don't the custom verification email template API operations appear in the SDK or CLI?

If you're unable to use the custom verification email template operations in an SDK or the AWS CLI, you may be using an older version of the SDK or CLI. The custom verification email template operations are available in the following SDKs and CLIs:

- Version 1.14.6 or later of the AWS Command Line Interface

- Version 3.3.205.0 or later of the AWS SDK for .NET
- Version 1.3.20170531.19 or later of the AWS SDK for C++
- Version 1.12.43 or later of the AWS SDK for Go
- Version 1.11.245 or later of the AWS SDK for Java
- Version 2.166.0 or later of the AWS SDK for JavaScript
- Version 3.45.2 or later of the AWS SDK for PHP
- Version 1.5.1 or later of the AWS SDK for Python (Boto)
- Version 1.5.0 or later of the `aws-sdk-ses` gem in the AWS SDK for Ruby

Q13. Why do I receive `ProductionAccessNotGranted` errors when I send custom verification emails?

The `ProductionAccessNotGranted` error indicates that your account is still in the Amazon SES sandbox. You can only send custom verification emails if your account has been removed from the sandbox. For more information, see Request production access (Moving out of the Amazon SES sandbox).

Managing identities in Amazon SES

In the Amazon SES console, you can view your created identities for each AWS Region, open an identity to see and edit its detail settings, associate a default configuration set, or delete one or more identities.

> **Note**
>
> The procedures referenced in this section apply only to identities in the selected AWS Region. To manage identities that were created in more than one Region, repeat the procedures for each AWS Region.

Contents

- View identities using the SES console
- Delete an identity using the SES console
- Edit an identity using the SES console

- Edit an identity to use a default configuration set using the SES API
- Retrieve the default configuration set used by the identity using the SES API
- Override the current default configuration set used by the identity using the SES API

View identities using the SES console

You can use the Amazon SES console to view domain and email address identities that are verified or are pending verification. You can also view those identifies for which verification was unsuccessful.

To view your domain and email address identities

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the console, use the Region selector to choose the AWS Region for which you want to view your list of identities.

   > **Note**
   >
   > This procedure only displays a list of identities for the selected AWS Region.

3. In the navigation pane, under **Configuration**, choose **Verified identities**. The **Loaded identities** table displays both domain and email address identities. The **Status** column displays whether an identity has been verified, is pending verification, or has failed the verification process - definitions of all possible status values are as follows:

   - **Verified** – your identity is successfully verified for sending in SES.
   - **Failure** – SES was unable to verify your identity. If it's a domain, it means SES was unable to detect the DNS records within 72 hours. If it's an email address, it means the verification email that was sent to the email address was not acknowledged within 24 hours.
   - **Pending** – SES is still trying to verify the identity.
   - **Temporary Failure** – for a previously verified domain, SES will periodically check for the DNS record required for verification. If at some point, SES is unable to detect the record, the status would change to *Temporary Failure*. SES will recheck for the DNS record for 72 hours, and if it's unable to detect the record, the domain status would change to *Failure*. If it's able to detect the record, the domain status would change to *Verified*.

- **Not started** – you have not yet started the verification process.

4. To sort identities by verification status, choose the **Status** column.
5. To view an identity's details page, select the identity that you want to view.

Delete an identity using the SES console

You can use the Amazon SES console to remove a domain or email address identity from your account in the selected AWS Region.

To remove a domain or email address identity

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the console, use the Region selector to choose the AWS Region from which you want to delete one or more identities.
3. In the navigation pane, under **Configuration**, choose **Verified identities**.

   The **Loaded identities** table displays a list of both domain and email address identities.
4. In the **Identity** column, select the identity that you want to delete. You can delete multiple identities by checking the box next to each identity that you want to delete.
5. Choose **Delete**.

Edit an identity using the SES console

You can use the Amazon SES console to edit a domain or email address identity in your account in the selected AWS Region.

To edit a domain or email address identity

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the console, use the Region selector to choose the AWS Region from which you want to edit one or more identities.
3. In the navigation pane, under **Configuration**, choose **Verified identities**.

   The **Loaded identities** table displays a list of both domain and email address identities.

4. In the **Identity** column, select the identity that you want to edit (by clicking directly on the identity name as opposed to selecting its checkbox).
5. On the identity's detail page, select the tab containing the categories you'd like to edit.
6. In any of the selected tab's categorical containers, choose the **Edit** button of the attribute you wish to edit, make your changes, then choose **Save changes**.

    a. If you wish to edit attributes under the **Authentication** tab and your domain identity is hosted in Amazon Route 53, and you haven't already published its DNS records, there will be a **Publish DNS records to Route53** button (next to the **Edit** button) in either or both of the **DomainKeys Identified Mail (DKIM)** or **Custom MAIL FROM domain** containers.

    > **Note**
    >
    > The **Authentication** tab is only present when your account has a verified domain or an email address that uses a verified domain in your account.

    b. You can publish the DNS records directly from the **Publish DNS records to Route53** button - just click it, a confirmation banner will be displayed, and the **Publish DNS records to Route53** button will no longer be visible for the respective container.
7. Repeat steps 5 & 6 for each attribute of the identity you'd like to edit.

Edit an identity to use a default configuration set using the SES API

You can use the PutEmailIdentityConfigurationSetAttributes operation to add or remove a default configuration set from an existing email identity.

> **Note**
>
> Before you complete the procedure in this section, you have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

To add a default configuration set using the AWS CLI

- At the command line, enter the following command to use the PutEmailIdentityConfigurationSetAttributes operation.

```
aws sesv2 put-email-identity-configuration-set-attributes --email-identity ADDRESS-OR-DOMAIN --configuration-set-name CONFIG-SET
```

In the preceding commands, replace *ADDRESS-OR-DOMAIN* with the email identity that you want to verify. Replace *CONFIG-SET* with the name of the configuration set you wish to set as the identity's default configuration set.

If the command executes successfully, it exits without providing any output.

To remove a default configuration set using the AWS CLI

- At the command line, enter the following command to use the PutEmailIdentityConfigurationSetAttributes operation.

```
aws sesv2 put-email-identity-configuration-set-attributes --email-identity ADDRESS-OR-DOMAIN
```

In the preceding commands, replace *ADDRESS-OR-DOMAIN* with the email identity that you want to verify.

If the command executes successfully, it exits without providing any output.

Retrieve the default configuration set used by the identity using the SES API

You can use the GetEmailIdentity operation to return the default configuration set for an email identity, if applicable.

> **Note**
>
> Before you complete the procedure in this section, you have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

To return a default configuration set using the AWS CLI

- At the command line, enter the following command to use the GetEmailIdentity operation.

```
aws sesv2 get-email-identity --email-identity ADDRESS-OR-DOMAIN
```

In the preceding commands, replace *ADDRESS-OR-DOMAIN* with with email identity for which you wish to know the default configuration set, if any.

If the command executes successfully, it provides a JSON object with the email identity details.

Override the current default configuration set used by the identity using the SES API

You can use the SendEmail operation to send email with a different configuration set. If you do, the configuration set that you specify overrides the default configuration set for the identity.

> **Note**
>
> Before you complete the procedure in this section, you have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

To override a default configuration set using the AWS CLI

- At the command line, enter the following command to use the SendEmail operation.

```
aws sesv2 send-email --destination file://DESTINATION-JSON --content file://CONTENT-
JSON --from-email-address ADDRESS-OR-DOMAIN --configuration-set-name CONFIG-SET
```

In the preceding commands, replace *DESTINATION-JSON* with your destination JSON file, *CONTENT-JSON* with your content JSON file, *ADDRESS-OR-DOMAIN* with your FROM email address, and *CONFIG-SET* with the name of the configuration set you wish to use instead of the default configuration set for the identity.

If the command executes successfully, it outputs a `MessageId`.

Configuring identities in Amazon SES

Amazon Simple Email Service (Amazon SES) uses the Simple Mail Transfer Protocol (SMTP) to send email. Because SMTP doesn't provide any authentication by itself, spammers can send email

messages that claim to originate from someone else, while hiding their true origin. By falsifying email headers and spoofing source IP addresses, spammers can mislead recipients into believing that the email messages that they are receiving are authentic.

Most ISPs that forward email traffic take measures to evaluate whether email is legitimate. One such measure that ISPs take is to determine whether an email is *authenticated*. Authentication requires senders to verify that they're the owner of the account that they are sending from. In some cases, ISPs refuse to forward email that is not authenticated. To ensure optimal deliverability, we recommend that you authenticate your emails.

The following sections describe two authentication mechanisms ISPs use—Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM)—and provide instructions for how to use these standards with Amazon SES.

- To learn about SPF, which provides a way to trace an email message back to the system from which it was sent, see Authenticating Email with SPF in Amazon SES.
- To learn about DKIM, a standard that allows you to sign your email messages to show ISPs that your messages are legitimate and have not been modified in transit, see Authenticating Email with DKIM in Amazon SES.
- To learn how to comply with Domain-based Message Authentication, Reporting and Conformance (DMARC), which relies on SPF and DKIM, see Complying with DMARC authentication protocol in Amazon SES.

Email authentication methods

Amazon Simple Email Service (Amazon SES) uses the Simple Mail Transfer Protocol (SMTP) to send email. Because SMTP does not provide any authentication by itself, spammers can send email messages that claim to originate from someone else, while hiding their true origin. By falsifying email headers and spoofing source IP addresses, spammers can mislead recipients into believing that the email messages that they are receiving are authentic.

Most ISPs that forward email traffic take measures to evaluate whether email is legitimate. One such measure that ISPs take is to determine whether an email is authenticated. Authentication requires senders to verify that they are the owner of the account that they are sending from. In some cases, ISPs refuse to forward email that is not authenticated. To ensure optimal deliverability, we recommend that you authenticate your emails.

Contents

- Authenticating Email with DKIM in Amazon SES
- Authenticating email with S/MIME in Amazon SES
- Authenticating Email with SPF in Amazon SES
- Using a custom MAIL FROM domain
- Complying with DMARC authentication protocol in Amazon SES
- Using BIMI in Amazon SES

Authenticating Email with DKIM in Amazon SES

DomainKeys Identified Mail (*DKIM*) is an email security standard designed to make sure that an email that claims to have come from a specific domain was indeed authorized by the owner of that domain. It uses public-key cryptography to sign an email with a private key. Recipient servers can then use a public key published to a domain's DNS to verify that parts of the email have not been modified during the transit.

DKIM signatures are optional. You might decide to sign your email using a DKIM signature to enhance deliverability with DKIM-compliant email providers. Amazon SES provides three options for signing your messages using a DKIM signature:

- **Easy DKIM**: SES generates a public-private key pair and automatically adds a DKIM signature to every message that you send from that identity, see Easy DKIM in Amazon SES.
- **Deterministic Easy DKIM (DEED)**: Enables you to maintain consistent DKIM signing across multiple AWS Regions by creating replica identities that automatically inherit the DKIM signing attributes as a parent identity that is using Easy DKIM, see Using Deterministic Easy DKIM (DEED) in Amazon SES.
- **BYODKIM (Bring Your Own DKIM)**: You provide your own public-private key pair and SES adds a DKIM signature to every message that you send from that identity, see Provide your own DKIM authentication token (BYODKIM) in Amazon SES.
- **Manually add DKIM signature**: You add your own DKIM signature to email that you send using the `SendRawEmail` API, see Manual DKIM signing in Amazon SES.

DKIM signing key length

Since many DNS providers now fully support DKIM 2048 bit RSA encryption, Amazon SES also supports DKIM 2048 to allow more secure authentication of emails and therefore uses it as the default key length when you configure Easy DKIM either from the API or the console. 2048 bit keys

can be setup and used in Bring Your Own DKIM (BYODKIM) as well, where your signing key length must be at least 1024 bits and no more than 2048 bits.

For the sake of security as well as your email's deliverability, when configured with Easy DKIM, you have the choice to use either 1024 and 2048 bit key lengths along with the flexibility of flipping back to 1024 in the event there are problems caused by any DNS providers who still don't support 2048. *When you create a new identity, it will be created with DKIM 2048 by default unless you specify 1024.*

To preserve the deliverability of in transit emails, there are restrictions on the frequency at which you can change your DKIM key length. Restrictions include:

- Not being able to switch to the same key length as is already configured.
- Not being able to switch to different key length more than once in a 24 hour period (unless it's the first downgrade to 1024 in that period).

When your email is in transit, DNS is using your public key to authenticate your email; therefore, if you change keys too quickly or frequently, DNS may not be able to DKIM authenticate your email as the former key may already be invalidated, thus, these restrictions safeguard against that.

DKIM considerations

When you use DKIM to authenticate your email, the following rules apply:

- You only need to set up DKIM for the domain that you use in your "From" address. You don't need to set up DKIM for domains that you use in "Return-Path" or "Reply-to" addresses.
- Amazon SES is available in several AWS Regions. If you use more than one AWS Region to send email, you have to complete the DKIM setup process in each of those Regions to ensure that all of your email is DKIM-signed.
- Because DKIM properties are inherited from the parent domain, when you verify a domain with DKIM authentication:
  - DKIM authentication will also apply to all subdomains of that domain.
    - DKIM settings for a subdomain can override the settings for the parent domain by disabling the inheritance if you don't want the subdomain to use DKIM authentication, as well as the ability to re-enable later.
  - DKIM authentication will also apply to all email sent from an email identity that references the DKIM verified domain in its address.

- DKIM settings for an email address can override the settings for the subdomain (if applicable) and the parent domain by disabling the inheritance if you want to send mail without DKIM authentication, as well as the ability to re-enable later.

Understanding inherited DKIM signing properties

It's important to first understand that an email address identity inherits its DKIM signing properties from its parent domain if that domain was configured with DKIM, regardless of whether Easy DKIM or BYODKIM was used. Therefore, disabling or enabling DKIM signing on the email address identity, is in effect, overriding the domain's DKIM signing properties based on these key facts:

- If you already set up DKIM for the domain that an email address belongs to, you do not need to enable DKIM signing for the email address identity as well.
  - When you set up DKIM for a domain, Amazon SES automatically authenticates every email from every address on that domain through the inherited DKIM properties from the parent domain.
- DKIM settings for a specific email address identity *automatically override the settings of the parent domain or subdomain (if applicable)* that the address belongs to.

Since the email address identity's DKIM signing properties are inherited from the parent domain, if you're planning on overriding these properties, you must keep in mind the hierarchical rules of overriding as explained in the table below.

Parent domain does not have DKIM signing enabled	Parent domain has DKIM signing enabled
You cannot enable DKIM signing on the email address identity.	You can disable DKIM signing on the email address identity.
	You can re-enable DKIM signing on the email address identity.

It’s generally never recommended to disable your DKIM signing as it risks tarnishing your sender reputation, and it increases the risk of having your sent mail go to junk or spam folders or having your domain spoofed.

However, the capability exists to override the domain inherited DKIM signing properties on an email address identity for any particular use case or outlying business decision that you might have to either permanently or temporarily disable DKIM signing, or to re-enable it at a later time. See the section called “Overriding DKIM signing on email address”.

Easy DKIM in Amazon SES

When you set up Easy DKIM for a domain identity, Amazon SES automatically adds a 2048-bit DKIM key to every email that you send from that identity. You can configure Easy DKIM by using the Amazon SES console, or by using the API.

> **Note**
>
> To set up Easy DKIM, you have to modify the DNS settings for your domain. If you use Route 53 as your DNS provider, Amazon SES can automatically create the appropriate records for you. If you use another DNS provider, see your provider's documentation to learn more about changing the DNS settings for your domain.

> **Warning**
>
> If you currently have BYODKIM enabled and are transitioning over to Easy DKIM, be aware that Amazon SES will not use BYODKIM to sign your emails while Easy DKIM is being set up and your DKIM status is in a pending state. Between the moment you make the call to enable Easy DKIM (either through the API or console) and the moment when SES can confirm your DNS configuration, your emails may be sent by SES without a DKIM signature. Therefore, it is advised to use an intermediary step to migrate from one DKIM signing method to the other (e.g., using a subdomain of your domain with BYODKIM enabled and then deleting it once Easy DKIM verification has passed), or perform this activity during your application's downtime, if any.

Setting up Easy DKIM for a verified domain identity

The procedure in this section is streamlined to just show the steps necessary to configure Easy DKIM on a domain identity that you've already created. If you haven't yet created a domain identity or you want to see all available options for customizing a domain identity, such as using a default configuration set, custom MAIL FROM domain, and tags, see the section called “Creating a domain identity”.

Part of creating an Easy DKIM domain identity is configuring its DKIM-based verification where you will have the choice to either accept the Amazon SES default of 2048 bits, or to override the default by selecting 1024 bits. See the section called “DKIM signing key length” to learn more about DKIM signing key lengths and how to change them.

To set up Easy DKIM for a domain

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the list of identities, choose an identity where the **Identity type** is *Domain*.

   > **Note**
   >
   > If you need to create or verify a domain, see Creating a domain identity.

4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** container, choose **Edit**.
5. In the **Advanced DKIM settings** container, choose the **Easy DKIM** button in the **Identity type** field.
6. In the **DKIM signing key length** field, choose either **RSA_2048_BIT** or **RSA_1024_BIT**.
7. In the **DKIM signatures** field, check the **Enabled** box.
8. Choose **Save changes**.
9. Now that you've configured your domain identity with Easy DKIM, you must complete the verification process with your DNS provider - proceed to the section called “Verifying a domain identity” and follow the DNS authentication procedures for Easy DKIM.

Change the Easy DKIM signing key length for an identity

The procedure in this section shows how you can easily change the Easy DKIM bits required for the signing algorithm. While a signing length of 2048 bits is always preferred for the enhanced security it provides, there may be situations that require you to use the 1024 bit length, such as having to use a DNS provider who only supports DKIM 1024.

To preserve the deliverability of in transit emails, there are restrictions on the frequency at which you can change or flip your DKIM key length.

When your email is in transit, DNS is using your public key to authenticate your email; therefore, if you change keys too quickly or frequently, DNS may not be able to DKIM authenticate your email as the former key may already be invalidated, thus, the following restrictions safeguard against that:

- You can't switch to the same key length as is already configured.
- You can't switch to a different key length more than once in a 24 hour period (unless it's the first downgrade to 1024 in that period).

In using the following procedures to change your key length, if you violate one of these restrictions, the console will return an error banner stating that *the input you provided is invalid* along with the reason of why it was invalid.

To change the DKIM signing key length bits

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity you want to change the DKIM signing key length for.
4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** container, choose **Edit**.
5. In the **Advanced DKIM settings** container, choose either **RSA_2048_BIT** or **RSA_1024_BIT** in the **DKIM signing key length** field.
6. Choose **Save changes**.

Using Deterministic Easy DKIM (DEED) in Amazon SES

Deterministic Easy DKIM (DEED) offers a solution for managing DKIM configurations across multiple AWS Regions. By simplifying DNS management and ensuring consistent DKIM signing, DEED helps you streamline your multi-region email sending operations while maintaining robust email authentication practices.

What is Deterministic Easy DKIM (DEED)?

Deterministic Easy DKIM (DEED) is a feature that generates consistent DKIM tokens across all AWS Regions based on a parent domain that is configured with Easy DKIM. This enables you to replicate

identities in different AWS Regions that automatically inherit and maintain the same DKIM signing configuration as a parent identity that is currently configured with Easy DKIM. With DEED, you only need to publish DNS records once for the parent identity, and replica identities will use the same DNS records to verify domain ownership and manage DKIM signing.

By simplifying DNS management and ensuring consistent DKIM signing, DEED helps you streamline your multi-region email sending operations while maintaining best email authentication practices.

Terminology used when talking about DEED:

- **Parent identity** – A verified identity configured with Easy DKIM that serves as the source for DKIM configuration for a replica identity.
- **Replica identity** – A copy of a parent identity that shares the same DNS setup and DKIM signing configuration.
- **Parent region** – The AWS Region where a parent identity is set up.
- **Replica region** – An AWS Region where a replica identity is set up.
- **DEED identity** – Any identity that is used as either a parent identity or a replica identity. (When a new identity is created, it is initially treated as a regular (non-DEED) identity. However, once a replica is created, the identity is then considered a DEED identity.)

Key benefits of using DEED include:

- **Simplified DNS management** – Publish DNS records only once for the parent identity.
- **Easier multi-region operations** – Simplify the process of expanding email sending operations to new regions.
- **Reduced administrative overhead** – Manage DKIM configurations centrally from the parent identity.

How Deterministic Easy DKIM (DEED) works

When you create a replica identity, Amazon SES automatically replicates the DKIM signing key from the parent identity to the replica identity. Any subsequent DKIM key rotations or key length changes made to the parent identity are automatically propagated to all replica identities.

The process involves the following workflow:

1. Create a parent identity in an AWS Region using Easy DKIM.

2. Set up the required DNS records for the parent identity.
3. Create replica identities in other AWS Regions, specifying the parent identity's domain name and DKIM signing region.
4. Amazon SES automatically replicates the parent's DKIM configuration to the replica identities.

Important considerations:

- You cannot create a replica of an identity that is already a replica.
- The parent identity must have Easy DKIM enabled—you cannot create replicas of BYODKIM or manually signed identities.
- Parent identities cannot be deleted until all replica identities are deleted.

Setting up a replica identity using DEED

This section will provide examples showing you how to create and verify a replica identity using DEED along with the necessary permissions required.

Topics

- Creating a replica identity
- Verifying replica identity configuration
- Required Permissions to use DEED

Creating a replica identity

To create replica identity:

1. In the AWS Region where you want to create a replica identity, open the SES console at https://console.aws.amazon.com/ses/.

   (In the SES console, replica identities are referred to as *Global identities*.)
2. In the navigation pane, choose **Identities**.
3. Choose **Create identity**.
4. Select **Domain** under **Identity type** and enter the domain name of an existing identity configured with Easy DKIM that you want to replicate and serve as the parent.
5. Expand **Advanced DKIM settings** and select **Deterministic Easy DKIM**.

6. From the **Parent region** dropdown menu, select a parent region where an Easy DKIM-signed identity with the same name as you entered for your Global (replica) identity resides. (Your replica region defaults to the region you signed into the SES console with.)
7. Ensure **DKIM signatures** is enabled.
8. (Optional) Add one or more **Tags** to your domain identity.
9. Review the configuration and choose **Create identity**.

Using the AWS CLI:

To create a replica identity based on a parent identity configured with Easy DKIM, you need to specify the parent's domain name, the region where you want to create the replica identity, and the parent's DKIM signing region as shown in this example:

```
aws sesv2 create-email-identity --email-identity example.com --region us-west-2 --dkim-signing-attributes '{"DomainSigningAttributesOrigin": "AWS_SES_US_EAST_1"}'
```

In the preceding example:

1. Replace *example.com* with the parent domain identity being replicated.
2. Replace *us-west-2* with the region where the replica domain identity will be created.
3. Replace *AWS_SES_US_EAST_1* with the parent's DKIM signing region that represents its Easy DKIM signing configuration that will be replicated to the replica identity.

> **Note**
>
> The AWS_SES_ prefix indicates that DKIM was configured for the parent identity by using Easy DKIM, and US_EAST_1 is the AWS Region where it was created.

Verifying replica identity configuration

After creating the replica identity, you can verify that it was configured correctly with the parent identity's DKIM signing configuration.

To verify a replica identity:

1. In the AWS Region where you created the replica identity, open the SES console at https://console.aws.amazon.com/ses/.

2. In the navigation pane, choose **Identities** and select the identity you want to verify from the **Identities** table.
3. Under the **Authentication** tab, the **DKIM configuration** field will indicate the status, and the **Parent region** field will indicate the region being used for the identity's DKIM signing configuration utilizing DEED.

Using the AWS CLI:

Use the `get-email-identity` command specifying the replica's domain name and region:

```
aws sesv2 get-email-identity --email-identity example.com --region us-west-2
```

The response will include the value of the parent region in the `SigningAttributesOrigin` parameter signifying that the replica identity has been successfully configured with the parent identity's DKIM signing configuration:

```
{
  "DkimAttributes": {
    "SigningAttributesOrigin": "AWS_SES_US_EAST_1"
  }
}
```

Required Permissions to use DEED

To use DEED, you need:

1. Standard permissions for creating email identities in the replica region.
2. Permission to replicate the DKIM signing key from the parent region.

Example IAM policy for DKIM replication

The following policy allows DKIM signing key replication from a parent identity to specified replica regions:

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDKIMReplication",
      "Effect": "Allow",
      "Action": "ses:ReplicateEmailIdentityDKIMSigningKey",
      "Resource": "arn:aws-eusc:ses:eusc-de-east-1:123456789124:identity/
example.com",
      "Condition": {
        "ForAllValues:StringEquals": {
           "ses:ReplicaRegion": ["eusc-de-east-1", "eusc-de-east-1"]
        }
      }
    }
  ]
}
```

Best practices

The following best practices are recommended:

- **Plan your parent and replica regions** – Give consideration to the parent region you choose, as it will be the source of truth for the DKIM configuration used in replica regions.
- **Use consistent IAM policies** – Ensure that your IAM policies allow for DKIM replication across all intended regions.
- **Keep parent identities active** – Remember that your replica identities inherit the DKIM signing configuration of the parent identity, because of this dependency, you cannot delete a parent identity until all replica identities are deleted.

Troubleshooting

If you encounter issues with DEED, consider the following:

- **Verification errors** – Ensure that you have the necessary permissions for DKIM replication.
- **Replication delays** – Allow some time for replication to complete, especially when creating new replica identities.
- **DNS issues** – Verify that the DNS records for the parent identity are correctly set up and propagated.

Provide your own DKIM authentication token (BYODKIM) in Amazon SES

As an alternative to using Easy DKIM, you can instead configure DKIM authentication by using your own public-private key pair. This process is known as *Bring Your Own DKIM* (*BYODKIM*).

With BYODKIM, you can use a single DNS record to configure DKIM authentication for your domains, as opposed to Easy DKIM, which requires you to publish three separate DNS records. Additionally, with BYODKIM you can rotate the DKIM keys for your domains as often as you want.

Topics in this section:

- Step 1: Create the key pair
- Step 2: Add the selector and public key to your DNS provider's domain configuration
- Step 3: Configure and verify a domain to use BYODKIM

> **Warning**
>
> If you currently have Easy DKIM enabled and are transitioning over to BYODKIM, be aware that Amazon SES will not use Easy DKIM to sign your emails while BYODKIM is being set up and your DKIM status is in a pending state. Between the moment you make the call to enable BYODKIM (either through the API or console) and the moment when SES can confirm your DNS configuration, your emails may be sent by SES without a DKIM signature. Therefore, it is advised to use an intermediary step to migrate from one DKIM signing method to the other (e.g., using a subdomain of your domain with Easy DKIM enabled and then deleting it once BYODKIM verification has passed), or perform this activity during your application's downtime, if any.

Step 1: Create the key pair

To use the Bring Your Own DKIM feature, you first have to create an RSA key pair.

The private key that you generate has to be in either PKCS #1 or PKCS #8 format, must use at least 1024-bit RSA encryption and up to 2048-bit, and be encoded using base64 (PEM) encoding. See the section called “DKIM signing key length” to learn more about DKIM signing key lengths and how to change them.

> **Note**
>
> You can use third-party applications and tools to generate RSA key pairs as long as the private key is generated with at least 1024-bit RSA encryption and up to 2048-bit, and is encoded using base64 (PEM) encoding.

In the following procedure, the example code which uses the `openssl genrsa` command that's built into most Linux, macOS, or Unix operating systems to create the key pair will automatically use base64 (PEM) encoding.

To create the key pair from the Linux, macOS, or Unix command line

1. At the command line, enter the following command to generate the private key replacing *nnnn* with a bit length of at least 1024 and up to 2048:

   ```
   openssl genrsa -f4 -out private.key nnnn
   ```

2. At the command line, enter the following command to generate the public key:

   ```
   openssl rsa -in private.key -outform PEM -pubout -out public.key
   ```

Step 2: Add the selector and public key to your DNS provider's domain configuration

Now that you've created a key pair, you have to add the public key as a TXT record to the DNS configuration for your domain.

To add the public key to the DNS configuration for your domain

1. Sign in to the management console for your DNS or hosting provider.
2. Add a new text record to the DNS configuration for your domain. The record should use the following format:

Name	Type	Value
selector._domainkey.*example.com*	TXT	p=*yourPublicKey*

In the preceding example, make the following changes:

- Replace *selector* with a unique name that identifies the key.

> **Note**
>
> A small number of DNS providers don't allow you to include underscores (_) in record names. However, the underscore in the DKIM record name is required. If your DNS provider doesn't allow you to enter an underscore in the record name, contact the provider's customer support team for assistance.

- Replace *example.com* with your domain.
- Replace *yourPublicKey* with the public key that you created earlier and include the p= prefix as shown in the *Value* column above.

> **Note**
>
> When you publish (add) your public key to your DNS provider, it must be formatted as follows:
>
> - You have to delete the first and last lines (`-----BEGIN PUBLIC KEY-----` and `-----END PUBLIC KEY-----`, respectively) of the generated public key. Additionally, you have to remove the line breaks in the generated public key. The resulting value is a string of characters with no spaces or line breaks.
> - You must include the p= prefix as shown in the *Value* column in the table above.

Different providers have different procedures for updating DNS records. The following table includes links to the documentation for a few widely used DNS providers. This list isn't exhaustive and doesn't signify endorsement; likewise, if your DNS provider isn't listed, it doesn't imply you can't use the domain with Amazon SES.

DNS/Hosting provider	Documentation link
Amazon Route 53	Editing Records in the *Amazon Route 53 Developer Guide*

DNS/Hosting provider	Documentation link
GoDaddy	Add a TXT record (external link)
DreamHost	How do I add custom DNS records? (external link)
Cloudflare	Managing DNS records in Cloudflare (external link)
HostGator	Manage DNS Records with HostGator/eNom (external link)
Namecheap	How do I add TXT/SPF/DKIM/DMARC records for my domain? (external link)
Names.co.uk	Changing your domains DNS Settings (external link)
Wix	Adding or Updating TXT Records in Your Wix Account (external link)

Step 3: Configure and verify a domain to use BYODKIM

You can set up BYODKIM for both new domains (that is, domains that you don't currently use to send email through Amazon SES) and existing domains (that is, domains that you've already set up to use with Amazon SES) by using either the console or AWS CLI. Before you use the AWS CLI procedures in this section, you first have to install and configure the AWS CLI. For more information, see the AWS Command Line Interface User Guide..

Option 1: Creating a new domain identity that uses BYODKIM

This section contains procedures for creating a new domain identity that uses BYODKIM. A new domain identity is a domain that you haven't previously set up to send email using Amazon SES.

If you want to configure an existing domain to use BYODKIM, complete the procedure in Option 2: Configuring an existing domain identity instead.

To create an identity using BYODKIM from the console

- Follow the procedures in Creating a domain identity, and when you get to Step 8, follow the BYODKIM specific instructions.

To create an identity using BYODKIM from the AWS CLI

To configure a new domain, use the `CreateEmailIdentity` operation in the Amazon SES API.

1. In a text editor, paste the following code:

```
{
    "EmailIdentity":"example.com",
    "DkimSigningAttributes":{
        "DomainSigningPrivateKey":"privateKey",
        "DomainSigningSelector":"selector"
    }
}
```

   In the preceding example, make the following changes:

   - Replace *`example.com`* with the domain that you want to create.
   - Replace *`privateKey`* with your private key.

     > **Note**
     >
     > You have to delete the first and last lines (`-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----`, respectively) of the generated private key. Additionally, you have to remove the line breaks in the generated private key. The resulting value is a string of characters with no spaces or line breaks.

   - Replace *`selector`* with the unique selector that you specified when you created the TXT record in the DNS configuration for your domain.

   When you finish, save the file as `create-identity.json`.

2. At the command line, enter the following command:

```
aws sesv2 create-email-identity --cli-input-json file://path/to/create-
identity.json
```

In the preceding command, replace *path/to/create-identity.json* with the complete path to the file that you created in the previous step.

Option 2: Configuring an existing domain identity

This section contains procedures for updating an existing domain identity to use BYODKIM. An existing domain identity is a domain that you have already set up to send email using Amazon SES.

To update a domain identity using BYODKIM from the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose an identity where the **Identity type** is *Domain*.

> **Note**
>
> If you need to create or verify a domain, see Creating a domain identity.

4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** pane, choose **Edit**.
5. In the **Advanced DKIM settings** pane, choose the **Provide DKIM authentication token (BYODKIM)** button in the **Identity type** field.
6. For **Private key**, paste the private key you generated earlier.

> **Note**
>
> You have to delete the first and last lines (`-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----`, respectively) of the generated private key. Additionally, you have to remove the line breaks in the generated private key. The resulting value is a string of characters with no spaces or line breaks.

7. For **Selector name**, enter the name of the selector that you specified in your domain's DNS settings.

8. In the **DKIM signatures** field, check the **Enabled** box.
9. Choose **Save changes**.

To update a domain identity using BYODKIM from the AWS CLI

To configure an existing domain, use the `PutEmailIdentityDkimSigningAttributes` operation in the Amazon SES API.

1. In a text editor, paste the following code:

```
{
    "SigningAttributes":{
        "DomainSigningPrivateKey":"privateKey",
        "DomainSigningSelector":"selector"
    },
    "SigningAttributesOrigin":"EXTERNAL"
}
```

   In the preceding example, make the following changes:

   - Replace *`privateKey`* with your private key.

     > **Note**
     >
     > You have to delete the first and last lines (`-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----`, respectively) of the generated private key. Additionally, you have to remove the line breaks in the generated private key. The resulting value is a string of characters with no spaces or line breaks.

   - Replace *`selector`* with the unique selector that you specified when you created the TXT record in the DNS configuration for your domain.

   When you finish, save the file as `update-identity.json`.

2. At the command line, enter the following command:

```
aws sesv2 put-email-identity-dkim-signing-attributes --email-identity example.com
 --cli-input-json file://path/to/update-identity.json
```

In the preceding command, make the following changes:

- Replace *path/to/update-identity.json* with the complete path to the file that you created in the previous step.
- Replace *example.com* with the domain that you want to update.

Verifying the DKIM status for a domain that uses BYODKIM

To verify the DKIM status of a domain from the console

After you configure a domain to use BYODKIM, you can use the SES console to verify that DKIM is properly configured.

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity whose DKIM status you want to verify.
4. It can take up to 72 hours for changes to DNS settings to propagate. As soon as Amazon SES detects all of the required DKIM records in your domain's DNS settings, the verification process is complete. If everything has been configured correctly, your domain's **DKIM configuration** field displays **Successful** in the **DomainKeys Identified Mail (DKIM)** pane, and the **Identity status** field displays **Verified** in the **Summary** pane.

To verify the DKIM status of a domain using the AWS CLI

After you configure a domain to use BYODKIM, you can use the GetEmailIdentity operation to verify that DKIM is properly configured.

- At the command line, enter the following command:

```
aws sesv2 get-email-identity --email-identity example.com
```

In the preceding command, replace *example.com* with your domain.

This command returns a JSON object that contains a section that resembles the following example.

```
{
    ...
    "DkimAttributes": {
        "SigningAttributesOrigin": "EXTERNAL",
        "SigningEnabled": true,
        "Status": "SUCCESS",
        "Tokens": [ ]
    },
    ...
}
```

If all of the following are true, BYODKIM is properly configured for the domain:

- The value of the `SigningAttributesOrigin` property is `EXTERNAL`.
- The value of `SigningEnabled` is `true`.
- The value of `Status` is SUCCESS.

Managing Easy DKIM and BYODKIM

You can manage the DKIM settings for your identities authenticated with either Easy DKIM or BYODKIM by using the web-based Amazon SES console, or by using the Amazon SES API. You can use either of these methods to obtain the DKIM records for an identity, or to enable or disable DKIM signing for an identity.

Obtaining DKIM Records for an identity

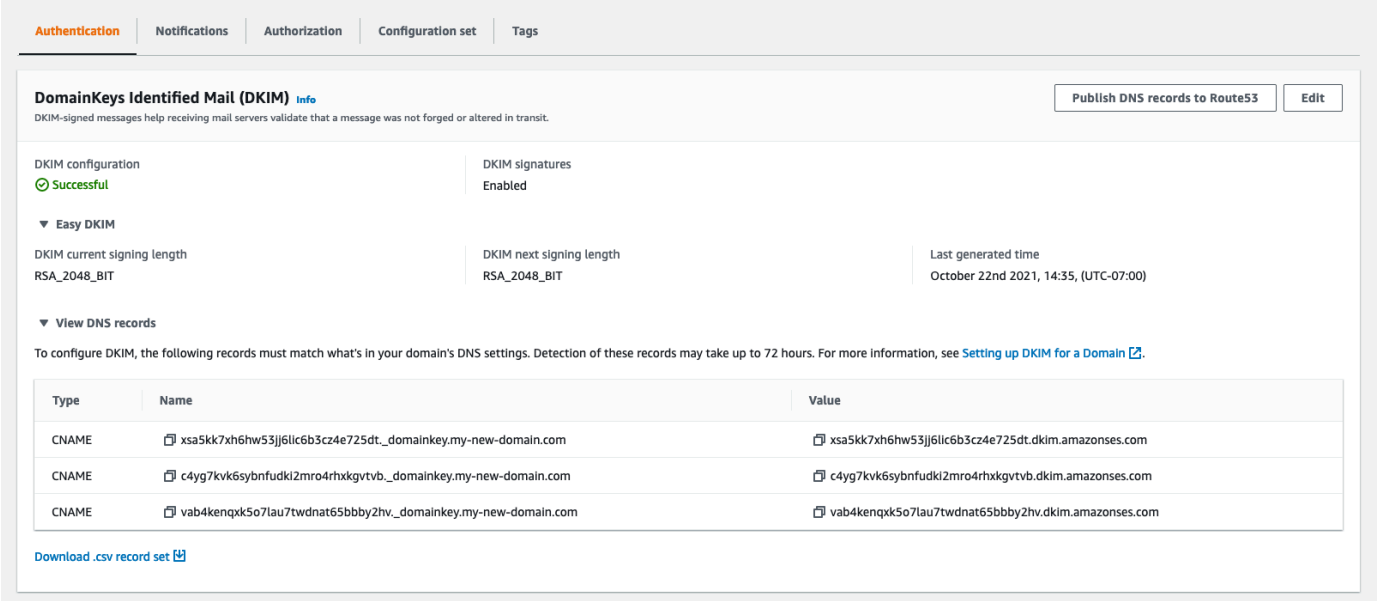
You can obtain the DKIM records for your domain or email address at any time by using the Amazon SES console.

To obtain the DKIM records for an identity by using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity for which you want to obtain DKIM records.
4. On the **Authentication** tab of the identity details page, expand **View DNS records**.

5. Copy either the three CNAME records if you used Easy DKIM, or the TXT record if you used BYODKIM, that appear in this section. Alternatively, you can choose **Download .csv record set** to save a copy of the records to your computer.

   The following image shows an example of the expanded **View DNS records** section revealing CNAME records associated with Easy DKIM.



You can also obtain the DKIM records for an identity by using the Amazon SES API v2. A common method of interacting with the API is to use the AWS CLI.

To obtain the DKIM records for an identity by using the AWS CLI

1. At the command line, enter the following command:

   ```
   aws sesv2 get-email-identity --email-identity "example.com"
   ```

   In the preceding example, replace *example.com* with the identity that you want to obtain DKIM records for. You can specify either an email address or a domain.

2. The output of this command contains a `DkimAttributes` object. The `Tokens` array provides the DKIM tokens, and the `SigningHostedZone` field provides the hosted zone to use in the CNAME record values, as shown in the following example:

   ```
   {
       "DkimAttributes": {
   ```

```
        "SigningEnabled": true,
        "Status": "SUCCESS",
        "Tokens": [
            "hirjd4exampled5477y22yd23ettobi",
            "v3rnz522czcl46quexamplek3efo5o6x",
            "y4examplexbhyhnsjcmtvzotfvqjmdqoj"
        ],
        "SigningAttributesOrigin": "AWS_SES",
        "SigningHostedZone": "dkim.us-west-2.amazonses.com"
    }
}
```

> **Note**
>
> The hosted zone returned in `SigningHostedZone` varies by AWS Region and can differ between identities. Always use the `SigningHostedZone` value returned for your identity to construct the CNAME records, rather than a hardcoded hosted zone.

You use the tokens together with the `SigningHostedZone` value to create the CNAME records that you add to the DNS settings for your domain. To create the CNAME records, use the following template:

```
token1._domainkey.example.com CNAME token1.SigningHostedZone
token2._domainkey.example.com CNAME token2.SigningHostedZone
token3._domainkey.example.com CNAME token3.SigningHostedZone
```

Replace each instance of *token1*, *token2*, and *token3* with the first, second, and third tokens from the `Tokens` array. Replace each instance of *SigningHostedZone* with the `SigningHostedZone` value returned for your identity.

For example, applying this template to the values shown in the preceding example produces the following records:

```
hirjd4exampled5477y22yd23ettobi._domainkey.example.com CNAME
 hirjd4exampled5477y22yd23ettobi.dkim.us-west-2.amazonses.com
v3rnz522czcl46quexamplek3efo5o6x._domainkey.example.com CNAME
 v3rnz522czcl46quexamplek3efo5o6x.dkim.us-west-2.amazonses.com
```

```
y4examplexbhyhnsjcmtvzotfvqjmdqoj._domainkey.example.com CNAME
 y4examplexbhyhnsjcmtvzotfvqjmdqoj.dkim.us-west-2.amazonses.com
```

Note

SES uses various DKIM hosted zones that might differ per AWS Region and email identity. Always use the `SigningHostedZone` value returned by the CreateEmailIdentity or GetEmailIdentity operation to construct your CNAME records.

Disabling Easy DKIM for an identity

You can quickly disable DKIM authentication for an identity by using the Amazon SES console.

To disable DKIM for an identity

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity for which you want to disable DKIM.
4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** container, choose **Edit**.
5. In **Advanced DKIM settings**, clear the **Enabled** box in the **DKIM signatures** field.

You can also disable DKIM for an identity by using the Amazon SES API. A common method of interacting with the API is to use the AWS CLI.

To disable DKIM for an identity by using the AWS CLI

- At the command line, type the following command:

```
aws ses set-identity-dkim-enabled --identity example.com --no-dkim-enabled
```

In the preceding example, replace *example.com* with the identity that you want to disable DKIM for. You can specify either an email address or a domain.

Enabling Easy DKIM for an identity

If you previously disabled DKIM for an identity, you can enable it again by using the Amazon SES console.

To enable DKIM for an identity

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity for which you want to enable DKIM.
4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** container, choose **Edit**.
5. In **Advanced DKIM settings**, check the **Enabled** box in the **DKIM signatures** field.

You can also enable DKIM for an identity by using the Amazon SES API. A common method of interacting with the API is to use the AWS CLI.

To enable DKIM for an identity by using the AWS CLI

- At the command line, type the following command:

  ```
  aws ses set-identity-dkim-enabled --identity example.com --dkim-enabled
  ```

  In the preceding example, replace *example.com* with the identity that you want to enable DKIM for. You can specify either an email address or a domain.

Overriding inherited DKIM signing on an email address identity

In this section you'll learn how to override (disable or enable) the inherited DKIM signing properties from the parent domain on a specific email address identity that you've already verified with Amazon SES. You can only do this for email address identities that belong to domains you already own because DNS settings are configured at the domain level.

> **Important**
>
> You can't disable/enable DKIM signing for email address identities...

- on domains that you don't own. For example, you can't toggle DKIM signing for a *gmail.com* or *hotmail.com* address,
- on domains that you own, but have not yet been verified in Amazon SES,
- on domains that you own, but have not enabled DKIM signing on the domain.

This section contains the following topics:

- Understanding inherited DKIM signing properties
- Overriding DKIM signing on an email address identity (console)
- Overriding DKIM signing on an email address identity (AWS CLI)

Understanding inherited DKIM signing properties

It's important to first understand that an email address identity inherits its DKIM signing properties from its parent domain if that domain was configured with DKIM, regardless of whether Easy DKIM or BYODKIM was used. Therefore, disabling or enabling DKIM signing on the email address identity, is in effect, overriding the domain's DKIM signing properties based on these key facts:

- If you already set up DKIM for the domain that an email address belongs to, you do not need to enable DKIM signing for the email address identity as well.
  - When you set up DKIM for a domain, Amazon SES automatically authenticates every email from every address on that domain through the inherited DKIM properties from the parent domain.
- DKIM settings for a specific email address identity *automatically override the settings of the parent domain or subdomain (if applicable)* that the address belongs to.

Since the email address identity's DKIM signing properties are inherited from the parent domain, if you're planning on overriding these properties, you must keep in mind the hierarchical rules of overriding as explained in the table below.

Parent domain does not have DKIM signing enabled	Parent domain has DKIM signing enabled
You cannot enable DKIM signing on the email address identity.	You can disable DKIM signing on the email address identity.
	You can re-enable DKIM signing on the email address identity.

It's generally never recommended to disable your DKIM signing as it risks tarnishing your sender reputation, and it increases the risk of having your sent mail go to junk or spam folders or having your domain spoofed.

However, the capability exists to override the domain inherited DKIM signing properties on an email address identity for any particular use case or outlying business decision that you might have to either permanently or temporarily disable DKIM signing, or to re-enable it at a later time.

Overriding DKIM signing on an email address identity (console)

The following SES console procedure explains how to override (disable or enable) the inherited DKIM signing properties from the parent domain on a specific email address identity that you've already verified with Amazon SES.

To disable/enable DKIM signing for an email address identity using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose an identity where the **Identity type** is *Email address* and belongs to one of your verified domains.
4. Under the **Authentication** tab, in the **DomainKeys Identified Mail (DKIM)** container, choose **Edit**.

> **Note**
>
> The **Authentication** tab is only present if the selected email address identity belongs to a domain that has already been verified by SES. If you haven't verified your domain yet, see Creating a domain identity.

5. Under **Advanced DKIM settings**, in the **DKIM signatures** field, clear the **Enabled** checkbox to disable DKIM signing, or select it to re-enable DKIM signing (if it had been overridden previously).
6. Choose **Save changes**.

Overriding DKIM signing on an email address identity (AWS CLI)

The following example uses the AWS CLI with a SES API command and parameters that will override (disable or enable) the inherited DKIM signing properties from the parent domain on a specific email address identity that you've already verified with SES.

To disable/enable DKIM signing for an email address identity using the AWS CLI

- Assuming you own the *example.com* domain, and you want to disable DKIM signing for one of the domain's email addresses, at the command line, type the following command:

```
aws sesv2 put-email-identity-dkim-attributes --email-identity marketing@example.com
 --no-signing-enabled
```

  a. Replace *marketing@example.com* with the email address identity that you want to disable DKIM signing for.

  b. `--no-signing-enabled` will disable DKIM signing. To re-enable DKIM signing, use `--signing-enabled`.

Manual DKIM signing in Amazon SES

As an alternative to using Easy DKIM, you can instead manually add DKIM signatures to your messages, and then send those messages using Amazon SES. If you choose to manually sign your messages, you first have to create a DKIM signature. After you create the message and the DKIM signature, you can use the SendRawEmail API to send it.

If you decide to manually sign your email, consider the following factors:

- Every message that you send by using Amazon SES contains a DKIM header that references a signing domain of *amazonses.com* (that is, it contains the following string: `d=amazonses.com`). Therefore, if you manually sign your messages, your messages will include *two* DKIM headers: one for your domain, and the one that Amazon SES automatically creates for *amazonses.com*.
- Amazon SES doesn't validate DKIM signatures that you manually add to your messages. If there are errors with the DKIM signature in a message, it might be rejected by email providers.
- When you sign your messages, you should use a bit length of at least 1024 bits.
- Don't sign the following fields: Message-ID, Date, Return-Path, Bounces-To.

> **Note**
>
> If you use an email client to send email using the Amazon SES SMTP interface, your client might automatically perform DKIM signing of your messages. Some clients might sign some of these fields. For information about which fields are signed by default, see the documentation for your email client.

Authenticating email with S/MIME in Amazon SES

Secure/Multipurpose Internet Mail Extensions (*S/MIME*) is a standard that uses a certificate-based digital signature. It proves that a message was sent by the holder of the From address and that its content was not altered in transit. For more information about the S/MIME message format, see RFC 8551: S/MIME Message Specification on the RFC Editor website.

S/MIME signing proves message authenticity and integrity at the individual sender level. Recipients can be confident that the message content has not been tampered with.

With Amazon SES, you can automatically apply an S/MIME signature without signing messages before submitting them. You manage the certificate in AWS Certificate Manager (ACM). Amazon SES is one of the services that integrate with ACM. For more information about services that integrate with ACM, see Services integrated with ACM.

> **S/MIME and DKIM**
>
> S/MIME and DKIM operate at different levels and are complementary. DKIM proves that the sending domain authorized the message using keys published in DNS. S/MIME uses

a certificate to assert a chain of trust back to a certificate authority. This proves that the specific email-address holder sent the message and that the content is intact. S/MIME is not a replacement for DKIM.

Understanding the S/MIME signature

Amazon SES signs an outbound message when you send it through a configuration set that has S/MIME signing enabled. The resolved email identity must have an `ACTIVE` certificate for the From address. Amazon SES signs the message server-side at send time.

Amazon SES produces a detached signature using the `multipart/signed` format (clear-signing), as defined in RFC 1847 on the RFC Editor website. The original message remains readable by recipients whose mail clients do not support S/MIME. Amazon SES sends the signed content as the first MIME part. It carries the signature separately as a second part of media type `application/pkcs7-signature` (typically named `smime.p7s`). Amazon SES embeds your signing certificate and its certificate chain in the signature (a Cryptographic Message Syntax (CMS) SignedData object). Recipients can then build the chain of trust back to the issuing certificate authority and verify message integrity and origin.

The signed message uses standard S/MIME headers. These include a `Content-Type: multipart/signed` header with a `protocol` parameter of `"application/pkcs7-signature"` and a `micalg` parameter that names the digest algorithm (for example, `sha-256`). The following example shows the top-level Content-Type header of a signed message:

```
Content-Type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha-256;
    boundary="----=_SmimeBoundary"
```

S/MIME certificate considerations

For a certificate to be usable for S/MIME signing in Amazon SES, it must meet all of the following requirements:

- The certificate must be in ACM in the same AWS Region where you send email.
- The email identity must be verified. Either domain verification (TXT record) or email-address verification is acceptable.

- The certificate's Subject Alternative Name (SAN) must include an RFC822Name email address that matches the From address. For a domain identity, this means an address under that domain or a subdomain. For an email-address identity, this means an exact match.
- The certificate key algorithm must use one of: RSA 2048, RSA 3072, RSA 4096, EC P-256, EC P-384, or EC P-521 (per RFC 8550 on the RFC Editor website).
- The certificate must be currently valid (not expired).

> ⚠ **Certificate authority trust requirement**
>
> Recipients' mail clients must trust the certificate authority (CA) that issued your certificate to validate the signature. Certificates from a public CA are trusted by most mail clients and ISPs by default. If you use a private CA, recipients must have access to and trust your root CA. ISPs and mail clients do not trust private CAs by default.

For more information about certificate standards, see RFC 5280: X.509 Certificate and CRL Profile and RFC 8550: S/MIME Certificate Handling on the RFC Editor website.

Topics

- Attaching a certificate to an email identity
- Enabling S/MIME signing on a configuration set

Attaching a certificate to an email identity

Before you can attach a certificate to an email identity, you must meet the following prerequisites:

- A verified email identity.
- An S/MIME certificate imported into ACM that meets the requirements described in S/MIME certificate considerations.

Importing your certificate into ACM

You import your S/MIME certificate, private key, and certificate chain into ACM. After import, you can select the certificate in the Amazon SES console. For instructions on importing a certificate, see Importing certificates into ACM.

Attaching a certificate

When you attach a certificate, you associate an ACM certificate with a From address under an email identity. The From address must align with the Subject Alternative Name (SAN) in the certificate. This comparison is case-sensitive. The From address must also belong to the email identity.

You can attach a certificate using either the Amazon SES console or the AWS CLI.

Attach a certificate using the console

To attach a certificate to an email identity

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the list of identities, choose the verified identity that you want to configure.
   - If the **Status** is *Unverified*, complete the procedures in Creating and verifying identities in Amazon SES to verify the identity.
4. On the **Authentication** tab of the identity details page, in the **Email certificates** container, choose **Attach certificate**.
5. In the **Email certificate details** container, enter the From address that this certificate is for.
6. Select a certificate from the ACM certificate list. The list shows certificates that are compatible with this email identity and From address.
7. Choose **Save changes**.

Attach a certificate using the AWS CLI

The following example attaches a certificate to a domain email identity for a specific sender address. In the AWS CLI, use the `associate-email-identity-certificate` command.

```
aws sesv2 associate-email-identity-certificate \
    --email-identity example.com \
    --from-address sender@example.com \
    --certificate-arn arn:aws:acm:us-east-1:111122223333:certificate/abcd1234-
abcd-1234-abcd-abcd12345678
```

For a domain email identity, the `--from-address` parameter is required and must match an email address in the certificate SAN. For an email-address identity, you can omit `--from-address`.

To view attached certificates and their status, use the following command:

```
aws sesv2 list-email-identity-certificates \
    --email-identity example.com
```

For more information about the associate-email-identity-certificate command, see associate-email-identity-certificate in the AWS CLI Command Reference.

Monitoring certificate status changes with Amazon EventBridge

Amazon SES emits an event to Amazon EventBridge when a certificate association status changes to `ACTIVE`, `DEPROVISIONING`, or `FAILED`. You can use these events to automate responses, for example alerting when a certificate enters `FAILED` status.

Removing a certificate

To remove a certificate using the console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Open the **Email certificates** table on the email identity's **Authentication** tab.
3. Select the certificate.
4. Choose **Delete**.

To remove a certificate using the AWS CLI, run the following command:

```
aws sesv2 disassociate-email-identity-certificate \
    --email-identity example.com \
    --from-address sender@example.com
```

Removing the certificate from the email identity does not delete it from ACM. Amazon SES stops signing messages from that From address after you remove the certificate.

> ⚠ **One certificate per From address**
>
> You cannot attach more than one certificate to a From address. To replace a certificate, you must first remove the existing one, unless it is already `DEPROVISIONING`.

Identity deletion restriction

You cannot delete an email identity while it still has attached certificates.

Enabling S/MIME signing on a configuration set

You control S/MIME signing per configuration set. When you enable S/MIME signing on a configuration set, Amazon SES signs messages sent with that configuration set. Amazon SES signs the message only if the resolved email identity has an `ACTIVE` certificate attached for the From address.

Enable S/MIME signing using the console

To enable S/MIME signing on a configuration set

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, under **Configuration**, choose **Configuration sets**.
3. Choose the configuration set you want to configure.
4. On the **Overview** tab, in the **General details** section, choose **Edit**.
5. Under **Secure email**, select the **Enable S/MIME signing** checkbox.
6. Choose **Save changes**.

After you enable S/MIME signing, the **Secure Email** field in **General details** displays "S/MIME Signing Enabled".

Enable S/MIME signing using the AWS CLI

To enable S/MIME signing on a configuration set, use the `update-configuration-set` command to set the `MessageSecurityOptions` with an S/MIME signing scheme:

```
aws sesv2 update-configuration-set \
    --configuration-set-name my-config-set \
    --message-security-options '{"SigningScheme":{"Smime":{}}}'
```

The `SignatureFormat` defaults to `DETACHED`, which is the only supported value. You can also specify it explicitly:

```
aws sesv2 update-configuration-set \
    --configuration-set-name my-config-set \
    --message-security-options '{"SigningScheme":{"Smime":
{"SignatureFormat":"DETACHED"}}}'
```

To disable S/MIME signing, set the signing scheme back to the default:

```
aws sesv2 update-configuration-set \
    --configuration-set-name my-config-set \
    --message-security-options '{"SigningScheme":{"Default":{}}}'
```

Send-time behavior

When you enable both S/MIME and DKIM signing, Amazon SES applies the S/MIME signature before the DKIM signature. This ordering preserves the integrity of the DKIM signature because the S/MIME signature modifies the message body.

Amazon SES applies open and click tracking, template rendering, and subscription management link insertion before it signs the message. Because Amazon SES makes these modifications before signing, open and click tracking and subscription management remain compatible with S/MIME signing.

> ⚠ **Active certificate required at send time**
>
> If you send a message with S/MIME signing enabled on the configuration set, the resolved email identity must have an `ACTIVE` certificate for the From address. If no active certificate exists, Amazon SES rejects the message and returns an error.

To verify that S/MIME signing is working, you can use the **Send test email** action on the email identity detail page. When the configuration set used has S/MIME signing enabled, the test message is S/MIME signed.

Authenticating Email with SPF in Amazon SES

Sender Policy Framework (SPF) is an email validation standard that's designed to prevent email spoofing. Domain owners use SPF to tell email providers which servers are allowed to send email from their domains. SPF is defined in RFC 7208.

Messages that you send through Amazon SES automatically use a subdomain of `amazonses.com` as the default MAIL FROM domain. SPF authentication successfully validates these messages because the default MAIL FROM domain matches the application that sent the email—in this case, SES. Therefore, in SES, SPF is implicitly set up for you.

However, if you don't want to use the SES default MAIL FROM domain, and would rather use a subdomain of a domain that you own, this is referred to in SES as using a *custom* MAIL FROM domain. To do this, it requires you to publish your own SPF record for your custom MAIL FROM domain. In addition, SES also requires you to set up an MX record so that your custom MAIL FROM domain can receive the bounce and complaint notifications that email providers send you.

Learn how to set up SPF authentication

Instructions are given for configuring your domain with SPF and how to publish the MX and SPF (type TXT) records in the section called “Using a custom MAIL FROM domain”.

Using a custom MAIL FROM domain

When an email is sent, it has two addresses that indicate its source: a From address that's displayed to the message recipient, and a MAIL FROM address that indicates where the message originated. The MAIL FROM address is sometimes called the *envelope sender*, *envelope from*, *bounce address*, or *Return Path* address. Mail servers use the MAIL FROM address to return bounce messages and other error notifications. The MAIL FROM address is usually only viewable by recipients if they view the source code for the message.

Amazon SES sets the MAIL FROM domain for the messages that you send to a default value unless you specify your own (custom) domain. This section discusses the benefits of setting up a custom MAIL FROM domain, and includes setup procedures.

Why use a custom MAIL FROM domain?

Messages that you send through Amazon SES automatically use a subdomain of `amazonses.com` as the default MAIL FROM domain. Sender Policy Framework (SPF) authentication successfully validates these messages because the default MAIL FROM domain matches the application that sent the email—in this case, SES.

If you don't want to use the SES default MAIL FROM domain, and would rather use a subdomain of a domain that you own, this is referred to in SES as using a *custom* MAIL FROM domain. To do this, it requires you to publish your own SPF record for your custom MAIL FROM domain. In addition,

SES also requires you to set up an MX record so that your domain can receive the bounce and complaint notifications that email providers send you.

By using a custom MAIL FROM domain, you have the flexibility to use SPF, DKIM, or both to achieve Domain-based Message Authentication, Reporting and Conformance (DMARC) validation. DMARC enables a sender's domain to indicate that emails sent from the domain are protected by one or more authentication systems. There are two ways to achieve DMARC validation: the section called "Complying with DMARC through SPF" and the section called "Complying with DMARC through DKIM".

Choosing a custom MAIL FROM domain

In the following, the term *MAIL FROM domain* always refers to a subdomain of a domain that you own - this subdomain that you use for your custom MAIL FROM domain must not be used for anything else and meet the following requirements:

- The MAIL FROM domain has to be a subdomain of the parent domain of a verified identity (email address or domain).
- The MAIL FROM domain shouldn't be a subdomain that you also use to send email from.
- The MAIL FROM domain shouldn't be a subdomain that you use to receive email.

Using SPF with your custom MAIL FROM domain

Sender Policy Framework (SPF) is an email validation standard that's designed to prevent email spoofing. You can configure your custom MAIL FROM domain with SPF to tell email providers which servers are allowed to send email from your custom MAIL FROM domain. SPF is defined in RFC 7208.

To set up SPF, you publish a TXT record to the DNS configuration for your custom MAIL FROM domain. This record contains a list of the servers that you authorize to send email from using your custom MAIL FROM domain. When an email provider receives a message from your custom MAIL FROM domain, it checks the DNS records for that domain to make sure that the email was sent from an authorized server.

If you want to use this SPF record as a way to comply with DMARC, the domain in the From address must match the MAIL FROM domain. See the section called "Complying with DMARC through SPF".

The next section, the section called "Configuring your custom MAIL FROM domain", explains how to set up SPF for your custom MAIL FROM domain.

Configuring your custom MAIL FROM domain

The process of setting up a custom MAIL FROM domain requires you to add records to the DNS configuration for the domain. SES requires you to publish an MX record so that your domain can receive the bounce and complaint notifications that email providers send you. You also have to publish an SPF (type TXT) record in order to prove that Amazon SES is authorized to send email from your domain.

You can set up a custom MAIL FROM domain for an entire domain or subdomain, as well as for individual email addresses. The following procedures show how to use the Amazon SES console to configure a custom MAIL FROM domain. You can also configure a custom MAIL FROM domain using the SetIdentityMailFromDomain API operation.

Setting up a custom MAIL FROM domain for a verified domain

These procedures show you how to configure a custom MAIL FROM domain for an entire domain or subdomain so that all messages sent from addresses on that domain will use the this custom MAIL FROM domain.

To configure a verified domain to use a specified custom MAIL FROM domain

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, under **Configuration**, choose **Identities**.
3. In the list of identities, choose the identity you want to configure where the **Identity type** is **Domain** and **Status** is *Verified*.

    - If the **Status** is *Unverified*, complete the procedures at Verifying a DKIM domain identity with your DNS provider to verify the email address's domain.
4. At the bottom of the screen in the **Custom MAIL FROM domain** pane, choose **Edit** .
5. In the **General details** pane, do the following:

    a. Select the **Use a custom MAIL FROM domain** checkbox.

    b. For **MAIL FROM domain**, enter the subdomain that you want to use as the MAIL FROM domain.

    c. For **Behavior on MX failure**, choose one of the following options:

    - **Use default MAIL FROM domain** – If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES uses a subdomain of `amazonses.com`. The subdomain varies based on the AWS Region that you use Amazon SES in.

- **Reject message** – If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES returns a `MailFromDomainNotVerified` error. Emails that you attempt to send from this domain are automatically rejected.

d. Choose **Save changes** - you'll be returned to the previous screen.

6. Publish the MX and SPF (type TXT) records to the DNS server of the custom MAIL FROM domain:

   In the **Custom MAIL FROM domain** pane, the **Publish DNS records** table now displays the MX and SPF (type TXT) records in that you have to publish (add) to your domain's DNS configuration. These records use the formats shown in the following table.

Name	Type	Value
subdomain .*domain*.*com*	MX	
subdomain .*domain*.*com*	TXT	

   In the preceding records,

   - *subdomain*.*domain*.*com* will be populated with your MAIL FROM subdomain
   - *region* will be populated with the name of the AWS Region where you want to verify the MAIL FROM domain (such as `us-west-2`, `us-east-1`, or `eu-west-1`, etc.)
   - The number *10* listed along with the MX value is the preference order for the mail server and will need to be entered into a separate value field as specified by your DNS provider's GUI
   - The SPF's TXT record value usually has to include the quotation marks, but some DNS providers don't require them.

   From the **Publish DNS records** table, copy the MX and SPF (type TXT) records by choosing the copy icon next to each value and paste them into the corresponding fields in your DNS provider's GUI. Alternatively, you can choose **Download .csv record set** to save a copy of the records to your computer.

Important

- The specific procedures for publishing the MX and SPF (type TXT) records depend on your DNS or hosting provider. See your provider's documentation or contact them for information about adding these records to the DNS configuration for your domain.
- To successfully set up a custom MAIL FROM domain with Amazon SES, you must publish exactly one MX record to the DNS server of your MAIL FROM domain. If the MAIL FROM domain has multiple MX records, the custom MAIL FROM setup with Amazon SES will fail.

If Route 53 provides the DNS service for your MAIL FROM domain, and you're signed in to the AWS Management Console under the same account that you use for Route 53, then choose **Publish Records Using Route 53**. The DNS records are automatically applied to your domain's DNS configuration.

If you use a different DNS provider, you have to publish the DNS records to the MAIL FROM domain's DNS server manually. The procedure for adding DNS records to your domain's DNS server varies based on your web hosting service or DNS provider.

The procedures for publishing DNS records for your domain depend on which DNS provider you use. The following table includes links to the documentation for a few widely used DNS providers. This list isn't exhaustive and doesn't signify endorsement; likewise, if your DNS provider isn't listed, it doesn't imply they don't support MAIL FROM domain configuration.

DNS/Hosting provider name	Documentation link
GoDaddy	• MX: Add an MX record (external link) • TXT: Add a TXT record (external link)
DreamHost	• MX: How do I change my MX records? (external link) • TXT: How do I add custom DNS records? (external link)

DNS/Hosting provider name	Documentation link
Cloudflare	• MX: How do I add or edit mail or MX records? (external link) • TXT: Managing DNS records in Cloudflare (external link)
HostGator	• MX: Set up MX Records (external link) • TXT: Manage DNS Records with HostGator /eNom (external link)
Namecheap	• MX: How can I set up MX records required for mail service? (external link) • TXT: How do I add TXT/SPF/DKIM/DMARC records for my domain? (external link)
Names.co.uk	• MX: Changing your domain's DNS settings (external link) • TXT: Changing your domains DNS Settings (external link)
Wix	• MX: Adding or Updating MX Records in Your Wix Account (external link) • TXT: Adding or Updating TXT Records in Your Wix Account (external link)

When Amazon SES detects that the records are in place, you receive an email informing you that your custom MAIL FROM domain was set up successfully. Depending on your DNS provider, there might be a delay of up to 72 hours before Amazon SES detects the MX record.

Setting up a custom MAIL FROM domain for a verified email address

You can also set up a custom MAIL FROM domain for a specific email address. In order to set up a custom MAIL FROM domain for an email address, you must modify the DNS records for the domain that the email address is associated with.

> **Note**
>
> You can't set up a custom MAIL FROM domain for addresses on a domain that you don't own (for example, you can't create a custom MAIL FROM domain for an address on the `gmail.com` domain, because you can't add the necessary DNS records to the domain).

To configure a verified email address to use a specified MAIL FROM domain

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, under **Configuration**, choose **Identities**.
3. In the list of identities, choose the identity you want to configure where the **Identity type** is **Email address** and **Status** is *Verified*.
   - If the **Status** is *Unverified*, complete the procedures at Verifying an email address identity to verify the email address's domain.
4. Under the **MAIL FROM Domain** tab, choose **Edit** in the **Custom MAIL FROM domain** pane.
5. In the **General details** pane, do the following:
   a. Select the **Use a custom MAIL FROM domain** checkbox.
   b. For **MAIL FROM domain**, enter the subdomain that you want to use as the MAIL FROM domain.
   c. For **Behavior on MX failure**, choose one of the following options:
      - **Use default MAIL FROM domain** – If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES uses a subdomain of `amazonses.com`. The subdomain varies based on the AWS Region that you use Amazon SES in.
      - **Reject message** – If the custom MAIL FROM domain's MX record is not set up correctly, Amazon SES returns a `MailFromDomainNotVerified` error. Emails that you attempt to send from this email address are automatically rejected.
   d. Choose **Save changes** - you'll be returned to the previous screen.
6. Publish the MX and SPF (type TXT) records to the DNS server of the custom MAIL FROM domain:

In the **Custom MAIL FROM domain** pane, the **Publish DNS records** table now displays the MX and SPF (type TXT) records in that you have to publish (add) to your domain's DNS configuration. These records use the formats shown in the following table.

Name	Type	Value
subdomain .*domain*.*com*	MX	
subdomain .*domain*.*com*	TXT	

In the preceding records,

- *subdomain*.*domain*.*com* will be populated with your MAIL FROM subdomain
- *region* will be populated with the name of the AWS Region where you want to verify the MAIL FROM domain (such as `us-west-2`, `us-east-1`, or `eu-west-1`, etc.)
- The number *10* listed along with the MX value is the preference order for the mail server and will need to be entered into a separate value field as specified by your DNS provider's GUI
- The SPF's TXT record value has to include the quotation marks

From the **Publish DNS records** table, copy the MX and SPF (type TXT) records by choosing the copy icon next to each value and paste them into the corresponding fields in your DNS provider's GUI. Alternatively, you can choose **Download .csv record set** to save a copy of the records to your computer.

> **Important**
>
> To successfully set up a custom MAIL FROM domain with Amazon SES, you must publish exactly one MX record to the DNS server of your MAIL FROM domain. If the MAIL FROM domain has multiple MX records, the custom MAIL FROM setup with Amazon SES will fail.

If Route 53 provides the DNS service for your MAIL FROM domain, and you're signed in to the AWS Management Console under the same account that you use for Route 53, then choose

Publish Records Using Route 53. The DNS records are automatically applied to your domain's DNS configuration.

If you use a different DNS provider, you have to publish the DNS records to the MAIL FROM domain's DNS server manually. The procedure for adding DNS records to your domain's DNS server varies based on your web hosting service or DNS provider.

The procedures for publishing DNS records for your domain depend on which DNS provider you use. The following table includes links to the documentation for a few widely used DNS providers. This list isn't exhaustive and doesn't signify endorsement; likewise, if your DNS provider isn't listed, it doesn't imply they don't support MAIL FROM domain configuration.

DNS/Hosting provider name	Documentation link
GoDaddy	• MX: Add an MX record (external link) • TXT: Add a TXT record (external link)
DreamHost	• MX: How do I change my MX records? (external link) • TXT: How do I add custom DNS records? (external link)
Cloudflare	• MX: How do I add or edit mail or MX records? (external link) • TXT: Managing DNS records in Cloudflare (external link)
HostGator	• MX: Changing MX records - Windows (external link) • TXT: Manage DNS Records with HostGator /eNom (external link)
Namecheap	• MX: How can I set up MX records required for mail service? (external link) • TXT: How do I add TXT/SPF/DKIM/DMARC records for my domain? (external link)

DNS/Hosting provider name	Documentation link
Names.co.uk	• MX: Changing your domain's DNS settings (external link) • TXT: Changing your domains DNS Settings (external link)
Wix	• MX: Adding or Updating MX Records in Your Wix Account (external link) • TXT: Adding or Updating TXT Records in Your Wix Account (external link)

When Amazon SES detects that the records are in place, you receive an email informing you that your custom MAIL FROM domain was set up successfully. Depending on your DNS provider, there might be a delay of up to 72 hours before Amazon SES detects the MX record.

Custom MAIL FROM domain setup states with Amazon SES

After you configure an identity to use a custom MAIL FROM domain, the state of the setup is "pending" while Amazon SES attempts to detect the required MX record in your DNS settings. The state then changes depending on whether Amazon SES detects the MX record. The following table describes the email-sending behavior, and the Amazon SES actions associated with each state. Each time the state changes, Amazon SES sends a notification to the email address associated with your AWS account.

State	Email sending behavior	Amazon SES actions
Pending	Uses custom MAIL FROM fallback setting	Amazon SES attempts to detect the required MX record for 72 hours. If unsuccessful, the

State	Email sending behavior	Amazon SES actions
		state changes to "Failed".
Success	Uses custom MAIL FROM domain	Amazon SES continuously checks that the required MX record is in place.
Temporary Failure	Uses custom MAIL FROM fallback setting	Amazon SES attempts to detect the required MX record for 72 hours. If unsuccessful, the state changes to "Failed"; if successful, the state changes to "Success".

State	Email sending behavior	Amazon SES actions
Failed	Uses custom MAIL FROM fallback setting	Amazon SES no longer attempts to detect the required MX record. To use a custom MAIL FROM domain, you have to restart the setup process in Configuring your custom MAIL FROM domain.

Complying with DMARC authentication protocol in Amazon SES

Domain-based Message Authentication, Reporting and Conformance (DMARC) is an email authentication protocol that uses Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM) to detect email spoofing and phishing. In order to comply with DMARC, messages must be authenticated through either SPF or DKIM, but ideally, when both are used with DMARC, you'll be ensuring the highest level of protection possible for your email sending.

Let's briefly review which each does and how DMARC ties them all together:

- **SPF** – Identifies which mail servers are allowed to send mail on behalf of your custom MAIL FROM domain through a DNS TXT record that is used by DNS. Recipient mail systems refer to the SPF TXT record to determine whether a message from your custom domain comes from an authorized messaging server. Basically, SPF is designed to help prevent spoofing, but there are spoofing techniques that SPF is susceptible to in practice and this is why you need to also use DKIM along with DMARC.
- **DKIM** – Adds a digital signature to your outbound messages in the email header. Receiving email systems can use this digital signature to help verify whether incoming email is signed by a key owned by the domain. However, when a receiving email system forwards a message, the message's envelope is changed in a way that invalidates SPF authentication. Since the digital

signature stays with the email message because it's part of the email header, DKIM works even when a message has been forwarded between mail servers (as long as the message content has not been modified).

- **DMARC** – Ensures that there is domain alignment with at least one of SPF and DKIM. Using SPF and DKIM alone does nothing to insure that the From address is authenticated (this is the email address your recipient sees in their email client). SPF only checks the domain specified in the MAIL FROM address (not seen by your recipient). DKIM only checks the domain specified in the DKIM signature (also, not seen by your recipient). DMARC addresses these two issues by requiring domain alignment to be correct on either SPF or DKIM:
  - For SPF to pass DMARC alignment the domain in the From address must match the domain in the MAIL FROM address (also referred to as Return-Path and Envelope-from address). This is rarely possible with forwarded mail because it gets stripped away or when sending mail through third-party bulk email providers because the Return-Path (MAIL FROM) is used for bounces and complaints that the provider (SES) tracks using an address they own.
  - For DKIM to pass DMARC alignment, the domain specified in the DKIM signature must match the domain in the From address. If you use third-party senders or services that send mail on your behalf, this can be accomplished by ensuring the third-party sender is properly configured for DKIM signing and you have added the appropriate DNS records within your domain. Receiving mail servers will then be able to verify email sent to them by your third-party as if it was email sent by someone authorized to use an address within the domain.

Putting it all together with DMARC

The DMARC alignment checks we discussed above show how SPF, DKIM, and DMARC all work together to increase trust of your domain and delivery of your email to inboxes. DMARC accomplishes this by ensuring that the From address, seen by the recipient, is authenticated by either SPF or DKIM:

- A message passes DMARC if one or both of the described SPF or DKIM checks pass.
- A message fails DMARC if both of the described SPF or DKIM checks fail.

Therefore, both SPF and DKIM are necessary for DMARC to have the best chance at achieving authentication for your sent email, and by utilizing all three, you'll help to ensure you have a fully protected sending domain.

DMARC also allows you to instruct email servers how to handle emails when they fail DMARC authentication through policies that you set. This will be explained in the following section, the section called “Setting up the DMARC policy on your domain”, that contains information on how to configure your SES domains so that the emails you send comply with the DMARC authentication protocol through both SPF and DKIM.

Setting up the DMARC policy on your domain

To set up DMARC, you have to modify the DNS settings for your domain. The DNS settings for your domain should include a TXT record that specifies the domain's DMARC settings. The procedures for adding TXT records to your DNS configuration depend on which DNS or hosting provider you use. If you use Route 53, see Working with Records in the *Amazon Route 53 Developer Guide*. If you use another provider, see the DNS configuration documentation for your provider.

The name of the TXT record you create should be `_dmarc.`*`example.com`*, where *`example.com`* is your domain. The value of the TXT record contains the DMARC policy that applies to your domain. The following is an example of a TXT record that contains a DMARC policy:

Name	Type	Value
`_dmarc.example.com`	`TXT`	`"v=DMARC1;p=quarantine;rua=mailto:my_dmarc_report@example.com"`

In the preceding DMARC policy example, this policy tells email providers to do the following:

- For any messages that fail authentication, send them to the Spam folder as specified by the policy parameter, `p=quarantine`. Other options include doing nothing by using `p=none`, or reject the message outright by using `p=reject`.
  - The next section discusses how and when to use these three policy settings—*using the wrong one at the wrong time can cause your email to not be delivered,* see the section called “Implementing DMARC”.
- Send reports about all emails that failed authentication in a digest (that is, a report that aggregates the data for a certain time period, rather than sending individual reports for each event) as specified by the reporting parameter, `rua=mailto:my_dmarc_report@example.com` (*rua* stands for Reporting URI for Aggregate

reports). Email providers typically send these aggregated reports once per day, although these policies differ from provider to provider.

To learn more about configuring DMARC for your domain, see the Overview on the DMARC website.

For complete specifications of the DMARC system, see Internet Engineering Task Force (IETF) DMARC Draft.

Best practices for implementing DMARC

It's best to implement your DMARC policy enforcement in a gradual and phased approach so that it doesn't interrupt the rest of your mail flow. Create and implement a roll-out plan that follows these steps. Do each of these steps first with each of your sub-domains, and finally with the top-level domain in your organization before moving on to the next step.

1. Monitor the impact of implementing DMARC (p=none).
   - Start with a simple monitoring-mode record for a sub-domain or domain that requests that mail receiving organizations send you statistics about messages that they see using that domain. A monitoring-mode record is a DMARC TXT record that has its policy set to none p=none.
   - Reports generated through DMARC will give the numbers and sources of messages that pass these checks, versus those that don't. You can easily see how much of your legitimate traffic is or isn't covered by them. You'll see signs of forwarding, since forwarded messages will fail SPF and DKIM if the content is modified. You'll also begin to see how many fraudulent messages are being sent, and where they're sent from.
   - The goals of this step are to learn what emails will be impacted when you implement one of the next two steps, and to have any third-party or authorized senders get their SPF or DKIM policies into alignment.
   - Best for existing domains.
2. Request that external mail systems quarantine mail that fails DMARC (p=quarantine).
   - When you believe that all or most of your legitimate traffic is sending domain-aligned with either SPF or DKIM, and you understand the impact of implementing DMARC, you can implement a quarantine policy. A quarantine policy is a DMARC TXT record that has its policy set to quarantine `p=quarantine`. By doing this, you're asking DMARC receivers to put messages from your domain that fail DMARC into the local equivalent of a spam folder instead of your customers' inboxes.

- Best for transitioning domains that have analyzed DMARC reports during Step 1.

3. Request that external mail systems not accept messages that fail DMARC (p=reject).
   - Implementing a reject policy is usually the final step. A reject policy is a DMARC TXT record that has its policy set to reject `p=reject`. When you do this, you're asking DMARC receivers not to accept messages that fail the DMARC checks—this means they won't even be quarantined to a spam or junk folder, but will be rejected outright.
   - When using a reject policy, you'll know exactly which messages are failing the DMARC policy as the rejection will result in a SMTP bounce. With quarantine, the aggregate data provides information about the percentages of email passing or failing SPF, DKIM, and DMARC checks.
   - Best for new domains or existing domains that have gone through the prior two steps.

Complying with DMARC through SPF

For an email to comply with DMARC based on SPF, both of the following conditions must be met:

- The message must pass an SPF check based on having a valid SPF (type TXT) record that you've to published to your custom MAIL FROM domain's DNS configuration.
- The domain in the From address of the email header must align (match) with the domain, or a subdomain of, that's specified in the MAIL FROM address. In order to achieve SPF alignment with SES, the domain's DMARC policy must not specify a strict SPF policy (aspf=s).

To comply with these requirements, complete the following steps:

- Set up a custom MAIL FROM domain by completing the procedures in the section called “Using a custom MAIL FROM domain”.
- Ensure that your sending domain uses a relaxed policy for SPF. If you haven't changed your domain's policy alignment, it uses a relaxed policy by default as does SES.

> **Note**
>
> You can determine your domain's DMARC alignment for SPF by typing the following command at the command line, replacing *example.com* with your domain:
>
> ```
> dig TXT _dmarc.example.com
> ```

In the output of this command, under **Non-authoritative answer**, look for a record that begins with `v=DMARC1`. If this record includes the string `aspf=r`, or if the `aspf` string is not present at all, then your domain uses relaxed alignment for SPF. If the record includes the string `aspf=s`, then your domain uses strict alignment for SPF. Your system administrator will need to remove this tag from the DMARC TXT record in your domain's DNS configuration.
Alternatively, you can use a web-based DMARC lookup tool, such as the DMARC Inspector from the dmarcian website or the DMARC Check Tool tool from the MxToolBox website, to determine your domain's policy alignment for SPF.

Complying with DMARC through DKIM

For an email to comply with DMARC based on DKIM, both of the following conditions must be met:

- The message must have a valid DKIM signature and passes the DKIM check.
- The domain specified in the DKIM signature must align (match) with the domain in the From address. If the domain's DMARC policy specifies strict alignment for DKIM, these domains must match exactly (SES uses a strict DKIM policy by default).

To comply with these requirements, complete the following steps:

- Set up Easy DKIM by completing the procedures in the section called “Easy DKIM”. When you use Easy DKIM, Amazon SES automatically signs your emails.

> **Note**
>
> Rather than use Easy DKIM, you can also manually sign your messages. However, use caution if you choose to do so, because Amazon SES does not validate the DKIM signature that you construct. For this reason, we highly recommend using Easy DKIM.

- Ensure the domain specified in the DKIM signature is aligned to the domain in the From address. Or, if sending from a subdomain of the domain in the From address, ensure that your DMARC policy is set to relaxed alignment.

Note

You can determine your domain's DMARC alignment for DKIM by typing the following command at the command line, replacing *example.com* with your domain:

```
dig TXT _dmarc.example.com
```

In the output of this command, under **Non-authoritative answer**, look for a record that begins with `v=DMARC1`. If this record includes the string `adkim=r`, or if the `adkim` string is not present at all, then your domain uses relaxed alignment for DKIM. If the record includes the string `adkim=s`, then your domain uses strict alignment for DKIM. Your system administrator will need to remove this tag from the DMARC TXT record in your domain's DNS configuration.
Alternatively, you can use a web-based DMARC lookup tool, such as the DMARC Inspector from the dmarcian website or the DMARC Check Tool tool from the MxToolBox website, to determine your domain's policy alignment for DKIM.

Using BIMI in Amazon SES

Brand Indicators for Message Identification (BIMI) is an email specification that enables email inboxes to display a brand's logo next to the brand's authenticated email messages within supporting email clients.

BIMI is an email specification that's directly connected to authentication, but it's not a standalone email authentication protocol as it requires all your email to comply with DMARC authentication.

While BIMI requires DMARC, DMARC requires your domain to have either SPF or DKIM records to align, but it's best to include both SPF and DKIM records for additional security, and because some email service providers (ESPs) require both when using BIMI. The following section goes over the steps to implement BIMI in Amazon SES.

Setting up BIMI in SES

You can configure BIMI for an email domain that you own—in SES that's referred to as a *custom* MAIL FROM domain. Once configured, all of the messages that you send from that domain will display your BIMI logo in email clients that support BIMI.

Enabling your emails to display a BIMI logo requires some prerequisites to be in place within SES—in the following procedure, these prerequisites are generalized and will reference dedicated sections that cover these topics in detail. The steps specific to BIMI and what is necessary to configure it in SES will be detailed here.

To set up BIMI on a custom MAIL FROM domain

1. You must have a custom MAIL FROM domain configured in SES with both SPF (type TXT) and MX records published for that domain. If you don't have a custom MAIL FROM domain, or would like to create a new one for your BIMI logo, see the section called “Using a custom MAIL FROM domain”.
2. Configure your domain with Easy DKIM. See the section called “Easy DKIM”.
3. Configure your domain with DMARC by publishing a TXT record with your DNS provider with the following enforcement policy specifics required for BIMI similar to either of the two examples:

Name	Type	Value
`_dmarc.example.com`	`TXT`	`v=DMARC1;p=quarantine;pct=100;rua=mailto:dmarcreports@example.com`
`_dmarc.example.com`	`TXT`	`v=DMARC1;p=reject;rua=mailto:dmarcreports@example.com`

In the preceding DMARC policy example as required for BIMI:

- *`example.com`* should be replaced with your domain or subdomain name.
- The p= value can be either:
  - *quarantine* with a *pct* value set to *100* as shown, or
  - *reject* as shown.
- If you're sending from a subdomain, BIMI requires that the parent domain must also have this enforcement policy. Subdomains will fall under the parent domain’s policy. However, if you add a DMARC record for your subdomain in addition to what is posted for the parent domain, your subdomain must also have the same enforcement policy to be eligible for BIMI.

- If you've never set up a DMARC policy for your domain, see the section called "Authenticating Email with DMARC" ensuring that you only use the DMARC policy values specific to BIMI as shown.

4. Produce your BIMI logo as a Scalable Vector Graphics (SVG) `.svg` file—the specific SVG profile required by BIMI is defined as SVG Portable/Secure (SVG P/S). In order for your logo to display in the email client it must conform exactly to these specifications. See the BIMI Group's guidance on creating SVG logo files and recommended SVG conversion tools.
5. (Optional) Obtain a Verified Mark Certificate (VMC). Some ESPs, such as Gmail and Apple, require a VMC to provide evidence that you own the trademark and content of your BIMI logo. Although this isn't a requirement for implementing BIMI on your domain, your BIMI logo will not display in the email client if the ESP you send mail to enforces VMC compliance. See the BIMI Group's references to participating certificate authorities to obtain a VMC for your logo.
6. Host your BIMI logo's SVG file on a server you have access to making it publicly accessible through HTTPS. For example, you could upload it to an Amazon S3 bucket.
7. Create and publish a BIMI DNS record that includes a URL to your logo. When an ESP that supports BIMI checks your DMARC record, it will also look for a BIMI record containing the URL for your logo's `.svg` file, and if configured, the URL for your VMC's `.pem` file. If the records match, they'll display your BIMI logo.

   Configure your domain with BIMI by publishing a TXT record with your DNS provider with the following values as shown—sending from a domain is represented in the first example; sending from a subdomain is represented in the second example:

Name	Type	Value
`default._bimi.example.com`	TXT	`v=BIMI1;l=https://myhostingserver.com/images/logo.svg;a=https://myhostingserver.com/certificate/vmc_2023-01-01.pem`
`default._bimi.marketing.example.com`		

   In the preceding BIMI record examples:

- The name value should literally specify `default._bimi.` as a subdomain of *`example.com`* or *`marketing.example.com`* which should be replaced with your domain or subdomain name.
- The `v=` value is the *version* of the BIMI record.
- The `l=` value is the *logo* representing the URL pointing to your image's `.svg` file.
- The `a=` value is the *authority* representing the URL pointing to your certificate's `.pem` file.

You can validate your BIMI record with a tool like the BIMI Group's BIMI Inspector.

The final step in this process is to have a regular sending pattern to the ESPs that support BIMI logo placement. Your domain should have a regular delivery cadence and should have a good reputation with the ESPs that you're sending to. BIMI logo placement may take time to populate to ESPs where you don't have an established reputation or sending cadence.

You can find more information and resources pertaining to BIMI through the BIMI Group organization.

Setting up event notifications for Amazon SES

In order to send email using Amazon SES, you must have a system in place for managing bounces and complaints. Amazon SES can notify you of bounce or complaint events in three ways: by sending a notification email, by notifying an Amazon SNS topic, or by publishing sending events. This section contains information about setting up Amazon SES to send certain kinds of notifications by email or by notifying an Amazon SNS topic. For more information about publishing sending events, see Monitor email sending using Amazon SES event publishing.

You can set up notifications using the Amazon SES console or the Amazon SES API.

Topics

- Important considerations
- Receiving Amazon SES notifications through email
- Receiving Amazon SES notifications using Amazon SNS

Important considerations

There are several important points to consider when you set up Amazon SES to send notifications:

- Email and Amazon SNS notifications apply to individual identities (the verified email addresses or domains you use to send email). When you enable notifications for an identity, Amazon SES only sends notifications for emails sent from that identity, and only in the AWS Region you configured notifications in.
- You have to enable one method of receiving bounce or complaint notifications. You can send notifications to the domain or email address that generated the bounce or complaint, or to an Amazon SNS topic. You can also use event publishing to send notifications about several different types of events (including bounces, complaints, deliveries, and more) to an Amazon SNS topic or an Firehose stream.

  If you don't set up one of these methods of receiving bounce or complaint notifications, Amazon SES automatically forwards bounce and complaint notifications to the Return-Path address (or the Source address, if you didn't specify a Return-Path address) in the email that resulted in the bounce or complaint event, even if you disabled email feedback forwarding.

  If you disable email feedback forwarding and enable event publishing, you must apply the configuration set that contains the event publishing rule to all emails you send. In this situation, if you don't use the configuration set, Amazon SES automatically forwards bounce and complaint notifications to the Return-Path or Source address in the email that resulted in the bounce or complaint event.
- If you set up Amazon SES to send bounce and complaint events using more than one method (such as by sending email notifications and by using sending events), you may receive more than one notification for the same event.

Receiving Amazon SES notifications through email

Amazon SES can send you email when you receive bounces and complaints by using a process called *email feedback forwarding*.

In order to send email using Amazon SES, you must configure it to send bounce and complaint notifications by using one of the following methods:

- By enabling email feedback forwarding. The procedure for setting up this type of notification is included in this section.
- By sending notifications to an Amazon SNS topic. For more information, see Receiving Amazon SES notifications using Amazon SNS.

- By publishing event notifications. For more information, see Monitor email sending using Amazon SES event publishing.

> **Important**
>
> For several important points about notifications, see Setting up event notifications for Amazon SES.

Topics

- Enabling email feedback forwarding
- Disabling email feedback forwarding
- Email feedback forwarding destination

Enabling email feedback forwarding

Email feedback forwarding is enabled by default. If you previously disabled it, you can enable it by following the procedures in this section.

To enable bounce and complaint forwarding through email using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of verified email addresses or domains, choose the email address or domain that you want to configure bounce and complaint notifications for.
4. In the details pane, expand the **Notifications** section.
5. Choose **Edit Configuration**.
6. Under **Email Feedback Forwarding**, choose **Enabled**.

> **Note**
>
> Changes you make on this page may take a few minutes to take effect.

You can also enable bounce and complaint notifications through email by using the SetIdentityFeedbackForwardingEnabled API operation.

Disabling email feedback forwarding

If you set up a different method of providing bounce and complaint notifications, you can disable email feedback forwarding so that you don't receive multiple notifications when a bounce or complaint event occurs.

To disable bounce and complaint forwarding through email using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of verified email addresses or domains, choose the email address or domain that you want to configure bounce and complaint notifications for.
4. In the details pane, expand the **Notifications** section.
5. Choose **Edit Configuration**.
6. Under **Email Feedback Forwarding**, choose **Disabled**.

> **Note**
>
> You must configure one method of receiving bounce and complaint notifications in order to send email through Amazon SES. If you disable email feedback forwarding, you must enable notifications sent by Amazon SNS, or publish bounce and complaint events to an Amazon SNS topic or a Firehose stream by using event publishing. If you use event publishing, you must also apply the configuration set that contains the event publishing rule to each email you send. If you don't set up a method of receiving bounce and complaint notifications, Amazon SES automatically forwards feedback notifications by email to the address in the Return-Path field (or the Source field, if you didn't specify a Return-Path address) of the message that resulted in the bounce or complaint event. In this situation, Amazon SES forwards bounce and complaint notifications even if you disabled email feedback notifications.

7. To save your notification configuration, choose **Save Config**.

> **Note**
>
> Changes you make on this page might take a few minutes to take effect.

You can also disable bounce and complaint notifications through email by using the SetIdentityFeedbackForwardingEnabled API operation.

Email feedback forwarding destination

When you receive notifications by email, Amazon SES rewrites the `From` header and sends the notification to you. The address to which Amazon SES forwards the notification depends on how you sent the original message.

If you used the SMTP interface to send the message, then the notifications are delivered according to the following rules:.

- If you specified a `Return-Path` header in the `SMTP DATA` section, then notifications go to that address.
- Otherwise, notifications go to the address you specified when you issued the MAIL FROM command.

If you used the `SendEmail` API operation to send the message, then the notifications are delivered according to the following rules:

- If you specified the optional `ReturnPath` parameter in your call to the `SendEmail` API, then notifications go to that address.
- Otherwise, notifications go to the address specified in the required `Source` parameter of `SendEmail`.

If you used the `SendRawEmail` API operation to send the message, then the notifications are delivered according to the following rules:

- If you specified a `Return-Path` header in the raw message, then notifications go to that address.
- Otherwise, if you specified a `Source` parameter in your call to the `SendRawEmail` API, then notifications go to that address.

- Otherwise, notifications go to the address in the `From` header of the raw message.

> **Note**
>
> When you specify a `Return-Path` address in an email, you receive notifications at that address. However, the version of the message that the recipient receives contains a `Return-Path` header that includes an anonymized email address (such as *a0b1c2d3e4f5a6b7-c8d9e0f1-a2b3-c4d5-e6f7-a8b9c0d1e2f3-000000@amazonses.com*). This anonymization happens regardless of how you sent the email.

Receiving Amazon SES notifications using Amazon SNS

You can configure Amazon SES to notify an Amazon SNS topic when you receive bounces or complaints, or when emails are delivered. Amazon SNS notifications are in JavaScript Object Notation (JSON) format, which enables you to process them programmatically.

In order to send email using Amazon SES, you must configure it to send bounce and complaint notifications by using one of the following methods:

- By sending notifications to an Amazon SNS topic. The procedure for setting up this type of notification is included in this section.
- By enabling email feedback forwarding. For more information, see Receiving Amazon SES notifications through email.
- By publishing event notifications. For more information, see Monitor email sending using Amazon SES event publishing.

> **Important**
>
> See Setting up event notifications for Amazon SES for important information about notifications.

Topics

- Configuring Amazon SNS notifications for Amazon SES
- Amazon SNS notification contents for Amazon SES

- Amazon SNS notification examples for Amazon SES

Configuring Amazon SNS notifications for Amazon SES

Amazon SES can notify you of your bounces, complaints, and deliveries through Amazon Simple Notification Service (Amazon SNS).

You can configure notifications in the Amazon SES console, or by using the Amazon SES API.

Topics in this section:

- Prerequisites
- Configuring notifications using the Amazon SES console
- Configuring notifications using the Amazon SES API
- Troubleshooting feedback notifications

Prerequisites

Complete the following steps before you set up Amazon SNS notifications in Amazon SES:

1. Create a topic in Amazon SNS. For more information, see Create a Topic in the *Amazon Simple Notification Service Developer Guide*.

   > **Important**
   >
   > When you create your topic using Amazon SNS, for **Type**, only choose **Standard**. (SES does not support FIFO type topics.)

   Whether you create a new SNS topic or select an existing one, you need to give access to SES to publish notifications to the topic.

   To give Amazon SES permission to publish notifications to the topic, on the **Edit topic** screen in the SNS console, expand **Access policy** and in the **JSON editor**, add the following permission policy:

   JSON

   ```
   {
   ```

```
    "Version":"2012-10-17",
    "Id": "notification-policy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": "sns:Publish",
            "Resource": "arn:aws-eusc:sns:eusc-de-
east-1:111122223333:topic_name",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceAccount": "111122223333",
                    "AWS:SourceArn": "arn:aws-
eusc:ses:topic_region:111122223333:identity/identity_name"
                }
            }
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *topic_region* with the AWS Region where you created the SNS topic.
- Replace *111122223333* with your AWS account ID.
- Replace *topic_name* with the name of your SNS topic.
- Replace *identity_name* with the verified identity (email address or domain) that you're subscribing to the SNS topic.

2. Subscribe at least one endpoint to the topic. For example, if you want to receive notifications by text message, subscribe an SMS endpoint (that is, a mobile phone number) to the topic. To receive notifications by email, subscribe an email endpoint (an email address) to the topic.

   For more information, see Getting Started in the *Amazon Simple Notification Service Developer Guide*.

3. (Optional) If your Amazon SNS topic uses AWS Key Management Service (AWS KMS) for server-side encryption, you have to add permissions to the AWS KMS key policy. You can add permissions by attaching the following policy to the AWS KMS key policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSESToUseKMSKey",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

Configuring notifications using the Amazon SES console

To configure notifications using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the **Identities** container, select the verified identity you want to receive feedback notifications for when a message sent from this identity results in either a bounce, complaint, or delivery.

> **Important**
>
> Verified domain notification settings apply to all mail sent from email addresses in that domain *except* for email addresses that are also verified.

4. In the details screen of the verified identity you selected, choose the **Notifications** tab and select **Edit** in the **Feedback notifications** container.

5. Expand the SNS topic list box of each feedback type you want to receive notifications for, and select either an SNS topic you own, **No SNS topic**, or **SNS topic you don't own**.

   - If you chose **SNS topic you don't own**, the **SNS topic ARN** field will be presented where you must enter the SNS topic ARN shared with you by your delegate sender. (Only your delegate sender will get these notifications because they own the SNS topic. To learn more about delegate sending, see Overview of sending authorization.)

   > **Important**
   >
   > The Amazon SNS topics that you use for bounce, complaint, and delivery notifications have to be in the same AWS Region that in which you use Amazon SES.
   > Additionally, you have to subscribe one or more endpoints to the topic in order to receive notifications. For example, if you want to have notifications sent to an email address, you have to subscribe an email endpoint to the topic. For more information, see Getting Started in the *Amazon Simple Notification Service Developer Guide*.

6. (Optional) If you want your topic notification to include the headers from the original email, check the **Include original email headers** box directly underneath the SNS topic name of each feedback type. This option is only available if you've assigned an Amazon SNS topic to the associated notification type. For information about the contents of the original email headers, see the `mail` object in Notification contents.
7. Choose **Save changes**. The changes you made to your notification settings might take a few minutes to take effect.
8. (Optional) If you chose Amazon SNS topic notifications for both bounces and complaints, you can disable email notifications entirely so that you don't receive double notifications through email and SNS notifications. To disable email notifications for bounces and complaints, under the **Notifications** tab on the details screen of the verified identity, in the **Email Feedback Forwarding** container, choose **Edit**, uncheck the **Enabled** box, and choose **Save changes**. .

After you configure your settings, you will start receiving bounce, complaint, and delivery notifications to your Amazon SNS topics. These notifications are in JavaScript Object Notation (JSON) format and follow the structure described in Notification contents.

You will be charged standard Amazon SNS rates for bounce, complaint, and delivery notifications. For more information, see the Amazon SNS pricing page.

> **Note**
>
> If an attempt to publish to your Amazon SNS topic fails because the topic has been deleted or your AWS account no longer has permissions to publish to it, Amazon SES removes the configuration for that topic if it's been configured for bounces or complaints (not deliveries - for delivery notifications, SES won't delete the SNS topic configuration setting). Additionally, Amazon SES re-enables bounce and complaint email notifications for the identity, and you receive a notification of the change by email. If multiple identities are configured to use the topic, the topic configuration for each identity is changed when each identity experiences a failure to publish to the topic.

Configuring notifications using the Amazon SES API

You can also configure bounce, complaint, and delivery notifications by using the Amazon SES API. Use the following operations to configure notifications:

- SetIdentityNotificationTopic
- SetIdentityFeedbackForwardingEnabled
- GetIdentityNotificationAttributes
- SetIdentityHeadersInNotificationsEnabled

You can use these API actions to write a customized front-end application for notifications. For a complete description of the API actions related to notifications, see the Amazon Simple Email Service API Reference.

Troubleshooting feedback notifications

Not receiving notifications

If you aren't receiving notifications, make sure that you subscribed an endpoint to the topic that the notifications are sent through. When you subscribe an email endpoint to a topic, you receive an email asking you to confirm your subscription. You have to confirm your subscription before you start receiving email notifications. For more information, see Getting Started in the *Amazon Simple Notification Service Developer Guide*.

`InvalidParameterValue` error when choosing a topic

If you receive an error stating that an `InvalidParameterValue` error occurred, check the Amazon SNS topic to see if it's encrypted using AWS KMS. If it is, you have to modify the policy for the AWS KMS key. See Prerequisites for a sample policy.

Amazon SNS notification contents for Amazon SES

Bounce, complaint, and delivery notifications are published to Amazon Simple Notification Service (Amazon SNS) topics in JavaScript Object Notation (JSON) format. The top-level JSON object contains a `notificationType` string, a `mail` object, and either a `bounce` object, a `complaint` object, or a `delivery` object.

See the following sections for descriptions of the different types of objects:

- Top-level JSON object
- `mail` object
- `bounce` object
- `complaint` object
- `delivery` object

The following are some important notes about the contents of Amazon SNS notifications for Amazon SES:

- For a given notification type, you might receive one Amazon SNS notification for multiple recipients, or you might receive a single Amazon SNS notification per recipient. Your code should be able to parse the Amazon SNS notification and handle both cases; SES does not make ordering or batching guarantees for notifications sent through Amazon SNS. However, different Amazon SNS notification types (for example, bounces and complaints) are not combined into a single notification.
- You might receive multiple types of Amazon SNS notifications for one recipient. For example, the receiving mail server might accept the email (triggering a delivery notification), but after processing the email, the receiving mail server might determine that the email actually results in a bounce (triggering a bounce notification). However, these are always separate notifications because they are different notification types.
- SES reserves the right to add additional fields to the notifications. As such, applications that parse these notifications must be flexible enough to handle unknown fields.
- SES overwrites the headers of the message when it sends the email. You can retrieve the headers of the original message from the `headers` and `commonHeaders` fields of the `mail` object.

Top-Level JSON object

The top-level JSON object in an SES notification contains the following fields.

Field name	Description
`notificationType`	A string that holds the type of notification represented by the JSON object. Possible values are `Bounce`, `Complaint` , or `Delivery`. If you set up event publishing, this field is named `eventType` .
`mail`	A JSON object that contains information about the original mail to which the notification pertains. For more information, see Mail object.
`bounce`	This field is present only if the `notificationType` is `Bounce` and contains a JSON object that holds information about the bounce. For more information, see Bounce object.
`complaint`	This field is present only if the `notificationType` is `Complaint` and contains a JSON object that holds information about the complaint. For more information, see Complaint object.
`delivery`	This field is present only if the `notificationType` is `Delivery` and contains a JSON object that holds information about the delivery. For more information, see Delivery object.

Mail object

Each bounce, complaint, or delivery notification contains information about the original email in the `mail` object. The JSON object that contains information about a `mail` object has the following fields.

Field name	Description
`timestamp`	The time at which the original message was sent (in ISO8601 format).
`messageId`	A unique ID that SES assigned to the message. SES returned this value to you when you sent the message. **Note** This message ID was assigned by SES. You can find the message ID of the original email in the `headers` field of the `mail` object.
`source`	The email address from which the original message was sent (the envelope MAIL FROM address).
`sourceArn`	The Amazon Resource Name (ARN) of the identity that was used to send the email. In the case of sending authorization, the `sourceArn` is the ARN of the identity that the identity owner authorized the delegate sender to use to send the email. For more information about sending authorization, see Email authentication methods.
`sourceIp`	The originating public IP address of the client that performed the email sending request to SES.

Field name	Description
`sendingAccountId`	The AWS account ID of the account that was used to send the email. In the case of sending authorization, the `sendingAccountId` is the delegate sender's account ID.
`callerIdentity`	The IAM identity of the SES user who sent the email.
`destination`	A list of email addresses that were recipients of the original mail.
`headersTruncated`	This object is only present if you configured the notification settings to include the headers from the original email. Indicates whether the headers are truncated in the notification. SES truncates the headers in the notification when the headers from the original message are 10 KB or larger in size. Possible values are `true` and `false`.

Field name	Description
`headers`	This object is only present if you configured the notification settings to include the headers from the original email. A list of the email's original headers. Each header in the list has a `name` field and a `value` field. **Note** Any message ID within the `headers` object is from the original message that you passed to SES. The message ID that SES subsequently assigned to the message is in the `messageId` field of the `mail` object.

Field name	Description
commonHeaders	This object is only present if you configured the notification settings to include the headers from the original email. Includes information about common email headers from the original email, including the From, To, and Subject fields. Within this object, each header is a key. The From and To fields are represented by arrays that can contain multiple values. **Note** For events, any message ID within the commonHeaders field is the message ID that Amazon SES subsequen tly assigned to the message in the messageId field of the mail object. Notifications will contain the message ID of the original email.

The following is an example of a `mail` object that includes the original email headers. When this notification type is not configured to include the original email headers, the `mail` object does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
{
   "timestamp":"2018-10-08T14:05:45 +0000",
   "messageId":"000001378603177f-7a5433e7-8edb-42ae-af10-f0181f34d6ee-000000",
   "source":"sender@example.com",
   "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/example.com",
   "sourceIp": "127.0.3.0",
   "sendingAccountId":"123456789012",
   "destination":[
      "recipient@example.com"
   ],
   "headersTruncated":false,
```

```
   "headers":[
      {
         "name":"From",
         "value":"\"Sender Name\" <sender@example.com>"
      },
      {
         "name":"To",
         "value":"\"Recipient Name\" <recipient@example.com>"
      },
      {
         "name":"Message-ID",
         "value":"custom-message-ID"
      },
      {
         "name":"Subject",
         "value":"Hello"
      },
      {
         "name":"Content-Type",
         "value":"text/plain; charset=\"UTF-8\""
      },
      {
         "name":"Content-Transfer-Encoding",
         "value":"base64"
      },
      {
         "name":"Date",
         "value":"Mon, 08 Oct 2018 14:05:45 +0000"
      }
   ],
   "commonHeaders":{
      "from":[
         "Sender Name <sender@example.com>"
      ],
      "date":"Mon, 08 Oct 2018 14:05:45 +0000",
      "to":[
         "Recipient Name <recipient@example.com>"
      ],
      "messageId":" custom-message-ID",
      "subject":"Message sent using SES"
   }
}
```

Bounce object

The JSON object that contains information about bounces contains the following fields.

Field name	Description
`bounceType`	The type of bounce, as determined by SES. For more information, see Bounce types.
`bounceSubType`	The subtype of the bounce, as determined by SES. For more information, see Bounce types.
`bouncedRecipients`	A list that contains information about the recipients of the original mail that bounced. For more information, see Bounced recipients.
`timestamp`	The date and time at which the bounce was sent (in ISO8601 format). Note that this is the time at which the notification was sent by the ISP, and not the time at which it was received by SES.
`feedbackId`	A unique ID for the bounce.

If SES was able to contact the remote Message Transfer Authority (MTA), the following field is also present.

Field name	Description
`remoteMtaIp`	The IP address of the MTA to which SES attempted to deliver the email.

If a delivery status notification (DSN) was attached to the bounce, the following field is also present.

Field name	Description
`reportingMTA`	The value of the `Reporting-MTA` field from the DSN. This is the value of the MTA that attempted to perform the delivery, relay, or gateway operation described in the DSN.

The following is an example of a `bounce` object.

```
{
   "bounceType":"Permanent",
   "bounceSubType": "General",
   "bouncedRecipients":[
      {
         "status":"5.0.0",
         "action":"failed",
         "diagnosticCode":"smtp; 550 user unknown",
         "emailAddress":"recipient1@example.com"
      },
      {
         "status":"4.0.0",
         "action":"delayed",
         "emailAddress":"recipient2@example.com"
      }
   ],
   "reportingMTA": "example.com",
   "timestamp":"2012-05-25T14:59:38.605Z",
   "feedbackId":"000001378603176d-5a4b5ad9-6f30-4198-a8c3-b1eb0c270a1d-000000",
   "remoteMtaIp":"127.0.2.0"
}
```

Bounced recipients

A bounce notification may pertain to a single recipient or to multiple recipients. The `bouncedRecipients` field holds a list of objects—one per recipient to whom the bounce notification pertains—and always contains the following field.

Field name	Description
emailAddress	The email address of the recipient. If a DSN is available, this is the value of the Final-Recipient field from the DSN.

Optionally, if a DSN is attached to the bounce, the following fields may also be present.

Field name	Description
action	The value of the Action field from the DSN. This indicates the action performed by the Reporting-MTA as a result of its attempt to deliver the message to this recipient.
status	The value of the Status field from the DSN. This is the per-recipient transport-independent status code that indicates the delivery status of the message.
diagnosticCode	The status code issued by the reporting MTA. This is the value of the Diagnostic-Code field from the DSN. This field may be absent in the DSN (and therefore also absent in the JSON).

The following is an example of an object that might be in the bouncedRecipients list.

```
{
    "emailAddress": "recipient@example.com",
    "action": "failed",
    "status": "5.0.0",
    "diagnosticCode": "X-Postfix; unknown user"
}
```

Bounce types

The bounce object contains a bounce type of `Undetermined`, `Permanent` *(hard)*, or `Transient` *(soft)*. The `Permanent` *(hard)* and `Transient` *(soft)* bounce types can also contain one of several bounce subtypes.

When you receive a bounce notification with a bounce type of `Transient` *(soft)*, you might be able to send email to that recipient in the future if the issue that caused the message to bounce is resolved.

When you receive a bounce notification with a bounce type of `Permanent` *(hard)*, it's unlikely that you'll be able to send email to that recipient in the future. For this reason, you should immediately remove the recipient whose address produced the bounce from your mailing lists.

> **Note**
>
> When a *soft bounce* (a bounce related to a temporary issue, such as the recipient's inbox being full) occurs, SES attempts to redeliver the email for a certain period of time. At the end of that period of time, if SES still can't deliver the email, it stops trying.
> SES provides notifications for hard bounces, and for soft bounces that it stopped trying to deliver. If you want to receive a notification each time a soft bounce occurs, enable event publishing and configure it to send notifications when delivery delay events occur.

bounceType	bounceSubType	Description
`Undetermined`	`Undetermined`	The recipient's email provider sent a bounce message. The bounce message didn't contain enough information for SES to determine the reason for the bounce. The bounce email, which was sent to the address in the Return-Path header of the email that resulted in the bounce, might contain additional informati on about the issue that caused the email to bounce.
`Permanent`	`General`	The recipient's email provider sent a hard bounce message.

bounceType	bounceSubType	Description
		Important When you receive this type of bounce notification, you should immediately remove the recipient's email address from your mailing list. Sending messages to addresses that produce hard bounces can have a negative impact on your reputation as a sender. If you continue sending email to addresses that produce hard bounces, we might pause your ability to send additional email. See the section called "Using the account-level suppression list".
Permanent	NoEmail	It was not possible to retrieve the recipient email address from the bounce message.
Permanent	Suppressed	The recipient's email address is on the SES suppression list because it has a recent history of producing hard bounces. To override the global suppression list, see Using the Amazon SES account-level suppression list.
Permanent	OnAccountSuppressionList	SES has suppressed sending to this address because it is on the account-level suppression list. This does not count toward your bounce rate metric.
Permanent	OnTenantSuppressionList	SES has suppressed sending to this address because it is on the tenant-level suppression list. This does not count toward your bounce rate metric.

bounceType	bounceSubType	Description
Permanent	UnsubscribedRecipient	This bounce type occurs when the recipient contact has unsubscribed from the topic and a mail is sent to them using list management options. SES respects the contact preference and doesn't attempt delivery. Also, this bounce doesn't impact sender reputation since the delivery was not attempted, nor is the recipient contact added to a suppression list due to the bounce. **Tip** It's recommended that you subscribe to UnsubscribedRecipient events to avoid continued sending to to unsubscribed recipients. Consider the section called “Using list management”. List management should be the source of truth for your subscriber list. From the perspective of SES enforcement, if you continue to send to suppressed or unsubscribed recipients, you'll have the reputation of not adhering to best practices for email sending.
Permanent	EmailValidationSuppressed	SES has suppressed sending to this address because the address did not meet the threshold for your email validation settings. This does not count toward your bounce rate metric.

bounceType	bounceSubType	Description
`Transient`	`General`	The recipient's email provider sent a general bounce message. You might be able to send a message to the same recipient in the future if the issue that caused the message to bounce is resolved. **Note** If you send an email to a recipient who has an active automatic response rule (such as an "out of the office" message), you might receive this type of notification. Even though the response has a notification type of `Bounce`, SES doesn't count automatic responses when it calculates the bounce rate for your account.
`Transient`	`MailboxFull`	The recipient's email provider sent a bounce message because the recipient's inbox was full. You might be able to send to the same recipient in the future when the mailbox is no longer full.
`Transient`	`MessageTooLarge`	The recipient's email provider sent a bounce message because message you sent was too large. You might be able to send a message to the same recipient if you reduce the size of the message.

bounceType	bounceSubType	Description
Transient	ContentRejected	The recipient's email provider sent a bounce message because the message you sent contains content that the provider doesn't allow. You might be able to send a message to the same recipient if you change the content of the message.
Transient	AttachmentRejected	The recipient's email provider sent a bounce message because the message contained an unacceptable attachment. For example, some email providers may reject messages with attachments of a certain file type, or messages with very large attachments. You might be able to send a message to the same recipient if you remove or change the content of the attachment.

Complaint object

The JSON object that contains information about complaints has the following fields.

Field name	Description
complainedRecipients	A list that contains information about recipients that may have been responsible for the complaint. For more information, see Complained recipients.
timestamp	The date and time when the ISP sent the complaint notification, in ISO 8601 format. The date and time in this field might not be the same as the date and time when SES received the notification.
feedbackId	A unique ID associated with the complaint.

Field name	Description
complaintSubType	The value of the complaintSubType field can either be null or OnAccountSuppressionList. If the value is OnAccountSuppressionList, SES accepted the message, but didn't attempt to send it because it was on the account-level suppression list. The value can also be OnTenantSuppressionList if the address was on the tenant-level suppression list.

Further, if a feedback report is attached to the complaint, the following fields may be present.

Field name	Description
userAgent	The value of the User-Agent field from the feedback report. This indicates the name and version of the system that generated the report.
complaintFeedbackType	The value of the Feedback-Type field from the feedback report received from the ISP. This contains the type of feedback.
arrivalDate	The value of the Arrival-Date or Received-Date field from the feedback report (in ISO8601 format). This field may be absent in the report (and therefore also absent in the JSON).

The following is an example of a complaint object.

```
{
   "userAgent":"ExampleCorp Feedback Loop (V0.01)",
   "complainedRecipients":[
```

```
      {
         "emailAddress":"recipient1@example.com"
      }
   ],
   "complaintFeedbackType":"abuse",
   "arrivalDate":"2009-12-03T04:24:21.000-05:00",
   "timestamp":"2012-05-25T14:59:38.623Z",
   "feedbackId":"000001378603177f-18c07c78-fa81-4a58-9dd1-fedc3cb8f49a-000000"
}
```

Complained recipients

The `complainedRecipients` field contains a list of recipients that may have submitted the complaint. You should use this information to determine which recipient submitted the complaint, and then immediately remove that recipient from your mailing lists.

> **Important**
>
> Most ISPs remove the email address of the recipient who submitted the complaint from their complaint notification. For this reason, this list contains information about recipients who might have sent the complaint, based on the recipients of the original message and the ISP from which we received the complaint. SES performs a lookup against the original message to determine this recipient list.

JSON objects in this list contain the following field.

Field name	Description
`emailAddress`	The email address of the recipient.

The following is an example of a complained recipient object.

```
{ "emailAddress": "recipient1@example.com" }
```

> **Note**
>
> Because of this behavior, you can be more certain that you know which email address complained about your message if you limit your sending to one message per recipient (rather than sending one message with 30 different email addresses in the bcc line).

Complaint types

You may see the following complaint types in the `complaintFeedbackType` field as assigned by the reporting ISP, according to the Internet Assigned Numbers Authority website:

- `abuse`—Indicates unsolicited email or some other kind of email abuse.
- `auth-failure`—Email authentication failure report.
- `fraud`—Indicates some kind of fraud or phishing activity.
- `not-spam`—Indicates that the entity providing the report does not consider the message to be spam. This may be used to correct a message that was incorrectly tagged or categorized as spam.
- `other`—Indicates any other feedback that does not fit into other registered types.
- `virus`—Reports that a virus is found in the originating message.

Delivery object

The JSON object that contains information about deliveries always has the following fields.

Field name	Description
`timestamp`	The time SES delivered the email to the recipient's mail server (in ISO8601 format).
`processingTimeMillis`	The time in milliseconds between when SES accepted the request from the sender to passing the message to the recipient's mail server.
`recipients`	A list of the intended recipients of the email to which the delivery notification applies.

Field name	Description
`smtpResponse`	The SMTP response message of the remote ISP that accepted the email from SES. This message varies by email, by receiving mail server, and by receiving ISP.
`reportingMTA`	The hostname of the SES mail server that sent the mail.
`remoteMtaIp`	The IP address of the MTA to which SES delivered the email.

The following is an example of a `delivery` object.

```
{
   "timestamp":"2014-05-28T22:41:01.184Z",
   "processingTimeMillis":546,
   "recipients":["success@simulator.amazonses.com"],
   "smtpResponse":"250 ok:  Message 64111812 accepted",
   "reportingMTA":"a8-70.smtp-out.amazonses.com",
   "remoteMtaIp":"127.0.2.0"
}
```

Amazon SNS notification examples for Amazon SES

The following sections provide examples of the three types of notifications:

- For bounce notification examples, see Amazon SNS bounce notification examples.
- For complaint notification examples, see Amazon SNS complaint notification examples.
- For delivery notification examples, see Amazon SNS delivery notification example.

Amazon SNS bounce notification examples

This section contains examples of bounce notifications with and without a Delivery Status Notification (DSN) provided by the email receiver that sent the feedback.

Bounce notification with a DSN

The following is an example of a bounce notification that contains a DSN and the original email headers. When bounce notifications are not configured to include the original email headers, the `mail` object within the notifications does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
   {
       "notificationType":"Bounce",
       "bounce":{
          "bounceType":"Permanent",
          "reportingMTA":"dns; email.example.com",
          "bouncedRecipients":[
             {
                "emailAddress":"jane@example.com",
                "status":"5.1.1",
                "action":"failed",
                "diagnosticCode":"smtp; 550 5.1.1 <jane@example.com>... User"
             }
          ],
          "bounceSubType":"General",
          "timestamp":"2016-01-27T14:59:38.237Z",
          "feedbackId":"00000138111222aa-33322211-cccc-cccc-cccc-ddddaaaa068a-000000",
          "remoteMtaIp":"127.0.2.0"
       },
       "mail":{
          "timestamp":"2016-01-27T14:59:38.237Z",
          "source":"john@example.com",
          "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
          "sourceIp": "127.0.3.0",
          "sendingAccountId":"123456789012",
          "callerIdentity": "IAM_user_or_role_name",
          "messageId":"00000138111222aa-33322211-cccc-cccc-cccc-ddddaaaa0680-000000",
          "destination":[
            "jane@example.com",
            "mary@example.com",
            "richard@example.com"],
          "headersTruncated":false,
          "headers":[
           {
             "name":"From",
             "value":"\"John Doe\" <john@example.com>"
```

```
          },
          {
            "name":"To",
            "value":"\"Jane Doe\" <jane@example.com>, \"Mary Doe\" <mary@example.com>,
\"Richard Doe\" <richard@example.com>"
          },
          {
            "name":"Message-ID",
            "value":"custom-message-ID"
          },
          {
            "name":"Subject",
            "value":"Hello"
          },
          {
            "name":"Content-Type",
            "value":"text/plain; charset=\"UTF-8\""
          },
          {
            "name":"Content-Transfer-Encoding",
            "value":"base64"
          },
          {
            "name":"Date",
            "value":"Wed, 27 Jan 2016 14:05:45 +0000"
          }
         ],
         "commonHeaders":{
            "from":[
               "John Doe <john@example.com>"
            ],
            "date":"Wed, 27 Jan 2016 14:05:45 +0000",
            "to":[
               "Jane Doe <jane@example.com>, Mary Doe <mary@example.com>, Richard Doe
<richard@example.com>"
            ],
            "messageId":"custom-message-ID",
            "subject":"Hello"
         }
      }
   }
```

Bounce notification without a DSN

The following is an example of a bounce notification that includes the original email headers but does not include a DSN. When bounce notifications are not configured to include the original email headers, the `mail` object within the notifications does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
   {
      "notificationType":"Bounce",
      "bounce":{
         "bounceType":"Permanent",
         "bounceSubType": "General",
         "bouncedRecipients":[
            {
               "emailAddress":"jane@example.com"
            },
            {
               "emailAddress":"richard@example.com"
            }
         ],
         "timestamp":"2016-01-27T14:59:38.237Z",
         "feedbackId":"00000137860315fd-869464a4-8680-4114-98d3-716fe35851f9-000000",
         "remoteMtaIp":"127.0.2.0"
      },
      "mail":{
         "timestamp":"2016-01-27T14:59:38.237Z",
         "messageId":"00000137860315fd-34208509-5b74-41f3-95c5-22c1edc3c924-000000",
         "source":"john@example.com",
         "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
         "sourceIp": "127.0.3.0",
         "sendingAccountId":"123456789012",
         "callerIdentity": "IAM_user_or_role_name",
         "destination":[
            "jane@example.com",
            "mary@example.com",
            "richard@example.com"
         ],
        "headersTruncated":false,
        "headers":[
         {
            "name":"From",
            "value":"\"John Doe\" <john@example.com>"
```

```
        },
        {
           "name":"To",
           "value":"\"Jane Doe\" <jane@example.com>, \"Mary Doe\" <mary@example.com>,
\"Richard Doe\" <richard@example.com>"
        },
        {
           "name":"Message-ID",
           "value":"custom-message-ID"
        },
        {
           "name":"Subject",
           "value":"Hello"
        },
        {
           "name":"Content-Type",
           "value":"text/plain; charset=\"UTF-8\""
        },
        {
           "name":"Content-Transfer-Encoding",
           "value":"base64"
        },
        {
           "name":"Date",
           "value":"Wed, 27 Jan 2016 14:05:45 +0000"
         }
        ],
        "commonHeaders":{
          "from":[
             "John Doe <john@example.com>"
          ],
          "date":"Wed, 27 Jan 2016 14:05:45 +0000",
          "to":[
             "Jane Doe <jane@example.com>, Mary Doe <mary@example.com>, Richard Doe
<richard@example.com>"
          ],
          "messageId":"custom-message-ID",
          "subject":"Hello"
        }
     }
 }
```

Amazon SNS complaint notification examples

This section contains examples of complaint notifications, with and without a feedback report, provided by the email receiver that sent the feedback.

Complaint notification with a feedback report

The following is an example of a complaint notification that contains a feedback report and the original email headers. When complaint notifications are not configured to include the original email headers, the `mail` object within the notifications does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
   {
      "notificationType":"Complaint",
      "complaint":{
         "userAgent":"AnyCompany Feedback Loop (V0.01)",
         "complainedRecipients":[
            {
               "emailAddress":"richard@example.com"
            }
         ],
         "complaintFeedbackType":"abuse",
         "arrivalDate":"2016-01-27T14:59:38.237Z",
         "timestamp":"2016-01-27T14:59:38.237Z",
         "feedbackId":"000001378603177f-18c07c78-fa81-4a58-9dd1-fedc3cb8f49a-000000"
      },
      "mail":{
         "timestamp":"2016-01-27T14:59:38.237Z",
         "messageId":"000001378603177f-7a5433e7-8edb-42ae-af10-f0181f34d6ee-000000",
         "source":"john@example.com",
         "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
         "sourceIp": "127.0.3.0",
         "sendingAccountId":"123456789012",
         "callerIdentity": "IAM_user_or_role_name",
         "destination":[
            "jane@example.com",
            "mary@example.com",
            "richard@example.com"
         ],
          "headersTruncated":false,
          "headers":[
           {
```

```
            "name":"From",
            "value":"\"John Doe\" <john@example.com>"
          },
          {
            "name":"To",
            "value":"\"Jane Doe\" <jane@example.com>, \"Mary Doe\" <mary@example.com>,
\"Richard Doe\" <richard@example.com>"
          },
          {
            "name":"Message-ID",
            "value":"custom-message-ID"
          },
          {
            "name":"Subject",
            "value":"Hello"
          },
          {
            "name":"Content-Type",
            "value":"text/plain; charset=\"UTF-8\""
          },
          {
            "name":"Content-Transfer-Encoding",
            "value":"base64"
          },
          {
            "name":"Date",
            "value":"Wed, 27 Jan 2016 14:05:45 +0000"
          }
        ],
        "commonHeaders":{
          "from":[
             "John Doe <john@example.com>"
          ],
          "date":"Wed, 27 Jan 2016 14:05:45 +0000",
          "to":[
             "Jane Doe <jane@example.com>, Mary Doe <mary@example.com>, Richard Doe
<richard@example.com>"
          ],
          "messageId":"custom-message-ID",
          "subject":"Hello"
        }
     }
  }
```

Complaint notification without a feedback report

The following is an example of a complaint notification that includes the original email headers but does not include a feedback report. When complaint notifications are not configured to include the original email headers, the `mail` object within the notifications does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
   {
      "notificationType":"Complaint",
      "complaint":{
         "complainedRecipients":[
            {
               "emailAddress":"richard@example.com"
            }
         ],
         "timestamp":"2016-01-27T14:59:38.237Z",
         "feedbackId":"0000013786031775-fea503bc-7497-49e1-881b-a0379bb037d3-000000"
      },
      "mail":{
         "timestamp":"2016-01-27T14:59:38.237Z",
         "messageId":"0000013786031775-163e3910-53eb-4c8e-a04a-f29debf88a84-000000",
         "source":"john@example.com",
         "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
         "sourceIp": "127.0.3.0",
         "sendingAccountId":"123456789012",
         "callerIdentity": "IAM_user_or_role_name",
         "destination":[
            "jane@example.com",
            "mary@example.com",
            "richard@example.com"
         ],
         "headersTruncated":false,
         "headers":[
          {
            "name":"From",
            "value":"\"John Doe\" <john@example.com>"
          },
          {
            "name":"To",
            "value":"\"Jane Doe\" <jane@example.com>, \"Mary Doe\" <mary@example.com>,
 \"Richard Doe\" <richard@example.com>"
          },
```

```
          {
            "name":"Message-ID",
            "value":"custom-message-ID"
          },
          {
            "name":"Subject",
            "value":"Hello"
          },
          {
            "name":"Content-Type",
            "value":"text/plain; charset=\"UTF-8\""
          },
          {
            "name":"Content-Transfer-Encoding",
            "value":"base64"
          },
          {
            "name":"Date",
            "value":"Wed, 27 Jan 2016 14:05:45 +0000"
          }
          ],
          "commonHeaders":{
             "from":[
                "John Doe <john@example.com>"
             ],
             "date":"Wed, 27 Jan 2016 14:05:45 +0000",
             "to":[
                "Jane Doe <jane@example.com>, Mary Doe <mary@example.com>, Richard Doe
<richard@example.com>"
             ],
             "messageId":"custom-message-ID",
             "subject":"Hello"
          }
       }
    }
```

Amazon SNS delivery notification example

The following is an example of a delivery notification that includes the original email headers. When delivery notifications are not configured to include the original email headers, the `mail` object within the notifications does not include the `headersTruncated`, `headers`, and `commonHeaders` fields.

```
   {
      "notificationType":"Delivery",
      "mail":{
         "timestamp":"2016-01-27T14:59:38.237Z",
         "messageId":"0000014644fe5ef6-9a483358-9170-4cb4-a269-f5dcdf415321-000000",
         "source":"john@example.com",
         "sourceArn": "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
         "sourceIp": "127.0.3.0",
         "sendingAccountId":"123456789012",
         "callerIdentity": "IAM_user_or_role_name",
         "destination":[
            "jane@example.com"
         ],
          "headersTruncated":false,
          "headers":[
           {
              "name":"From",
              "value":"\"John Doe\" <john@example.com>"
           },
           {
              "name":"To",
              "value":"\"Jane Doe\" <jane@example.com>"
           },
           {
              "name":"Message-ID",
              "value":"custom-message-ID"
           },
           {
              "name":"Subject",
              "value":"Hello"
           },
           {
              "name":"Content-Type",
              "value":"text/plain; charset=\"UTF-8\""
           },
           {
              "name":"Content-Transfer-Encoding",
              "value":"base64"
           },
           {
              "name":"Date",
              "value":"Wed, 27 Jan 2016 14:58:45 +0000"
```

```
          }
         ],
         "commonHeaders":{
           "from":[
              "John Doe <john@example.com>"
           ],
           "date":"Wed, 27 Jan 2016 14:58:45 +0000",
           "to":[
              "Jane Doe <jane@example.com>"
           ],
           "messageId":"custom-message-ID",
           "subject":"Hello"
         }
      },
     "delivery":{
        "timestamp":"2016-01-27T14:59:38.237Z",
        "recipients":["jane@example.com"],
        "processingTimeMillis":546,
        "reportingMTA":"a8-70.smtp-out.amazonses.com",
        "smtpResponse":"250 ok:  Message 64111812 accepted",
        "remoteMtaIp":"127.0.2.0"
     }
  }
```

Using identity authorization in Amazon SES

Identity authorization policies define how individual verified identities can use Amazon SES by specifying which SES API actions are allowed or denied for the identity and under what conditions.

Through the use of these authorization polices, you can maintain control over your identities by changing or revoking permissions at any time. You can even authorize other users to use the identities that you own (domains or email addresses) with their own SES accounts.

Topics

- Amazon SES policy anatomy
- Creating an identity authorization policy in Amazon SES
- Identity policy examples in Amazon SES
- Managing your identity authorization policies in Amazon SES

Amazon SES policy anatomy

Policies adhere to a specific structure, contain elements, and must meet certain requirements.

Policy structure

Each authorization policy is a JSON document that is attached to an identity. Each policy includes the following sections:

- Policy-wide information at the top of the document.
- One or more individual statements, each of which describes a set of permissions.

The following example policy grants AWS account ID *123456789012* permissions specified in the *Action* section for the verified domain *example.com*.

JSON

```
{
  "Id":"ExampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeAccount",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:GetEmailIdentity",
        "ses:UpdateEmailIdentityPolicy",
        "ses:ListRecommendations",
        "ses:CreateEmailIdentityPolicy",
        "ses:DeleteEmailIdentity"
      ]
    }
  ]
}
```

You can find more authorization policy examples at Identity policy examples.

Policy elements

This section describes the elements contained in identity authorization policies. First we describe policy-wide elements, and then we describe elements that apply only to the statement in which they are included. We follow with a discussion of how to add conditions to your statements.

For specific information about the syntax of the elements, see Grammar of the IAM Policy Language in the *IAM User Guide*.

Policy-wide information

There are two policy-wide elements: `Id` and `Version`. The following table provides information about these elements.

Name	Description	Required	Valid values
`Id`	Uniquely identifies the policy.	No	Any string
`Version`	Specifies the policy access language version.	No	Any string. As a best practice, we recommend that you include this field with a value of "2012-10-17".

Statements specific to the policy

Identity authorization policies require at least one statement. Each statement can include the elements described in the following table.

Name	Description	Required	Valid values
`Sid`	Uniquely identifies the statement.	No	Any string.
`Effect`	Specifies the result that you want the policy statement to	Yes	"Allow" or "Deny".

Name	Description	Required	Valid values
	return at evaluation time.		
`Resource`	Specifies the identity to which the policy applies. (For sending authorization, this is the email address or domain that the identity owner is authorizing the delegate sender to use.)	Yes	The Amazon Resource Name (ARN) of the identity.

Name	Description	Required	Valid values
`Principal`	Specifies the AWS account, user, or AWS service that receives the permission in the statement.	Yes	A valid AWS account ID, user ARN, or AWS service. AWS account IDs and user ARNs are specified using `"AWS"` (for example, `"AWS": ["123456789012"]` or `"AWS": ["arn:aws-eusc:iam::123456789012:root"]`). AWS service names are specified using `"Service"` (for example, `"Service": ["cognito-idp.amazonaws.com"]`). For examples of the format of user ARNs, see the AWS General Reference.

Name	Description	Required	Valid values
`Action`	Specifies the action that the statement applies to.	Yes	"ses:BatchGetMetricData", "ses:CancelExportJob", "ses:CreateDeliverabilityTestReport", "ses:CreateEmailIdentityPolicy", "ses:CreateExportJob", "ses:DeleteEmailIdentity", "ses:DeleteEmailIdentityPolicy", "ses:GetDomainStatisticsReport", "ses:GetEmailIdentity","ses:GetEmailIdentityPolicies", "ses:GetExportJob", "ses:ListExportJobs", "ses:ListRecommendations", "ses:PutEmailIdentityConfigurationSetAttributes", "ses:PutEmailIdentityDkimAttributes", "ses:PutEmailIdentityDkimSigningAttributes", "ses:PutEmailIdentityFeedbackAttributes", "ses:PutEmailIdentityMailFromAttributes", "ses:TagResource", "ses:Unta

Name	Description	Required	Valid values
			gResource", "ses:UpdateEmailIdentityPolicy" (Sending authorization actions: "ses:SendEmail", "ses:SendRawEmail", "ses:SendTemplatedEmail", "ses:SendBulkTemplatedEmail") You can specify one or more of these operations.
`Condition`	Specifies any restrictions or details about the permission.	No	See the information about conditions following this table.

Conditions

A *condition* is any restriction about the permission in the statement. The part of the statement that specifies the conditions can be the most detailed of all the parts. A *key* is the specific characteristic that's the basis for access restriction, such as the date and time of the request.

You use both conditions and keys together to express the restriction. For example, if you want to restrict the delegate sender from making requests to Amazon SES on your behalf after July 30, 2019, you use the condition called `DateLessThan`. You use the key called `aws:CurrentTime` and set it to the value `2019-07-30T00:00:00Z`.

SES implements only the following AWS-wide policy keys:

- `aws:CurrentTime`
- `aws:EpochTime`

- `aws:SecureTransport`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

For more information about these keys, see the IAM User Guide.

Policy requirements

Policies must meet all of the following requirements:

- Each policy has to include at least one statement.
- Each policy has to include at least one valid principal.
- Each policy has to specify one resource, and that resource has to be the ARN of the identity that the policy is attached to.
- Identity owners can associate up to 20 policies with each unique identity.
- Policies can't exceed 4 kilobytes (KB) in size.
- Policy names can't exceed 64 characters. Additionally, they can only include alphanumeric characters, dashes, and underscores.

Creating an identity authorization policy in Amazon SES

An identity authorization policy is comprised of statements specifying what API actions are allowed or denied for an identity and under what conditions.

To authorize an Amazon SES domain or email address identity that you own, you create an authorization policy, and then attach that policy to the identity. An identity can have zero, one, or many policies. However, a single policy can only be associated with a single identity.

For a list of API actions that can be used in an identity authorization policy, see the *Action* row in the the section called “Statements specific to the policy” table.

You can create an identity authorization policy in the following ways:

- **By using the policy generator** – You can create a simple policy by using the policy generator in the SES console. In addition to allowing or denying permissions on SES API actions, you can constrain the actions with conditions. You can also use the policy generator to quickly create the basic structure of a policy and then customize it later by editing the policy.
- **By creating a custom policy** – If you want to include more advanced conditions or use an AWS service as the principal, you can create a custom policy and attach it to the identity by using the SES console or the SES API.

Topics

- Using the policy generator
- Creating a custom policy

Using the policy generator

You can use the policy generator to create a simple authorization policy by following these steps.

To create a policy by using the policy generator

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the **Identities** container on the **Identities** screen, select the verified identity you wish to create an authorization policy for.
4. In the details screen of the verified identity you selected in the previous step, choose the **Authorization** tab.
5. In the **Authorization policies** pane, choose **Create policy** and select **Use policy generator** from the dropdown.
6. In the **Create statement** pane, choose **Allow** in the **Effect** field. (If you want to create a policy to restrict this identity, choose **Deny** instead.)
7. In the **Principals** field, enter the *AWS account ID*, *IAM user ARN*, or AWS service to receive the permissions you want to authorize for this identity, then choose **Add**. (If you wish to authorize more than one, repeat this step for each one.)
8. In the **Actions** field, select the check box for each action you would like to authorize for your principals.

9. (Optional) Expand **Specify conditions** if you wish to add a qualifying statement to the permission.

    a. Select an operator from the **Operator** dropdown.

    b. Select a type from the **Key** dropdown.

    c. Respective to the key type you selected, enter its value in the **Value** field. (If you wish to add more conditions, choose **Add new condition** and repeat this step for each additional one.)

10. Choose **Save statement**.
11. (Optional) Expand **Create another statement** if you wish to add more statements to your policy and repeat steps 6 - 10.
12. Choose **Next** and on the **Customize policy** screen, the **Edit policy details** container has fields where you can change or customize the policy's **Name** and the **Policy document** itself.
13. Choose **Next** and on the **Review and apply** screen, the **Overview** container will show the verified identity you're authorizing as well as the name of this policy. In the **Policy document** pane will be the actual policy you just wrote along with any conditions you added - review the policy and if it looks correct, choose **Apply policy**. (If you need to change or correct something, choose **Previous** and work in the **Edit policy details** container.)

Creating a custom policy

If you want to create a custom policy and attach it to an identity, you have the following options:

- **Using the Amazon SES API** – Create a policy in a text editor and then attach the policy to the identity by using the `PutIdentityPolicy` API described in the Amazon Simple Email Service API Reference.
- **Using the Amazon SES console** – Create a policy in a text editor and attach it to an identity by pasting it into the custom policy editor in the Amazon SES console. The following procedure describes this method.

To create a custom policy by using the custom policy editor

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.

3. In the **Identities** container on the **Identities** screen, select the verified identity you wish to create an authorization policy for.
4. In the details screen of the verified identity you selected in the previous step, choose the **Authorization** tab.
5. In the **Authorization policies** pane, choose **Create policy** and select **Create custom policy** from the dropdown.
6. In the **Policy document** pane, type or paste the text of your policy in JSON format. You can also use the policy generator to quickly create the basic structure of a policy and then customize it here.
7. Choose **Apply Policy**. (If you ever need to modify your custom policy, just select its check box under the **Authorization** tab, choose **Edit**, and make your changes in the **Policy document** pane followed by **Save changes**).

Identity policy examples in Amazon SES

Identity authorization enables you to specify the fine-grained conditions under which you allow or deny API actions for an identity.

The following examples show you how to write policies to control different aspects API actions:

- Specifying the principal
- Restricting the action
- Using multiple statements

Specifying the principal

The *principal*, which is the entity to which you are granting permission, can be an AWS account, an AWS Identity and Access Management (IAM) user, or an AWS service that belongs to the same account.

The following example shows a simple policy that allows AWS ID *123456789012* to control the verified identity *example.com* which is also owned by AWS account *123456789012*.

JSON

```
{
  "Id":"SampleAuthorizationPolicy",
```

```
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeMarketer",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:DeleteEmailIdentity",
        "ses:PutEmailIdentityDkimSigningAttributes"
      ]
    }
  ]
}
```

The following example policy grants permission to two users to control the verified identity *example.com*. Users are specified by their Amazon Resource Name (ARN).

JSON

```
{
  "Id":"ExampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeIAMUser",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com",
      "Principal":{
        "AWS":[
          "arn:aws-eusc:iam::123456789012:user/John",
          "arn:aws-eusc:iam::123456789012:user/Jane"
        ]
      },
      "Action":[
```

```
        "ses:DeleteEmailIdentity",
        "ses:PutEmailIdentityDkimSigningAttributes"
      ]
    }
  ]
}
```

Restricting the action

There are multiple actions that can be specified in an identity authorization policy depending on the level of control you want to authorize:

```
"BatchGetMetricData",
"ListRecommendations",
"CreateDeliverabilityTestReport",
"CreateEmailIdentityPolicy",
"DeleteEmailIdentity",
"DeleteEmailIdentityPolicy",
"GetDomainStatisticsReport",
"GetEmailIdentity",
"GetEmailIdentityPolicies",
"PutEmailIdentityConfigurationSetAttributes",
"PutEmailIdentityDkimAttributes",
"PutEmailIdentityDkimSigningAttributes",
"PutEmailIdentityFeedbackAttributes",
"PutEmailIdentityMailFromAttributes",
"TagResource",
"UntagResource",
"UpdateEmailIdentityPolicy"
```

Identity authorization policies also enable you to restrict the principal to just one of those actions.

JSON

```
{
    "Id": "ExamplePolicy",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ControlAction",
            "Effect": "Allow",
```

```
            "Resource": "arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            },
            "Action": [
                "ses:PutEmailIdentityMailFromAttributes"
            ]
        }
    ]
}
```

Using multiple statements

Your identity authorization policy can include multiple statements. The following example policy has two statements. The first statement denies two users to access `getemailidentity` from *sender@example.com* within the same account `123456789012`. The second statement denies `UpdateEmailIdentityPolicy` for the principal, *Jack*, within the same account `123456789012`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"DenyGet",
      "Effect":"Deny",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
sender@example.com",
      "Principal":{
        "AWS":[
          "arn:aws-eusc:iam::123456789012:user/John",
          "arn:aws-eusc:iam::123456789012:user/Jane"
        ]
      },
      "Action":[
        "ses:GetEmailIdentity"
      ]
    },
```

```
    {
      "Sid":"DenyUpdate",
      "Effect":"Deny",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
sender@example.com",
      "Principal":{
        "AWS":"arn:aws-eusc:iam::123456789012:user/Jack"
      },
      "Action":[
        "ses:UpdateEmailIdentityPolicy"
      ]
    }
  ]
}
```

Managing your identity authorization policies in Amazon SES

In addition to creating and attaching policies to identities, you can edit, remove, list, and retrieve an identity's policies as described in the following sections.

Managing policies using the Amazon SES console

Managing Amazon SES polices entails viewing, editing, or deleting a policy attached to an identity by using the Amazon SES console.

To manage policies using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Verified identities**.
3. In the list of identities, choose the identity you want to manage.
4. On the identity's detail page, navigate to the **Authorization** tab. Here you'll find a list of all the policies attached to this identity.
5. Select the policy you want to manage by choosing its checkbox.
6. Depending on the desired management task, choose the respective button as follows:

    a. To view the policy, choose **View policy**. If you need a copy of it, choose the **Copy** button and it will be copied to your clipboard.

b. To edit the policy, choose **Edit**. In the **Policy document** pane, edit the policy, and then choose **Save changes**.

> **Note**
>
> To revoke permissions, you can either edit the policy or remove it.

c. To remove the policy, choose **Delete**.

> **Important**
>
> Removing a policy is permanent. We recommend that you back up the policy by copying and pasting it into a text file before you remove it.

Managing policies using the Amazon SES API

Managing Amazon SES polices entails viewing, editing, or deleting a policy attached to an identity by using the Amazon SES API.

To list and view policies using the Amazon SES API

- You can list the policies that are attached to an identity by using the ListIdentityPolicies API operation. You can also retrieve the policies themselves by using the GetIdentityPolicies API operation.

To edit a policy using the Amazon SES API

- You can edit a policy that's attached to an identity by using the PutIdentityPolicy API operation.

To delete a policy using the Amazon SES API

- You can delete a policy that's attached to an identity by using the DeleteIdentityPolicy API operation.

Using sending authorization with Amazon SES

You can configure Amazon SES to authorize other users to send emails from the identities that you own (domains or email addresses) using their own Amazon SES accounts. With the *sending authorization* feature, you can maintain control over your identities so that you can change or revoke permissions at any time. For example, if you're a business owner, you can use sending authorization to enable a third party (such as an email marketing company) to send email from a domain you own.

This chapter covers the specifics of sending authorization which replaces the legacy cross-account notifications feature. You should first understand the basics of identity based authorization using authorization policies as explained in Using identity authorization in Amazon SES which covers important topics such as the anatomy of an authorization policy and how to manage your polices.

Cross-account notifications legacy support

Feedback notifications for bounces, complaints, and deliveries associated with email sent from a delegate sender that's been authorized by an identity owner to send from one of his verified identities, have traditionally been configured using cross-account notifications where the delegate sender would associate a topic with an identity they didn't own (that's the cross-account). However, cross-account notifications have been replaced by using configuration sets and verified identities in association with delegate sending where the delegate sender has been authorized by the identity owner to use one of their verified identities to send email from. This new method allows the flexibility to configure bounce, complaint, delivery, and other event notifications by the following two constructs depending if you're the delegate sender or the owner of the verified identity:

- **Configuration sets** – The delegate sender can set up event publishing in his own configuration set that he can specify when sending email from a verified identity he doesn't own, but has been authorized to send from by the identity owner through an authorization policy. Event publishing allows bounce, complaint, delivery, and other event notifications to be published to Amazon CloudWatch, Amazon Data Firehose, Amazon Pinpoint, and Amazon SNS. See Create event destinations.
- **Verified identities** – Besides having the identity owner authorize the delegate sender to use one of his verified identities to send email from, he can also, at the request of the delegate sender, configure feedback notifications on the shared identity to use SNS topics owned by the delegate sender. Only the delegate sender will get these notifications because they own the SNS topic. See Step 14 for how to configure an "SNS topic you don't own" in the authorization policy procedures.

Note

For compatibility, cross-account notifications are being supported for legacy cross-account notifications currently being used in your account. This support is limited to being able to modify and use any current cross-accounts you created in the Amazon SES classic console; however, you can no longer create *new* cross-account notifications. To create new ones in the Amazon SES new console, use the new methods of delegate sending either with configuration sets using event publishing, or with verified identities configured with your own SNS topics.

Topics

- Overview of Amazon SES sending authorization
- Identity owner tasks for Amazon SES sending authorization
- Delegate sender tasks for Amazon SES sending authorization

Overview of Amazon SES sending authorization

This topic provides an overview of the sending authorization process and then explains how the email sending features of Amazon SES, such as sending quotas and notifications, work with sending authorization.

This section uses the following terms:

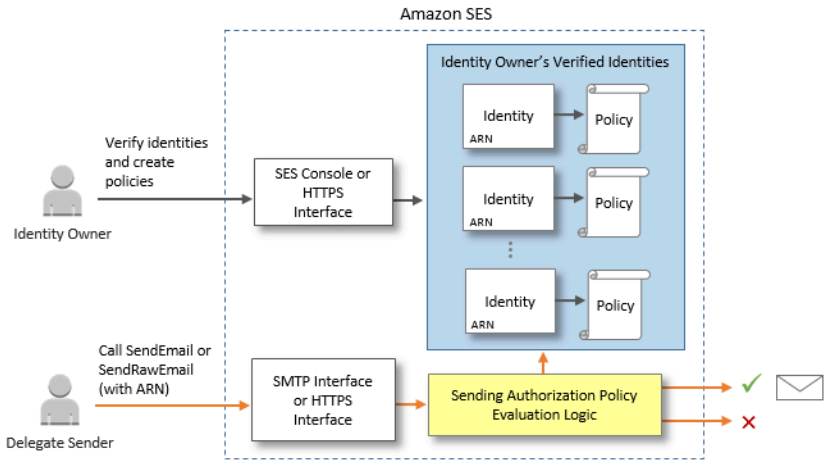
- **Identity** – An email address or domain that Amazon SES users use to send email.
- **Identity owner** – An Amazon SES user who has verified ownership of an email address or domain by using the procedures described in Verified identities.
- **Delegate sender** – An AWS account, an AWS Identity and Access Management (IAM) user, or an AWS service that's been authorized through an authorization policy to send email on behalf of the identity owner.
- **Sending authorization policy** – A document that you attach to an identity to specify who may send for that identity and under which conditions.
- **Amazon Resource Name (ARN)** – A standardized way to uniquely identify an AWS resource across all AWS services. For sending authorization, the resource is the identity that the identity owner has authorized the delegate sender to use. An example of an ARN is *arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/example.com*.

Sending authorization process

Sending authorization is based on sending authorization policies. If you want to enable a delegate sender to send on your behalf, you create a sending authorization policy and associate the policy to your identity by using the Amazon SES console or the Amazon SES API. When the delegate sender attempts to send an email through Amazon SES on your behalf, the delegate sender passes the ARN of your identity in the request or in the header of the email.

When Amazon SES receives the request to send the email, it checks your identity's policy (if present) to determine if you have authorized the delegate sender to send on the identity's behalf. If the delegate sender is authorized, Amazon SES accepts the email; otherwise, Amazon SES returns an error message.

The following diagram shows the high-level relationship between sending authorization concepts:



The sending authorization process consists of the following steps:

1. The identity owner selects a verified identity for the delegate sender to use. (If you haven't verified an identity, see Verified identities.)

   > **Note**
   >
   > The verified identity you choose for your delegate sender cannot have a default configuration set assigned to it.

2. The delegate sender lets the identity owner know which AWS account ID or IAM user ARN they want to use for sending.

3. If the identity owner agrees to allow the delegate sender to send from one of the owner's accounts, the owner creates a sending authorization policy and attaches the policy to the chosen identity by using the Amazon SES console or the Amazon SES API.
4. The identity owner gives the delegate sender the ARN of the authorized identity so that the delegate sender can provide the ARN to Amazon SES at the time of email sending.
5. The delegate sender can set up bounce and complaint notifications through event publishing enabled in a configuration set specified during delegate sending. The identity owner can also set up email feedback notifications for bounce and complaint events to be sent to the delegate sender's Amazon SNS topics.

> **Note**
>
> If the identity owner disables sending event notifications, the delegate sender must set up event publishing to publish bounce and complaint events to an Amazon SNS topic or a Firehose stream. The sender must also apply the configuration set that contains the event publishing rule to each email they send. If neither the identity owner nor the delegate sender sets up a method of sending notifications for bounce and complaint events, then Amazon SES automatically sends event notifications by email to the address in the Return-Path field of the email (or the address in the Source field, if you didn't specify a Return-Path address), even if the identity owner disabled email feedback forwarding.

6. The delegate sender attempts to send an email through Amazon SES on behalf of the identity owner by passing the ARN of the identity owner's identity in the request or in the header of the email. The delegate sender can send the email by using either the Amazon SES SMTP interface or the Amazon SES API. Upon receiving the request, Amazon SES examines any policies that are attached to the identity, and accepts the email if the delegate sender is authorized to use the specified "From" address and "Return Path" address; otherwise, Amazon SES returns an error and does not accept the message.

> **Important**
>
> The AWS account of the delegate sender has to be removed from the sandbox before it can be used to send email to non-verified addresses.

7. If the identity owner needs to de-authorize the delegate sender, the identity owner edits the sending authorization policy or deletes the policy entirely. The identity owner can perform either action by using the Amazon SES console or the Amazon SES API.

For more information about how the identity owner or delegate sender can perform those tasks, see Identity owner tasks or Delegate sender tasks, respectively.

Attribution of email sending features

It's important to understand the role of the delegate sender and the identity owner with respect to Amazon SES email sending features such as daily sending quota, bounces and complaints, DKIM signing, feedback forwarding, and so on. The attribution is the following:

- **Sending quotas** – Email sent from the identity owner's identities count against the delegate sender's quotas.
- **Bounces and complaints** – Bounce and complaint events are recorded against the delegate sender's Amazon SES account, and can therefore impact the delegate sender's reputation.
- **DKIM signing** – If the identity owner has enabled Easy DKIM signing for an identity, all email sent from that identity will be DKIM-signed, including email sent by the delegate sender. Only the identity owner can control whether the emails are DKIM-signed.
- **Notifications** – Both the identity owner and the delegate sender can set up notifications for bounces and complaints. The email identity owner can also enable email feedback forwarding. For information about setting up notifications, see Monitoring your Amazon SES sending activity.
- **Verification** – Identity owners are responsible for following the procedure in Verified identities to verify that they own the email addresses and domains that they're authorizing delegate senders to use. Delegate senders don't need to verify any email addresses or domains specifically for sending authorization.

  > **Important**
  >
  > The AWS account of the delegate sender has to be removed from the sandbox before it can be used to send email to non-verified addresses.

- **AWS Regions** – The delegate sender must send the emails from the AWS Region in which the identity owner's identity is verified. The sending authorization policy that gives permission to the delegate sender must be attached to the identity in that Region.

- **Billing** – All messages that are sent from the delegate sender's account, including emails that the delegate sender sends using the identity owner's addresses, are billed to the delegate sender.

Identity owner tasks for Amazon SES sending authorization

This section describes the steps that identity owners must take when configuring sending authorization.

Topics

- Verifying an identity for Amazon SES sending authorization
- Setting up identity owner notifications for Amazon SES sending authorization
- Getting information from the delegate sender for Amazon SES sending authorization
- Creating a sending authorization policy in Amazon SES
- Sending policy examples
- Providing the delegate sender with the identity information for Amazon SES sending authorization

Verifying an identity for Amazon SES sending authorization

The first step in configuring sending authorization is to prove that you own the email address or domain that the delegate sender will use to send email. The verification procedure is described in Verified identities.

You can confirm that an email address or domain is verified by checking its status in the Verified Identities section of the https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/ or by using the `GetIdentityVerificationAttributes` API operation.

Before you or the delegate sender can send email to non-verified email addresses, you have to submit a request to have your account removed from the Amazon SES sandbox. For more information, see Request production access (Moving out of the Amazon SES sandbox).

Important

- The AWS account of the delegate sender must be removed from the sandbox before it can be used to send email to or from non-verified addresses.

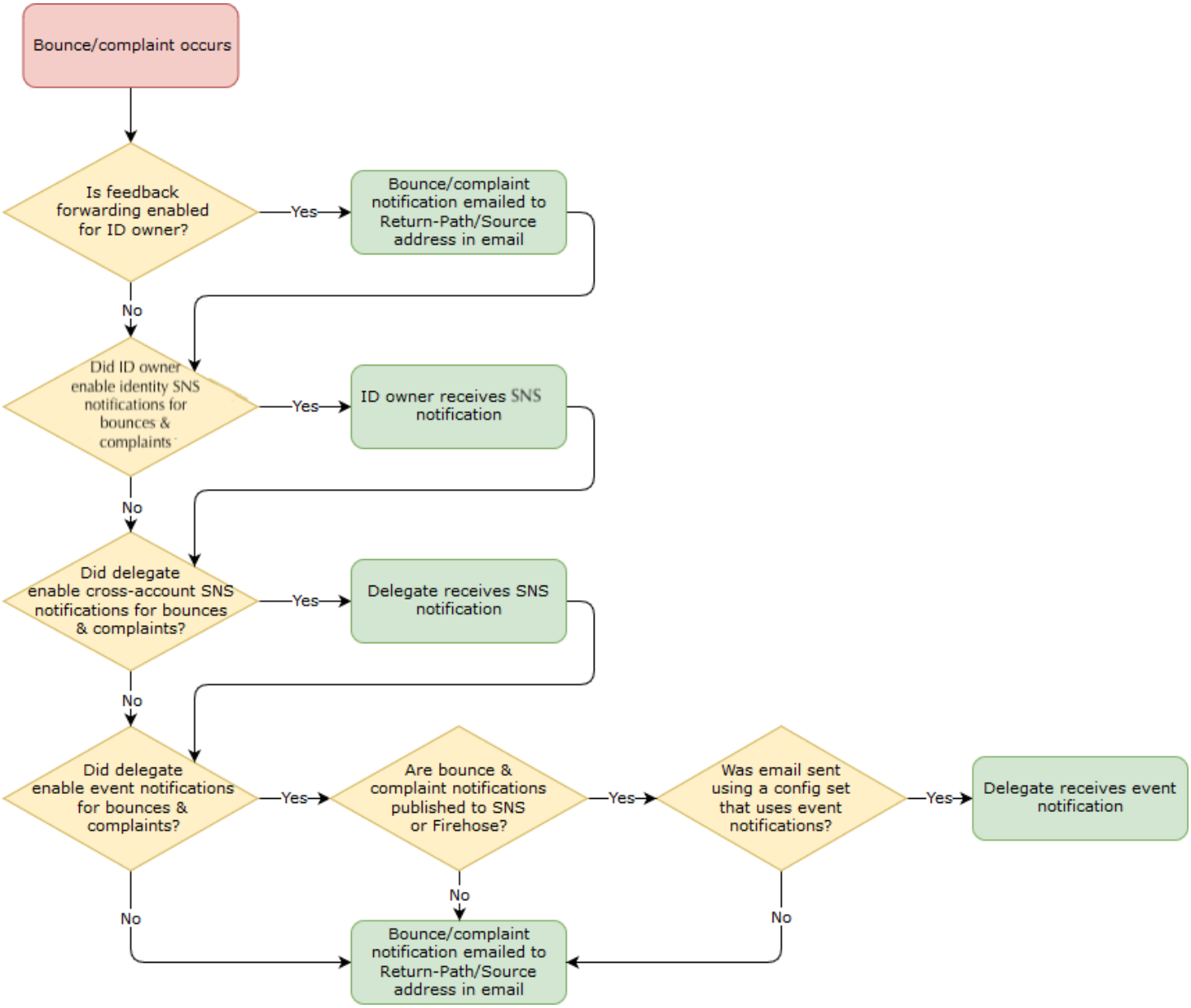
- If your account is in sandbox, you cannot send to email addresses that are not verified in your account, even if those domains or email addresses have been verified in the identity account

Setting up identity owner notifications for Amazon SES sending authorization

If you authorize a delegate sender to send email on your behalf, Amazon SES counts all bounces or complaints that those emails generate toward the delegate sender's bounce and complaint limits, rather than your own. However, if your IP address appears on third-party anti-spam, DNS-based Blackhole Lists (DNSBLs) as a result of messages sent by a delegate sender, the reputation of your identities may be damaged. For this reason, if you're an identity owner, you should set up email feedback forwarding for all your identities, including those that you've authorized for delegate sending. For more information, see Receiving Amazon SES notifications through email.

Delegate senders can and should set up their own bounce and complaint notifications for the identities that you have authorized them to use. They can set up event publishing to to publish bounce and complaint events to an Amazon SNS topic or a Firehose stream.

If neither the identity owner nor the delegate sender sets up a method of sending notifications for bounce and complaint events, or if the sender doesn't apply the configuration set that uses the event publishing rule, then Amazon SES automatically sends event notifications by email to the address in the Return-Path field of the email (or the address in the Source field, if you didn't specify a Return-Path address), even if you disabled email feedback forwarding. This process is illustrated in the following image.



Getting information from the delegate sender for Amazon SES sending authorization

Your sending authorization policy must specify at least one *principal*, which is the entity of your delegate sender that you're granting access to so they can send on behalf of one of your verified identities. For Amazon SES sending authorization policies, the principal can be either your delegate sender's AWS account or AWS Identity and Access Management (IAM) user ARN, or an AWS service.

An easy way to think about this is that the *principal* (delegate sender) is the grantee, and you (identity owner) are the grantor in the authorization policy where you are granting them the *Allow* permission to send any combination of email, raw email, templated email, or bulk templated email from the *resource* (verified identity) that you own.

If you want the finest grain control, ask the delegate sender to set up an IAM user so that only one delegate sender can send for you rather than any user in the delegate sender's AWS account. The delegate sender can find information about setting up an IAM user in Creating an IAM user in Your AWS Account in the *IAM User Guide*.

Ask your delegate sender for the AWS account ID or the IAM user's Amazon Resource Name (ARN) so that you can include it in your sending authorization policy. You can refer your delegate sender to the instructions for finding this information in Providing information to the identity owner. If the delegate sender is an AWS service, see the documentation for that service to determine the service name.

The following example policy illustrates the basic elements of what is needed in a policy created by the identity owner to authorize the delegate sender to send from the identity owner's resource. The identity owner would go into the Verified identities workflow, and under Authorization, use the Policy generator to create, in its simplest form, the following basic policy allowing the delegate sender to send on behalf of a resource owned by the identity owner:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSESSendEmail",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc:iam::111122223333:root"
      },
      "Action": [
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource": [
          "arn:aws-eusc:ses:eusc-de-east-1:444455556666:identity/bob@example.com"
      ],
      "Condition": {}
    }
  ]
}
```

For the policy above, the following legend explains the key elements and who owns them:

- **Principal** – this field is populated with the delegate sender's IAM user ARN.
- **Action** – this field is populated with two SES actions (`SendEmail` & `SendRawEmail`) that the identity owner is allowing the delegate sender to perform from the identity owner's resource.
- **Resource** – this field is populated with the identity owner's verified resource that they are authorizing the delegate sender to send from.

Creating a sending authorization policy in Amazon SES

Similar to creating any authorization policy in Amazon SES, as explained in Creating an identity authorization policy, to authorize a delegate sender to send emails using an email address or domain (an *identity*) that you own, you create the policy with SES sending API actions specified, and then attach that policy to the identity.

For a list of API actions that can be specified in a sending authorization policy, see the *Action* row in the the section called “Statements specific to the policy” table.

You can create a sending authorization policy by either using the policy generator or by creating a custom policy. Procedures specific to creating a sending authorization policy are provided for either method.

> **Note**
>
> - Sending authorization policies that you attach to email address identities take precedence over policies that you attach to their corresponding domain identities. For example, if you create a policy for *example.com* that disallows a delegate sender, and you create a policy for *sender@example.com* that allows the delegate sender, then the delegate sender can send email from *sender@example.com*, but not from any other address on the *example.com* domain.
> - If you create a policy for *example.com* that allows a delegate sender, and you create a policy for *sender@example.com* that disallows the delegate sender, then the delegate sender can send email from any address on the *example.com* domain, except for *sender@example.com*.
> - If you're unfamiliar with the structure of SES authorization policies, see Policy anatomy.
> - If the identity you're authorizing is duplicated in a secondary region as part of the Global endpoints feature, you'll need to create sending authorization policies on the identity in

both the primary and secondary regions so that the delegate sender has permission to use this identity for sending in both regions.

Creating a sending authorization policy by using the policy generator

You can use the policy generator to create a sending authorization policy by following these steps.

To create a sending authorization policy by using the policy generator

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the **Identities** container on the **Verified identities** screen, select the verified identity you wish to authorize for the delegate sender to send on your behalf.
4. Choose the verified identity's **Authorization** tab.
5. In the **Authorization policies** pane, choose **Create policy** and select **Use policy generator** from the dropdown.
6. In the **Create statement** pane, choose **Allow** in the **Effect** field. (If you want to create a policy to restrict your delegate sender, choose **Deny** instead.)
7. In the **Principals** field, enter the *AWS account ID* or *IAM user ARN* that your delegate sender shared with you to authorize them to send email on behalf of your account for this identity, then choose **Add**. (If you wish to authorize more than one delegate sender, repeat this step for each one.)
8. In the **Actions** field, select the check box for each send type you would like to authorize for your delegate sender.
9. (Optional) Expand **Specify conditions** if you wish to add a qualifying statement to the delegate sender permission.
    a. Select an operator from the **Operator** dropdown.
    b. Select a type from the **Key** dropdown.
    c. Respective to the key type you selected, enter its value in the **Value** field. (If you wish to add more conditions, choose **Add new condition** and repeat this step for each additional one.)
10. Choose **Save statement**.

11. (Optional) Expand **Create another statement** if you wish to add more statements to your policy and repeat steps 6 - 10.
12. Choose **Next** and on the **Customize policy** screen, the **Edit policy details** container has fields where you can change or customize the policy's **Name** and the **Policy document** itself.
13. Choose **Next** and on the **Review and apply** screen, the **Overview** container will show the verified identity you're authorizing for your delegate sender as well as the name of this policy. In the **Policy document** pane will be the actual policy you just wrote along with any conditions you added - review the policy and if it looks correct, choose **Apply policy**. (If you need to change or correct something, choose **Previous** and work in the **Edit policy details** container.) The policy you just created will allow your delegate sender to send on your behalf.
14. (Optional) If your delegate sender also wants to use an SNS topic that they own, to receive feedback notifications when they receive bounces or complaints, or when emails are delivered, you'll need to configure their SNS topic in this verified identity. (Your delegate sender will need to share with you their SNS topic ARN.) Select the **Notifications** tab and select **Edit** in the **Feedback notifications** container:

    a. On the **Configure SNS topics** pane, in any of the feedback fields, (Bounce, Complaint, or Delivery), select **SNS topic you don't own** and enter the **SNS topic ARN** owned and shared with you by your delegate sender. (Only your delegate sender will get these notifications because they own the SNS topic - you, as the identity owner, will not.)

    b. (Optional) If you want your topic notification to include the headers from the original email, check the **Include original email headers** box directly underneath the SNS topic name of each feedback type. This option is only available if you've assigned an Amazon SNS topic to the associated notification type. For information about the contents of the original email headers, see the `mail` object in Notification contents.

    c. Choose **Save changes**. The changes you made to your notification settings might take a few minutes to take effect.

    d. (Optional) Since your delegate sender will be getting Amazon SNS topic notifications for bounces and complaints, you can disable email notifications entirely if you don't want to receive feedback for this identity's sends. To disable email feedback for bounces and complaints, under the **Notifications** tab, in the **Email Feedback Forwarding** container, choose **Edit**, uncheck the **Enabled** box, and choose **Save changes**. Delivery status notifications will now only be sent to the SNS topics owned by your delegate sender.

Creating a custom sending authorization policy

If you want to create a custom sending authorization policy and attach it to an identity, you have the following options:

- **Using the Amazon SES API** – Create a policy in a text editor and then attach the policy to the identity by using the `PutIdentityPolicy` API described in the Amazon Simple Email Service API Reference.
- **Using the Amazon SES console** – Create a policy in a text editor and attach it to an identity by pasting it into the custom policy editor in the Amazon SES console. The following procedure describes this method.

To create a custom sending authorization policy by using the custom policy editor

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Identities**.
3. In the **Identities** container on the **Verified identities** screen, select the verified identity you wish to authorize for the delegate sender to send on your behalf.
4. In the details screen of the verified identity you selected in the previous step, choose the **Authorization** tab.
5. In the **Authorization policies** pane, choose **Create policy** and select **Create custom policy** from the dropdown.
6. In the **Policy document** pane, type or paste the text of your policy in JSON format. You can also use the policy generator to quickly create the basic structure of a policy and then customize it here.
7. Choose **Apply Policy**. (If you ever need to modify your custom policy, just select its check box under the **Authorization** tab, choose **Edit**, and make your changes in the **Policy document** pane followed by **Save changes**).
8. (Optional) If your delegate sender also wants to use an SNS topic that they own, to receive feedback notifications when they receive bounces or complaints, or when emails are delivered, you'll need to configure their SNS topic in this verified identity. (Your delegate sender will need to share with you their SNS topic ARN.) Select the **Notifications** tab and select **Edit** in the **Feedback notifications** container:

a. On the **Configure SNS topics** pane, in any of the feedback fields, (Bounce, Complaint, or Delivery), select **SNS topic you don't own** and enter the **SNS topic ARN** owned and shared with you by your delegate sender. (Only your delegate sender will get these notifications because they own the SNS topic - you, as the identity owner, will not.)

b. (Optional) If you want your topic notification to include the headers from the original email, check the **Include original email headers** box directly underneath the SNS topic name of each feedback type. This option is only available if you've assigned an Amazon SNS topic to the associated notification type. For information about the contents of the original email headers, see the `mail` object in Notification contents.

c. Choose **Save changes**. The changes you made to your notification settings might take a few minutes to take effect.

d. (Optional) Since your delegate sender will be getting Amazon SNS topic notifications for bounces and complaints, you can disable email notifications entirely if you don't want to receive feedback for this identity's sends. To disable email feedback for bounces and complaints, under the **Notifications** tab, in the **Email Feedback Forwarding** container, choose **Edit**, uncheck the **Enabled** box, and choose **Save changes**. Delivery status notifications will now only be sent to the SNS topics owned by your delegate sender.

Sending policy examples

Sending authorization enables you to specify the fine-grained conditions under which you allow delegate senders to send on your behalf.

The following conditions and examples show you how to write policies to control different aspects of sending:

- Conditions specific to sending authorization
- Specifying the delegate sender
- Restricting the "From" address
- Restricting the time at which the delegate can send email
- Restricting the email sending action
- Restricting the display name of the email sender
- Using multiple statements

Conditions specific to sending authorization

A *condition* is any restriction about the permission in the statement. The part of the statement that specifies the conditions can be the most detailed of all the parts. A *key* is the specific characteristic that's the basis for access restriction, such as the date and time of the request.

You use both conditions and keys together to express the restriction. For example, if you want to restrict the delegate sender from making requests to Amazon SES on your behalf after July 30, 2019, you use the condition called `DateLessThan`. You use the key called `aws:CurrentTime` and set it to the value `2019-07-30T00:00:00Z`.

You can use any of the AWS-wide keys listed at Available Keys in the *IAM User Guide*, or you can use one of the following keys specific to SES that are useful in sending authorization policies:

Condition key	Description
`ses:Recipients`	Restricts the recipient addresses, which include the To:, "CC", and "BCC" addresses.
`ses:FromAddress`	Restricts the "From" address.
`ses:FromDisplayName`	Restricts the contents of the string that is used as the "From" display name (sometimes called "friendly from"). For example, the display name of "John Doe <johndoe@example.com>" is John Doe.
`ses:FeedbackAddress`	Restricts the "Return Path" address, which is the address where bounce and complaints can be sent to you by email feedback forwarding. For informati on about email feedback forwarding, see Receiving Amazon SES notifications through email.

You can use the `StringEquals` and `StringLike` conditions with Amazon SES keys. These conditions are for case-sensitive string matching. For `StringLike`, the values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. For example, the following condition specifies that the delegate sender can only send from a "From" address that starts with *invoicing* and ends with *@example.com*:

```
"Condition": {
    "StringLike": {
      "ses:FromAddress": "invoicing*@example.com"
    }
}
```

You can also use the `StringNotLike` condition to prevent delegate senders from sending email from certain email addresses. For example, you can disallow sending from *admin@example.com*, and also similar addresses such as *"admin"@example.com*, *admin+1@example.com*, or *sender@admin.example.com*, by including the following condition in your policy statement:

```
"Condition": {
    "StringNotLike": {
      "ses:FromAddress": "*admin*example.com"
    }
 }
```

For more information about how to specify conditions, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Specifying the delegate sender

The *principal*, which is the entity to which you are granting permission, can be an AWS account, an AWS Identity and Access Management (IAM) user, or an AWS service.

The following example shows a simple policy that allows AWS ID *123456789012* to send email from the verified identity *example.com* (which is owned by AWS account *888888888888*). The `Condition` statement in this policy only allows the delegate (that is, AWS ID *123456789012*) to send email from the address *marketing+.*@example.com*, where * is any string that the sender wants to add after *marketing+.*.

JSON

```
{
  "Id":"SampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeMarketer",
      "Effect":"Allow",
```

```
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition":{
        "StringLike":{
          "ses:FromAddress":"marketing+.*@example.com"
        }
      }
    }
  ]
}
```

The following example policy grants permission to two IAM users to send from identity *example.com*. IAM users are specified by their Amazon Resource Name (ARN).

JSON

```
{
  "Id":"ExampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeIAMUser",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "arn:aws-eusc:iam::111122223333:user/John",
          "arn:aws-eusc:iam::444455556666:user/Jane"
        ]
      },
      "Action":[
```

```
        "ses:SendEmail",
        "ses:SendRawEmail"
      ]
    }
  ]
}
```

The following example policy grants permission to Amazon Cognito to send from identity *example.com*.

JSON

```
{
  "Id":"ExampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeService",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "Service":[
          "cognito-idp.amazonaws.com"
        ]
      },
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "888888888888",
          "aws:SourceArn": "arn:aws-eusc:cognito-idp:eusc-de-
east-1:888888888888:userpool/your-user-pool-id-goes-here"
        }
      }
    }
  ]
}
```

The following example policy grants permission to all accounts within an AWS Organization to send from identity example.com. The AWS Organization is specified using the PrincipalOrgID global condition key.

JSON

```
{
  "Id":"ExampleAuthorizationPolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeOrg",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":"*",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition":{
        "StringEquals":{
          "aws:PrincipalOrgID":"o-xxxxxxxxxx"
        }
      }
    }
  ]
}
```

Restricting the "From" address

If you use a verified domain, you may want to create a policy that allows only the delegate sender to send from a specified email address. To restrict the "From" address, you set a condition on the key called *ses:FromAddress*. The following policy enables AWS account ID *123456789012* to send from the identity *example.com*, but only from the email address *sender@example.com*.

JSON

```
{
  "Id":"ExamplePolicy",
```

```
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeFromAddress",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition":{
        "StringEquals":{
          "ses:FromAddress":"sender@example.com"
        }
      }
    }
  ]
}
```

Restricting the time at which the delegate can send email

You can also configure your sender authorization policy so that a delegate sender can send email only at a certain time of day, or within a certain date range. For example, if you plan to send an email campaign during the month of September 2021, you can use the following policy to restrict the delegate's ability to send email to that month only.

JSON

```
{
  "Id":"ExamplePolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"ControlTimePeriod",
      "Effect":"Allow",
```

```
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition":{
        "DateGreaterThan":{
          "aws:CurrentTime":"2021-08-31T12:00Z"
        },
        "DateLessThan":{
          "aws:CurrentTime":"2021-10-01T12:00Z"
        }
      }
    }
  ]
}
```

Restricting the email sending action

There are two actions that senders can use to send an email with Amazon SES: `SendEmail` and `SendRawEmail`, depending on how much control the sender wants over the format of the email. Sending authorization policies enable you to restrict the delegate sender to one of those two actions. However, many identity owners leave the details of the email sending calls up to the delegate sender by enabling both actions in their policies.

> **Note**
>
> If you want to enable the delegate sender to access Amazon SES through the SMTP interface, you must choose `SendRawEmail` at a minimum.

If your use case is such that you want to restrict the action, you can do so by including only one of the actions in your sending authorization policy. The following example shows you how to restrict the action to `SendRawEmail`.

JSON

```
{
  "Id":"ExamplePolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"ControlAction",
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:SendRawEmail"
      ]
    }
  ]
}
```

Restricting the display name of the email sender

Some email clients display the "friendly" name of the email sender (if the email header provides it), rather than the actual "From" address. For example, the display name of "John Doe <johndoe@example.com>" is John Doe. For instance, you might send emails from *user@example.com*, but you prefer that recipients see that the email is from *Marketing* rather than from *user@example.com*. The following policy enables AWS account ID 123456789012 to send from identity *example.com*, but only if the display name of the "From" address includes *Marketing*.

JSON

```
{
  "Id":"ExamplePolicy",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AuthorizeFromAddress",
```

```
      "Effect":"Allow",
      "Resource":"arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/
example.com",
      "Principal":{
        "AWS":[
          "123456789012"
        ]
      },
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Condition":{
        "StringLike":{
          "ses:FromDisplayName":"Marketing"
        }
      }
    }
  ]
}
```

Using multiple statements

Your sending authorization policy can include multiple statements. The following example policy has two statements. The first statement authorizes two AWS accounts to send from *sender@example.com* as long as the "From" address and the feedback address both use the domain *example.com*. The second statement authorizes an IAM user to send email from *sender@example.com* as long as the recipient's email address is under the *example.com* domain.

Providing the delegate sender with the identity information for Amazon SES sending authorization

After you create your sending authorization policy and attach it to your identity, you can provide the delegate sender with the Amazon Resource Name (ARN) of the identity. The delegate sender will pass that ARN to Amazon SES in the email-sending operation or in the header of the email. To find your identity's ARN, follow these steps.

To find the ARN of an identity

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

2. In the navigation pane, under **Configuration**, choose **Verified identities**.
3. In the list of identities, choose the identity to which you attached the sending authorization policy.
4. In the **Summary** pane, the second column, **Amazon Resource Name (ARN)**, will contain the identity's ARN. It will look similar to *arn:aws:ses:us-east-1:123456789012:identity/user@example.com*. Copy the entire ARN and give it to your delegate sender.

> **Important**
>
> If the identity you're authorizing is duplicated in a secondary region as part of the Global endpoints feature, replace the region parameter, such as, `us-east-1`, with an asterisk `*` as in the following example, `arn:aws:ses:*:123456789012:identity/user@example.com`.

Delegate sender tasks for Amazon SES sending authorization

As a delegate sender, you're sending emails on behalf of an identity that you don't own, but are authorized to use. Even though you're sending on the identity owner's behalf, bounces and complaints count toward the bounce and complaint metrics for your AWS account, and the number of messages you send counts toward your sending quota. You're also responsible for requesting any sending quota increases that you might need in order to send the identity owner's emails.

As a delegate sender, you must complete the following tasks:

- Providing information to the identity owner
- Using delegate sender notifications
- Sending emails for the identity owner

Providing information to the identity owner for Amazon SES sending authorization

As a delegate sender, you must provide the identity owner with either your AWS account ID or your IAM user Amazon Resource Name (ARN) since you will be sending email on behalf of the identity owner. The identity owner needs your account information so they can create a policy that grants you permission to send from one of their verified identities.

If you want to use your own SNS topics, you can request that your identity owner configure feedback notifications for bounces, complaints, or deliveries to be sent to one or more of your SNS

topics. To do this, you'll need to share your SNS topic ARN with your identity owner so that they can configure your SNS topic in the verified identity they're authorizing you to send from.

The following procedures explain how to find your account information and SNS topic ARNs to share with your identity owner.

To find your AWS account ID

1. Sign in to the AWS Management Console at https://eusc-de-east-1.console.amazonaws-eusc.eu.
2. In the upper right-hand corner of the console, expand your name/account number, and choose **My Account** in the dropdown.
3. The Account settings page will open and display all of your account information including your AWS account ID.

To find your IAM user ARN

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, choose **Users**.
3. In the list of users, choose the user name. The **Summary** section displays the IAM user ARN. The ARN resembles the following example: *arn:aws:iam::123456789012:user/John*.

To find your SNS topic ARN

1. Open the Amazon SNS console at https://eusc-de-east-1.console.amazonaws-eusc.eu/sns/v3/home.
2. In the navigation pane, choose **Topics**.
3. In the list of topics, the SNS topic ARNs are displayed in the **ARN** column. The ARN resembles the following example: *arn:aws:sns:us-east-1:444455556666:my-sns-topic*.

Using delegate sender notifications for Amazon SES sending authorization

As a delegate sender, you're sending emails on behalf of an identity that you don't own, but are authorized to use; however, bounces and complaints still count toward *your* bounce and complaint metrics, not those of the identity owner.

If the bounce or complaint rates for your account gets too high, your account is at risk of being placed under review or have its ability to send email paused. For this reason, it's important that you set up notifications and have a process in place to monitor them. You also need to have a process in place for removing addresses that have bounced or complained from your mailing lists.

Therefore, as a delegate sender, you can configure Amazon SES to send notifications when bounce and complaint events occur for the emails you send on behalf of any identities that you don't own, but have been authorized to use by the identity owner. You can also set up event publishing to publish bounce and complaint notifications to Amazon SNS or Firehose.

> **Note**
>
> If you set up Amazon SES to send notifications by using Amazon SNS, you're charged standard Amazon SNS rates for the notifications you receive. For more information, see the Amazon SNS pricing page.

Create a new delegate sender notification

You can set up delegate sending notifications with either configuration sets using event publishing, or with verified identities configured with your own SNS topics.

Procedures are given below for setting up new delegate sending notifications using either method:

- Event publishing through a configuration set
- Feedback notifications to SNS topics you own

To set up event publishing through a configuration set for your delegate sending

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Follow the procedures in Create event destinations.
3. After you've set up event publishing in your configuration set, specify the name of the configuration set when you send email as a delegate sender using the verified identity the identity owner authorized you to send from. See Sending emails for the identity owner.

To set up feedback notifications to SNS topics you own for your delegate sending

1. After you've decided which of your SNS topics you'd like to use for feedback notifications, follow the procedures to find your SNS topic ARN and copy the full ARN and share it with your identity owner.
2. Ask your identity owner to configure your SNS topics for feedback notifications on the shared identity he's authorized you to send from. (Your identity owner will need to follow the procedures given for configuring SNS topics in the authorization policy procedures.)

Sending emails for the identity owner for Amazon SES sending authorization

As a delegate sender, you send emails the same way that other Amazon SES senders do, except that you provide the Amazon Resource Name (ARN) of the identity that the identity owner has authorized you to use. When you call Amazon SES to send the email, Amazon SES checks to see if the identity that you specified has a policy that authorizes you to send for it.

There are different ways that you can specify the identity's ARN when you send an email. The method that you use depends on whether you send the email by using the Amazon SES API operations or the Amazon SES SMTP interface.

Important

- To successfully send an email, you have to connect to the Amazon SES endpoint in the AWS Region that the identity owner verified the identity in.
- Additionally, the AWS accounts of **both** the identity owner and the delegate sender have to be removed from the sandbox before either account can send email to non-verified addresses. For more information, see Request production access (Moving out of the Amazon SES sandbox).
- If the identity you've been authorized to use is duplicated in a secondary region as part of the Global endpoints feature:
  - The identity owner should have supplied you with an identity ARN that had the region parameter, such as, `us-east-1`, replaced with an asterisk `*` as in the following example, `arn:aws:ses:*:123456789012:identity/user@example.com`.
  - The identity owner should have created sending authorization policies for you in both the primary and secondary regions.

Using the Amazon SES API

As with any Amazon SES email sender, if you access Amazon SES through the Amazon SES API (either directly through HTTPS or indirectly through an AWS SDK), you can choose between one of three email-sending actions: `SendEmail`, `SendTemplatedEmail`, and `SendRawEmail`. The Amazon Simple Email Service API Reference describes the details of these APIs, but we provide an overview of the sending authorization parameters here.

SendRawEmail

If you want to use `SendRawEmail` so that you can control the format of your emails, you can specify the delegated authorized identity in one of two ways:

- **Pass optional parameters to the `SendRawEmail` API**. The required parameters are described in the following table:

Parameter	Description
`SourceArn`	The ARN of the identity that is associated with the sending authorization policy that permits you to send for the email address specified in the `Source` parameter of `SendRawEmail` . **Note** If you only specify the `SourceArn` , Amazon SES sets the "From" address and the "Return Path" addresses to the identity that you specified in `SourceArn` .
`FromArn`	The ARN of the identity that is associated with the sending authorization policy that permits you to specify a particular "From" address in the header of the raw email.
`ReturnPathArn`	The ARN of the identity that is associated with the sending authorization policy that permits you to use

Parameter	Description
	the email address specified in the `ReturnPath` parameter of `SendRawEmail` .

- **Include X-headers in the email**. X-headers are custom headers that you can use in addition to standard email headers (such as the From, Reply-To, or Subject headers). Amazon SES recognizes three X-headers that you can use to specify sending authorization parameters:

> **Important**
>
> Do not include these X-headers in the DKIM signature, because they are removed by Amazon SES before sending the email.

X-Header	Description
`X-SES-SOURCE-ARN`	Corresponds to the `SourceArn` .
`X-SES-FROM-ARN`	Corresponds to the `FromArn`.
`X-SES-RETURN-PATH-ARN`	Corresponds to the `ReturnPathArn` .

Amazon SES removes all X-headers from the email before sending it. If you include multiple instances of an X-header, Amazon SES uses only the first instance.

The following example shows an email that includes sending authorization X-headers:

```
X-SES-SOURCE-ARN: arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/example.com
X-SES-FROM-ARN: arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/example.com
X-SES-RETURN-PATH-ARN: arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com

From: sender@example.com
To: recipient@example.com
Return-Path: feedback@example.com
Subject: subject
Content-Type: multipart/alternative;
 boundary="----=_boundary"
```

```
------=_boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

body
------=_boundary
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

body
------=_boundary--
```

SendEmail and SendTemplatedEmail

If you use the `SendEmail` or `SendTemplatedEmail` operation, you can specify the delegated authorized identity by passing in the optional parameters below. You can't use the X-header method when you use the `SendEmail` or `SendTemplatedEmail` operation.

Parameter	Description
`SourceArn`	The ARN of the identity that is associated with the sending authorization policy that permits you to send for the email address specified in the `Source` parameter of either `SendEmail` or `SendTemplatedEmail`.
`ReturnPathArn`	The ARN of the identity that is associated with the sending authorization policy that permits you to use the email address specified in the `ReturnPath` parameter of either `SendEmail` or `SendTemplatedEmail`.

The following example shows how to send an email that includes the `SourceArn` and `ReturnPathArn` attributes using either the `SendEmail` or `SendTemplatedEmail` operation and the AWS SDK for Python (Boto3).

```
import boto3
```

```
from botocore.exceptions import ClientError

# Create a new SES resource and specify a region.
client = boto3.client('ses',region_name="eusc-de-east-1")

# Try to send the email.
try:
    #Provide the contents of the email.
    response = client.send_email(
        Destination={
            'ToAddresses': [
                'recipient@example.com',
            ],
        },
        Message={
            'Body': {
                'Html': {
                    'Charset': 'UTF-8',
                    'Data': 'This email was sent with Amazon SES.',
                },
            },
            'Subject': {
                'Charset': 'UTF-8',
                'Data': 'Amazon SES Test',
            },
        },
        SourceArn='arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/example.com',
        ReturnPathArn='arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/
example.com',
        Source='sender@example.com',
        ReturnPath='feedback@example.com'
    )
# Display an error if something goes wrong.
except ClientError as e:
    print(e.response['Error']['Message'])
else:
    print("Email sent! Message ID:"),
    print(response['ResponseMetadata']['RequestId'])
```

Using the Amazon SES SMTP interface

When you use the Amazon SES SMTP interface for delegate sending, you have to include the `X-SES-SOURCE-ARN`, `X-SES-FROM-ARN`, and `X-SES-RETURN-PATH-ARN` headers in your message. Pass these headers after you issue the `DATA` command in the SMTP conversation.

Sending test emails in Amazon SES with the simulator

We recommend using the Amazon SES console to send a test email with Amazon SES. Because the console requires you to manually enter information, you typically only use it to send test emails. After you get started with Amazon SES, you will most likely send your emails by using either the Amazon SES SMTP interface or API. However,the console is useful for monitoring your sending activity.

The following topics explain how to use the mailbox simulator from both the console and manually by sending emails:

- Using the mailbox simulator from the console
- Using the mailbox simulator manually

Using the mailbox simulator from the console

Important

- In this tutorial, you send an email to yourself from the console so that you can check to see if you received it. For further experimentation or load testing, see Using the mailbox simulator manually.
- Emails that you send to the mailbox simulator do not count toward your sending quota or your bounce and complaint rates, nor do they affect Virtual Deliverability Manager metrics.

Before you follow these steps, complete the tasks in Setting up Amazon Simple Email Service.

To send a test email message from the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane under **Configuration** choose **Identities**.
3. From the **Identities** table, select a verified email identity (by clicking directly on the identity name as opposed to selecting its checkbox). *If you don't have a verified email identity, see Creating an email address identity.*
4. On the selected email identity's detail page, choose **Send test email**.
5. For **Message details**, choose the **Email Format**. The two choices are as follows:

   - **Formatted**—This is the simplest option. Choose this option if you simply want to type the text of your message into the **Body** text box. When you send the email, Amazon SES puts the text into email format for you.
   - **Raw**—Choose this option if you want to send a more complex message, such as a message that includes HTML or an attachment. Because of this flexibility, you need to format the message, as described in Sending raw email using the Amazon SES API v2, yourself, and then paste the entire formatted message, including the headers, into the **Body** text box. You can use the following example, which contains HTML, to send a test email using the **Raw** email format. Copy and paste this message in its entirety into the **Body** text box. Ensure that there is not a blank line between the `MIME-Version` header and the `Content-Type` header; a blank line between these two lines causes the email to be formatted as plain text instead of HTML.

```
Subject: Amazon SES Raw Email Test
MIME-Version: 1.0
Content-Type: text/html

<!DOCTYPE html>
<html>
<body>
<h1>This text should be large, because it is formatted as a header in HTML.</h1>
<p>Here is a formatted link: <a href="https://docs.aws.eu/ses/latest/
DeveloperGuide/Welcome.html">Amazon Simple Email Service Developer Guide</a>.</p>
</body>
</html>
```

6. Choose the type of simulated email scenario you want to test by expanding the **Scenario** list box.

    - If you choose **Custom** and you're still in the Amazon SES sandbox, make sure that the address in the **Custom recipient** field is a verified email address. For more information, see Creating an email address identity.

7. Fill out the remaining fields as desired.
8. Choose **Send test email**.
9. Sign in to the email client of the address you sent the email to. You will find the message that you sent.

Using the mailbox simulator manually

Amazon SES includes a mailbox simulator that you can use to test how your application handles different email sending scenarios. The mailbox simulator is useful when, for example, you want to test an email sending application without creating fictitious email addresses, or when you want to find your system's maximum throughput without impacting your daily sending quota.

Important considerations

Consider the following features and limitations when you use the Amazon SES mailbox simulator:

- You can use the mailbox simulator even if your account is in the Amazon SES sandbox.
- Emails that you send to the mailbox simulator are limited by your account's maximum sending rate, but they don't affect your daily sending quota. For example, if your account is authorized to send 10,000 messages per 24-hour period, and you send 100 messages to the mailbox simulator, you can still send up to 10,000 messages to regular recipients without reaching your sending quota.
- Emails that you send to the mailbox simulator don't impact your email deliverability or reputation metrics. For example, if you send a large number of messages to the bounce address of the email simulator, it doesn't display a message warning you that your bounce rate is too high on the reputation metrics console page.
- For billing purposes, emails that you send to the Amazon SES mailbox simulator are the same as any other email you send using Amazon SES. In other words, we bill you the same amount for messages that you send to the mailbox simulator as for those that you send to regular recipients.

- The mailbox simulator supports labeling, which enables you to send emails to the same mailbox simulator address in multiple ways, or to see how your application handles Variable Envelope Return Path (VERP). For example, you can send an email to *bounce+label1@simulator.amazonses.com* and *bounce+label2@simulator.amazonses.com* to see if your application can match a bounce message with the email address that caused the bounce.
- If you use the mailbox simulator to simulate multiple bounces from the same sending request, Amazon SES combines the bounce responses into a single response.

Using the mailbox simulator

To use the email simulator, find the scenario in the following table, and then send an email to the corresponding email address.

> **Note**
>
> When you send an email to a mailbox simulator address, you must send it through Amazon SES, by using the AWS CLI, an AWS SDK, the Amazon SES console, the Amazon SES SMTP interface, or the Amazon SES API. The mailbox simulator doesn't respond to emails that it receives from external sources.

Simulated scenario	Email address
Successful delivery—The recipient's email provider accepts your email. If you set up delivery notifications as described in Setting up event notifications for Amazon SES, Amazon SES sends you a delivery notification through Amazon Simple Notification Service (Amazon SNS).	success@simulator.amazonses.com
Bounce—The recipient's email provider rejects your email with an SMTP 550 5.1.1 ("Unknown User") response code. Amazon SES generates a bounce notification and, depending on how you set up your account, sends it to you in an email or sends a notification to an	bounce@simulator.amazonses.com

Simulated scenario	Email address
Amazon SNS topic. The mailbox simulator email address isn't placed on the Amazon SES suppression list, which would normally happen when a hard bounce occurs. The bounce response that you receive from the mailbox simulator is compliant with RFC 3464. For information about how to receive bounce feedback, see Setting up event notifications for Amazon SES.	
Automatic responses—The recipient's email provider accepts your email and delivers it to the recipient's inbox. The email provider sends an automatic response, such as an "out of the office" (OOTO) message, to the address in the Return-Path header of the email, or the envelope sender ("MAIL FROM") address if the Return-Path header isn't present. The automatic response that you receive from the mailbox simulator is compliant with RFC 3834.	ooto@simulator.amazonses.com
Complaint—The recipient's email provider accepts your email and delivers it to the recipient's inbox. The recipient decides that your message is unsolicited and clicks "Mark as Spam" in his or her email client. Amazon SES then forwards the complaint notification to you by email or by notifying an Amazon SNS topic, depending on how you set up your account. The complaint response that you receive from the mailbox simulator is compliant with RFC 5965. For information about how to receive complaint feedback, see Setting up event notifications for Amazon SES.	complaint@simulator.amazonses.com

Simulated scenario	Email address
Recipient address on suppression list—Amazon SES generates a hard bounce as if the recipient's address is on the global suppression list.	suppressionlist@simulator.amazonses.com

Testing Reject events

Every message that you send through Amazon SES is scanned for viruses. If you send a message that contains a virus, Amazon SES accepts the message, detects the virus, and rejects the entire message. When Amazon SES rejects the message, it stops processing the message, and doesn't attempt to deliver it to the recipient's mail server. It then generates a Reject event.

The Amazon SES mailbox simulator doesn't include an address for testing Reject events. However, you can test Reject events by using a European Institute for Computer Antivirus Research (EICAR) test file. This file is an industry-standard method of testing antivirus software in a safe manner. To create an EICAR test file, paste the following text into a file:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Save the file as `sample.txt`, attach it to an email, and then send the email to a verified address. If there are no other issues with the email, Amazon SES accepts the message, but then rejects it as it would if it contained an actual virus.

> **Note**
>
> Rejected emails—including those that you send by using the procedure above—count against your daily sending quota. We bill you for each message that you send, including rejected messages.

To learn more about EICAR test files, see the EICAR test file page on Wikipedia.

Using configuration sets in Amazon SES

Configuration sets are groups of rules that you can apply to your verified identities. A verified identity is a domain, subdomain, or email address you use to send email through Amazon SES. When you apply a configuration set to an email, all of the rules in that configuration set are applied to the email.

You can use configuration sets to apply the following types of rules to your email sending and can contain one, both, or neither of these types:

- *Event destinations* – Allow you to publish email sending metrics, including the numbers of sends, deliveries, opens, clicks, bounces, and complaints to other AWS products for each email you send. For example, you can send your email metrics to an Amazon Data Firehose destination, and then analyze it using Amazon Managed Service for Apache Flink. Alternatively, you can send bounce and complaint information to Amazon SNS and receive notifications immediately when those events occur.
- *IP pool management* – If you lease dedicated IP addresses to use with Amazon SES, you can create groups of these addresses called *dedicated IP pools* to be used for sending specific types of email. For example, you can associate these dedicated IP pools with configuration sets and use one for sending marketing communications, and another for sending transactional emails. Your sender reputation for transactional emails is then isolated from that of your marketing emails.

To associate a configuration set with a verified identity can be done in the following ways:

- Include a reference to the configuration set in the headers of the email. For more information about specifying configuration sets in your emails, see Specifying a configuration set when you send email.
- Specify an existing configuration set to be used as the identity's *default configuration set*, either at the time of identity creation, or later while editing a verified identity. See Understanding default configuration sets.

Contents

- Creating configuration sets in SES
- Managing configuration sets in Amazon SES
- Specifying a configuration set when you send email

- Viewing and exporting reputation metrics

Creating configuration sets in SES

You can use the SES console, the `CreateConfigurationSet` action in the Amazon SES API v2, or the `aws sesv2 create-configuration-set` command in the Amazon SES CLI v2 to create a new configuration set. This section shows how to create configuration sets using the SES console and the Amazon SES CLI v2.

Create a configuration set (console)

To create a configuration set using the SES console, follow these steps:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. Choose **Create set**.
4. **General details** – This section provides options to customize your configuration set:

   - **Configuration set name** – The name for your configuration set. The name can contain up to 64 alphanumeric characters, including letters, numbers, hyphens (-) and underscores (_) only.
   - **Sending IP pool** – When you send email using this configuration set, messages are sent from the dedicated IP addresses in the assigned pool. Select an IP pool from the list.

     > **Note**
     >
     > The **default** (ses-default-dedicated-pool) contains dedicated IP addresses that haven't been assigned to any other pool. To learn more about managing IP pools, see Assign IP pools.

   - **Tracking options**
     - **Use a custom redirect domain** – Select the check box to use a custom redirect domain to handle open and click tracking for email sent with this configuration set.
     - **Custom redirect domain** – Select a verified domain from the *Choose a verified domain* list to be your custom redirect domain. You can also enter a subdomain in the *Enter a subdomain* field.

> **Note**
>
> Custom redirect domains can be specified as follows:
>
> - You must first create and verify a custom redirect domain in the AWS Region you want to send and track email, as well as set up a Content Delivery Network (CDN). This is explained in Configuring custom domains to handle open and click tracking.
> - Then, to use your custom redirect domain for open and click tracking, you must indicate it while creating or editing your configuration set here in this step.
> - Finally, after specifying your custom redirect domain, **View DNS records** will appear in the configuration set's **General details** container. If you expand it, you'll see the CNAME record that contains the tracking domain being used in your AWS Region. For example, if your custom subdomain is called *marketing.example.com* and it was created in the AWS Region `us-east-1`, expanding **View DNS records** would reveal a CNAME record with the following values: **Name** = *marketing.example.com* and **Value** = *r.us-east-1.awstrack.me*.
>
> You can use this information simply as a confirmation that you chose the correct tracking domain from the table when you set up your CDN as explained in Configuring custom domains to handle open and click tracking, or you can do this first, and use the CNAME record values from here to use in your CDN setup.

- **HTTPS policy** – Select an HTTPS policy option for the protocol of the open and click tracking links for your custom redirect domain:
  - **Optional** – (Default behavior) Open tracking links will be wrapped using HTTP. Click tracking links will be wrapped using the original protocol of the link.
  - **Required** – Open and Click tracking links will both be wrapped using HTTPS.
  - **Required for opens** – Open tracking links will be wrapped using HTTPS. Click tracking links will be wrapped using the original protocol of the link.
- **Advanced delivery options** – Choose the arrow on the left to expand the advanced delivery options section.
  - **Transport Layer Security (TLS)** – To require SES to establish a secure connection with the receiving mail server, and send emails using the TLS protocol, select the **Required** check box.

> **Note**
>
> SES supports TLS 1.2 and recommends TLS 1.3. To learn more, see Infrastructure security in SES.

- **Maximum delivery duration** – To specify a time limit for SES to attempt email delivery through this configuration set, enter a value in seconds ranging from 300 and up to 50,400.

> **Note**
>
> Setting a custom maximum delivery limit (shorter than the SES default of 14 hours), can be useful in such cases as time-sensitive emails (like those containing a one-time-password), transactional emails, and email that you want to ensure isn't delivered during non-business hours.
>
> > **Tip**
> >
> > - To calculate minutes to seconds, multiply by 60, e.g., 7 minutes * 60 = 420 seconds.
> > - To calculate hours to seconds, multiply by 3600, e.g., 2 hours * 3600 = 7200 seconds.

5. **Reputation options** – This section provides for setting up reputation metrics:

   - **Reputation metrics** – Used to track bounce and complaint metrics in CloudWatch for emails sent using this configuration set. *(Additional charges apply, see Price per metric for CloudWatch.)*
     - **Enabled** – Select this check box to enable reputation metrics for the configuration set.

6. **Suppression list options** – This section provides a decision set to define customized suppression starting with the option to use this configuration set to override your account-level suppression. The configuration set-level suppression logic map will help you understand the effects of the override combinations. These multitiered selections of overrides can be combined to implement three different levels of suppression:

a. **Use account-level suppression:** Do not override your account-level suppression and do not implement any configuration set-level suppression - basically, any email sent using this configuration set will just use your account-level suppression. To do this:

   - In **Suppression list settings**, uncheck the **Override account level settings** box.

b. **Do not use any suppression:** Override your account-level suppression without enabling any configuration set-level suppression - this means any email sent using this configuration set will not use any of your account-level suppression; in other words, all suppression is cancelled. To do this:

   i. In **Suppression list settings**, check the **Override account level settings** box.

   ii. In **Suppression list**, uncheck the **Enabled** box.

c. **Use configuration set-level suppression:** Override your account-level suppression with custom suppression list settings defined in this configuration set - this means any email sent using this configuration set will only use its own suppression settings and ignore any account-level suppression settings. To do this:

   i. In **Suppression list settings**, check the **Override account level settings** box.

   ii. In **Suppression list**, check **Enabled**.

   iii. In **Specify the reason(s)...**, select one of the suppression reasons for this configuration set to use.

7. **Virtual Deliverability Manager options** – This section will only be present if you have Virtual Deliverability Manager features enabled. Here, you can define custom settings for how this configuration set will use engagement tracking and optimized shared delivery by overriding how they've been defined in your Virtual Deliverability Manager settings at the account level:

   a. To disable both engagement tracking and optimized shared delivery for this configuration set:

      i. Check the **Override account level settings** box.

      ii. Ensure **Enabled** is unchecked for both *Engagement tracking* and *Optimized shared delivery*, then choose **Save changes**.

   b. To enable or disable either, or both, engagement tracking and optimized shared delivery for this configuration set:

      i. Check the **Override account level settings** box.

ii. Check or uncheck **Enabled** for either or both *Engagement tracking* and *Optimized shared delivery*, then choose **Save changes**.

c. To revert back to your Virtual Deliverability Manager account level settings for engagement tracking and optimized shared delivery for this configuration set:

- Uncheck the **Override account level settings** box, then choose **Save changes**.

8. **Archiving options** – This section provides the option to archive email sent from this configuration set:

a. Select the **Enabled** check box.

b. Click inside the **Archive** field and select an archive from the list followed by **Save changes**, or choose **Create archive** and continue with the remaining steps.

c. Enter a unique name in the **Archive name** field.

d. (Optional) Select a retention period in the **Retention period** field to override the default retention period of 180 days.

e. (Optional) You can encrypt your archive either by entering your own AWS KMS key into the **KMS key ARN** field, or by selecting **Create an AWS KMS key**.

f. Choose **Create archive**.

9. **Tags** – In this section, you can optionally add one or more tags to your configuration set:

a. Choose **Add new tag**.

b. Enter the tag **Key**.

c. Enter the tag **Value** (optional).

To remove a tag you've entered, choose **Remove** for that tag. You can enter a maximum of 50 tags.

10. Choose **Create set** to create your configuration set.

Now that you've created your configuration set, you have the option to define event destinations for your configuration set which enables event publishing that is triggered on the event types you specify for the event destination. A configuration set can have multiple event destinations with multiple event types defined. See Creating Amazon SES event destinations.

Create a configuration set (AWS CLI)

You can create a configuration set using a JSON file as input to the `aws sesv2 create-configuration-set` command in the AWS CLI.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the following keys, plus values that are valid for your environment, or use the SES API v2 `aws sesv2 create-configuration-set` command with the `--generate-cli-skeleton` option with no value specified to print a sample JSON structure to standard output.

   This example uses a file named `create-configuration-set.json`:

```
{
    "ConfigurationSetName": "sample-configuration-set",
    "TrackingOptions": {
        "CustomRedirectDomain": "some.domain.com",
        "HttpsPolicy": "REQUIRE"
    },
    "DeliveryOptions": {
        "TlsPolicy": "REQUIRE",
        "SendingPoolName": "sending pool",
        "MaxDeliverySeconds": 300
    },
    "ReputationOptions": {
        "ReputationMetricsEnabled": true,
        "LastFreshStart": timestamp
    },
    "SendingOptions": {
        "SendingEnabled": true
    },
    "Tags": [
        {
            "Key": "tag key",
            "Value": "tag value"
        }
    ],
    "SuppressionOptions": {
        "SuppressedReasons": ["BOUNCE","COMPLAINT"]
    },
    "ArchivingOptions": {
```

```
        "ArchiveArn": "arn:aws:ses:us-east-1:123456789012:mailmanager-
archive/MyArchiveID"
    }
}
```

> **Note**
>
> - You must include the `file://` notation at the beginning of the JSON file path.
> - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws sesv2 create-configuration-set --cli-input-json file://create-configuration-
set.json
```

> **Note**
>
> To review the AWS CLI reference for this command, see create-configuration-set.

Managing configuration sets in Amazon SES

After creating a configuration set, you can manage it with the view, edit, and delete options using the SES console, the Amazon SES API v2, and the Amazon SES CLI v2. Configuration sets can also be assigned to a verified identity as its default configuration set that is applied every time email is sent from the identity.

Topics in this section:

- View, edit, & delete configuration set (console)
- List configuration sets (AWS CLI)
- Get configuration set details (AWS CLI)
- Delete a configuration set (AWS CLI)
- Stop sending email from a configuration set (AWS CLI)

- Understanding default configuration sets
- Creating Amazon SES event destinations
- Assigning IP pools in Amazon SES
- Configuring custom domains to handle open and click tracking

View, edit, & delete configuration set (console)

Access an existing configuration set's detail page

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. Select a **Name** from the configuration set list to open its details page in the **Overview** tab where you can view, edit, or disable the options you've selected. The same can be done for **Events destination** options by selecting its tab. For more information about each option and their fields, see the related section in Create a configuration set (console).
4. At the top of each configuration set's details page, visible from either the **Overview** or **Events destination** tab, are the following options:

   - **Delete** – this button will delete your configuration set.
   - **Disable sending** – this button will stop sending emails from your configuration set.

List configuration sets (AWS CLI)

You can use the **list-configuration-sets** command in the AWS CLI to generate a list of all the configuration sets associated with your account in the current Region, as follows:

```
aws sesv2 list-configuration-sets
```

Get configuration set details (AWS CLI)

You can use the **get-configuration-set** command in the AWS CLI to get details for a specific configuration set, as follows:

```
aws sesv2 get-configuration-set --configuration-set-name name
```

Delete a configuration set (AWS CLI)

You can use the **delete-configuration-set** command in the AWS CLI to delete a specific configuration set, as follows:

```
aws sesv2 delete-configuration-set --configuration-set-name name
```

Stop sending email from a configuration set (AWS CLI)

You can use the **put-configuration-set-sending-options** command in the AWS CLI to stop sending email from a specific configuration set, as follows:

```
aws sesv2 put-configuration-set-sending-options --configuration-set-name name --no-sending-enabled
```

To start sending again, run the same command with the `--sending-enabled` option instead, as follows:

```
aws sesv2 put-configuration-set-sending-options --configuration-set-name name --sending-enabled
```

Understanding default configuration sets

The concept of assigning a configuration set as the default to be used by a verified identity is explained in this section to help understand the benefits and use case.

A default configuration set automatically applies its rules to all messages that you send from the email identity associated with that configuration set. You can apply default configuration sets to both email address and domain identities during the creation of the identity or after the fact as an edit function of an existing identity.

Default configuration set considerations

- The configuration set must be created first before associating it with an identity.
- Default configuration sets will only be applied if the identity is verified.
- An email identity can be associated with only one configuration set at a time. However, you can apply the same configuration set to multiple identities.

- A default configuration set at the email address level overrides a default configuration set at the domain level. For example, a default configuration set associated with *joe@example.com* overrides the configuration set for the domain of *example.com*.
- A default configuration set at the domain level applies to all email addresses for that domain (unless you verify specific addresses for the domain).
- If you delete a configuration set that's designated as the default configuration set for an identity, and then attempt to send email through that identity, your call to Amazon SES fails with a "bad request" error.
- A default configuration set cannot be assigned to a verified identity that's being used by a delegate sender.
- How to specify an existing configuration set to be used as the identity's default configuration set is actually a function of verified identities, so instructions are given in the identity workflows accordingly:
  - **Specify a default configuration set during identity creation** – follow the instructions given in the optional Step 6 for either Domain identity default configuration set or Email identity default configuration set located in the Creating and verifying identities in Amazon SES chapter.
  - **Specify a default configuration set for an existing identity** – follow the steps in Edit an identity using the console along with these details for Step 5:
    a. Choose the **Configuration set** tab.
    b. Choose **Edit** in the **Default configuration set** container.
    c. Select the list box and choose an existing configuration set to be used as the default.
    d. Continue with the remaining steps in Edit an identity using the console.

> **Note**
>
> If the configuration set you assign as a default has reputation metrics enabled, additional charges will be incurred for any mail sent using the default configuration set, see Price per metric for CloudWatch.

Creating Amazon SES event destinations

Event destinations allow you to publish the following outgoing email tracking actions to other AWS services for monitoring:

- Sends
- Rendering failures
- Rejects
- Deliveries
- Hard bounces
- Complaints
- Delivery delays
- Subscriptions
- Opens
- Clicks

To learn more about setting up event publishing, see the section called “Monitor email sending using event publishing”.

Creating an event destination

After you’ve created a configuration set, you have the option to create event destinations for your configuration set which enables event publishing that is triggered on the event types you specify for the event destination. A configuration set can have multiple event destinations with multiple event types defined.

If you haven't created a configuration set, see the section called “Create configuration sets”.

The following steps show how to create or add an event destination to a configuration set.

To create or add and event destination using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. Choose a configuration set's name from the **Name** column to access its details.

4. Select the **Event destinations** tab.
5. Choose **Add destination**.
6. **Select event types**

   Email sending events are metrics relating to your sending activity that you can measure using Amazon SES. In this step, you select which types of email sending events you would like Amazon SES to publish to your event destination.

   To learn more about event types, see Monitoring your Amazon SES sending activity.

   a. Choose **Event types** to publish

      - **Sending and delivery** – to choose event types to publish, select their respective check boxes, or choose **Select all** to publish all of the event types.

        Event types

        - **Sends** – the send request was successful and Amazon SES will attempt to deliver the message to the recipient's mail server.
        - **Rendering failures** – the email wasn't sent because of a template rendering issue. This event type can occur when template data is missing, or when there is a mismatch between template parameters and data. (This event type only occurs when you send email using the `SendTemplatedEmail` or `SendBulkTemplatedEmail` API operations.)
        - **Rejects** – Amazon SES accepted the email, but determined that it contained a virus and didn't attempt to deliver it to the recipient's mail server.
        - **Deliveries** – Amazon SES successfully delivered the email to the recipient's mail server.
        - **Hard bounces** – the recipient's mail server permanently rejected the email. (*Soft bounces* are only included when Amazon SES fails to deliver the email after retrying for a period of time.)
        - **Complaints** – the email was successfully delivered to the recipient's mail server, but the recipient marked it as spam.
        - **Delivery delays** – the email couldn't be delivered to the recipient's mail server because a temporary issue occurred. Delivery delays can occur, for example, when the recipient's inbox is full, or when the receiving email server experiences a transient issue. *(This event type not supported by Amazon Pinpoint.)*

- **Subscriptions** – the email was successfully delivered, but the recipient updated the subscription preferences by clicking `List-Unsubscribe` in the email header or the `Unsubscribe` link in the footer. *(This event type not supported by Amazon Pinpoint.)*

- **Open and click tracking** – to measure subscriber engagement, choose one or both of the check boxes to track **Opens** and **Clicks**.
  - **Opens** – the recipient received the message and opened it in their email client.
  - **Clicks** – the recipient clicked one or more links in the email.

> **Note**
>
> *Open and click event publishing* defined here, or in any other configuration set, does not affect engagement tracking options for Virtual Deliverability Manager dashboard; these are defined through either Virtual Deliverability Manager's account settings or configuration set overrides. For example, if you have engagement tracking disabled through Virtual Deliverability Manager, it will not disable the open and click event publishing you have set up here in SES event destinations.

  - **Configuration set redirect domain** – this field will appear and be prepopulated with the name of the custom redirect domain if you assigned one when creating the configuration set.

> **Note**
>
> You can update the **Custom redirect domain** in the configuration set for open and click tracking under that domain—see Tracking options in Step 4 of Create configuration sets. For more information about configuring custom open and click domains see Configuring custom domains to handle open and click tracking.

b. Choose **Next** to continue.

7. **Specify destination**

An event destination is an AWS service to which email sending events can be published. Choosing the appropriate destination depends on the level of detail you want to capture and how you want to receive the data.

a. **Destination options**

- **Destination type** – when you select the radio button next to the AWS service to publish your events to, a details panel will appear with fields respective to the service. Selecting the links below will give instructions about the service's detail panel:
  - Amazon CloudWatch *(Additional charges apply, see Price per metric for CloudWatch.)*
  - Amazon Data Firehose
  - Amazon EventBridge
  - Amazon Pinpoint *(Does not support **Delivery delays** or **Subscriptions** event types.)*
  - Amazon SNS

  To learn more about using the event publishing model to monitor your email operation, see Monitor email sending using Amazon SES event publishing.
- **Name** – enter the name of the destination for this configuration set. The name can include letters, numbers, dashes, and hyphens.
- **Event publishing** – to turn on event publishing for this destination, select the **Enabled** check box.

b. Choose **Next** to continue.

8. **Review**

   When you are satisfied that your entries are correct, choose **Add destination** to add your event destination.

You can also create an event destination using the Amazon SES console, the Amazon SES API v2, or the Amazon SES CLI v2.

To create an event destination using the SES API:

- For creating an event destination using the SES API, see `CreateConfigurationSetEventDestination`.

Editing, disabling/enabling, or deleting an event destination

Follow these steps to edit, disable/enable, or delete an event destination using the SES console:

To edit, disable/enable, or delete an event destination using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. Choose a configuration set's name from the **Name** column to access its details.
4. Select the configuration set's **Event destinations** tab.
5. Select the event destination's name under the **Name** column.
6. - **To edit** – Choose the **Edit** button on the respective panel for the set of fields you want to edit and make your changes followed by **Save changes**.
   - **To disable or enable** – Choose the button that's either labeled **Disable** or **Enable** in the upper-right corner.
   - **To delete** – Choose the **Delete** button in the upper-right corner.

You can also edit, disable/enable, or delete an event destination using the Amazon SES console, the Amazon SES API v2, or the Amazon SES CLI v2.

To edit, disable/enable, or delete an event destination using the SES API:

1. For disabling/enabling an event destination using the SES API, see `UpdateConfigurationSetEventDestination`.
2. For deleting an event destination using the SES API, see `DeleteConfigurationSetEventDestination`.

Assigning IP pools in Amazon SES

You can use IP pools to create groups of dedicated IP addresses for sending specific types of email. You can also use a pool of IP addresses that are shared by all Amazon SES customers.

When assigning an IP pool to a configuration set, you can choose from the following options:

- *A specific dedicated IP pool* – When you select an existing dedicated IP pool, emails that use the configuration set are sent using only the dedicated IP addresses that belong to that pool. For procedures on how to create:
  - new *standard* IP pools, see Creating standard dedicated IP pools for dedicated IPs (standard).
  - new *managed* IP pools, see Creating a managed IP pool to enable dedicated IPs (managed).

- **ses-default-dedicated-pool** – This pool contains all of the dedicated IP addresses for your account that do not already belong to an IP pool. If you send an email using a configuration set that is not associated with a pool, or if you send an email without specifying a configuration set at all, the email is sent from one of the addresses in this default pool. This pool is automatically managed by SES and cannot be edited.
- **ses-shared-pool** – This pool contains a large set of IP addresses that are shared among all Amazon SES customers. This option may be useful when you need to send email that doesn't align with your usual sending behaviors.

Assigning an IP pool to a configuration set

This section references the procedures for assigning and modifying IP pools in a configuration set using the Amazon SES console.

- **To assign an IP pool to a configuration set using the console**...
  - **while creating a new configuration set** – see Sending IP pool in Step 4 of Create configuration sets
  - **while modifying an existing configuration set** – select the **Edit** button in the **General details** panel of the selected configuration set, and follow the directions for Sending IP pool in Step 4 of Create configuration sets

Configuring custom domains to handle open and click tracking

When you use event publishing to capture open and click events, Amazon SES makes minor changes to the emails you send. To capture open events, SES adds a 1 pixel by 1 pixel transparent GIF image in each email sent through SES which includes a unique file name for each email, and is hosted on a server operated by SES; when the image is downloaded, SES can tell exactly which message was opened and by whom.

By default, this pixel is inserted at the bottom of the email; however, some email providers' applications truncate the preview of an email when it exceeds a certain size and may provide a link to view the remainder of the message. In this scenario, the SES pixel tracking image does not load and will throw off the open rates you're trying to track. To get around this, you can optionally place the pixel at the beginning of the email, or anywhere else, by inserting the `{{ses:openTracker}}` placeholder into the email body. Once SES receives the message with the placeholder, it will be replaced with open tracking pixel image.

Important

- Any `{{ses:openTracker}}` placeholders in excess of one will be removed at sending by SES.
- Only add one `{{ses:openTracker}}` placeholder if you're using them in an email template, as more than one will result in a `400 BadRequestException` error code being returned.

To capture link click events, SES replaces the links in your emails with links to a server operated by SES. This immediately redirects the recipient to his or her intended destination. The total size of headers, including cookies, of requests made to this server must not exceed 8192 bytes, else a `400 BadRequestException` error code is returned.

You also have the option to use your own domains, rather than domains owned and operated by SES, to create a more cohesive experience for your recipients, meaning all SES indicators are removed. You can configure multiple custom domains to handle open and click tracking events. These custom domains are associated with configuration sets. When you send an email using a configuration set, if that configuration set is configured to use a custom domain, then the open and click links in that email automatically use the custom domain specified in that configuration set.

This section contains procedures for setting up a subdomain on a server you own to automatically redirect users to the open and click tracking servers operated by SES. There are three steps involved in setting up these domains. First, you configure the subdomain itself, then set a configuration set to use the custom domain, and then set its event destination to publish open and click events. This topic contains procedures for completing all of these steps.

However, if you simply want to enable open or click tracking without setting up a custom domain, you can proceed directly to defining event destinations for your configuration set which enables event publishing that is triggered on the event types you specify, including open and click events. A configuration set can have multiple event destinations with multiple event types defined. See Creating Amazon SES event destinations.

Part 1: Setting up a domain for handling open and click link redirects

The specific procedures for setting up a redirect domain vary depending on your web hosting provider (and your Content Delivery Network, if you use an HTTPS server). The procedures in the following sections provide general guidance rather than specific steps.

Option 1: Configuring an HTTP domain

If you plan to use an HTTP domain to handle open and click links (as opposed to an HTTPS domain), the process for configuring the subdomain involves only a few steps.

> **Note**
>
> If you set up a custom domain that uses the HTTP protocol, and you send an email that contains links that use the HTTPS protocol, your customers may see a warning message when they click the links in your email. If you plan to send emails that contain links that use the HTTPS protocol, you should use an HTTPS domain for handling click tracking events.

To set up an HTTP subdomain for handling open and click links

1. Create a subdomain to use for open and click tracking links. SES recommends that this subdomain is specifically dedicated to handling these links, and that a subdomain is created for each AWS Region you send email in that you want to track.
2. Verify the subdomain for use with SES. For more information, see Creating a domain identity.
3. Add a new CNAME record to your subdomain's DNS settings that redirects requests to the SES tracking domain. The address that you redirect to must be the in the same AWS Region as your custom subdomain.

   - Use the Tracking domains table in the AWS General Reference to select the tracking domain that's in the same region as your custom domain.

   > **Note**
   >
   > Depending on your web hosting provider, it may take several minutes for the changes you make to the subdomain's DNS record to take effect. Your web hosting provider or IT organization can provide additional information about these delays.

Option 2: Configuring an HTTPS domain

You can also use an HTTPS domain for tracking open and link clicks. To set up an HTTPS domain for tracking open and link clicks, you have to perform some additional steps, beyond those required for setting up an HTTP domain.

To set up an HTTPS subdomain for handling open and click links

1. Create a subdomain to use for open and click tracking links. SES recommends that this subdomain is specifically dedicated to handling these links, and that a subdomain is created for each AWS Region you send email in that you want to track.
2. Verify the subdomain for use with SES. For more information, see Creating a domain identity.
3. Create a new account with a Content Delivery Network (CDN), such as Amazon CloudFront, see Get started with a basic CloudFront distribution.
4. Configure the CDN to the origin which is the SES tracking domain, such as `r.us-east-1.awstrack.me` for example. The CDN must point to the AWS tracking domain that's in the same region as your custom domain. The CDN must pass the `Host` header supplied by the requester to the origin, see this AWS re:Post article for more information.

   - Use the Tracking domains table in the AWS General Reference to select the tracking domain that's in the same region as your custom domain.
5. If you use Route 53 to manage the DNS configuration for your domain and CloudFront as your CDN, create an Alias record in Route 53 that refers to your CloudFront distribution (such as *d111111abcdef8.cloudfront.net*). For more information, see Creating Records by Using the Amazon Route 53 Console in the *Amazon Route 53 Developer Guide*.

   Otherwise, in the DNS configuration for your subdomain, add a CNAME record that refers to the address of your CDN.
6. Acquire an SSL certificate from a trusted Certificate Authority. The certificate should cover both the subdomain you created in step 1 as well as the CDN you configured in steps 3–5. Upload the certificate to the CDN.
7. You can use the following curl command to validate that your newly created custom domain is using the correct region and HTTPS protocol. In the following example, everything is a literal except for the name of your domain:

   ```
   curl --head https://custom.domain.com/favicon.ico
   ```

A response is returned as in the following example:

```
(python-sdk-test) jdoe@12a34567b89c BaconRedirectService % curl --head https://
custom.domain.com/favicon.ico
HTTPS/1.1 200 OK
x-amz-ses-region: us-east-1
x-amz-ses-request-protocol: https
Content-Type: image/x-icon
Transfer-Encoding: chunked
Date: Fri, 30 Aug 2024 13:50:14 GMT
```

This response contains the following properties:

- The `x-amz-ses-region` header value is the SES region which received the request.
- The `x-amz-ses-request-protocol` header value is the protocol used for the request between the CDN and SES in the header.

If your setup is correct, the region should reflect the region your domain was created in and the protocol should be HTTPS.

Part 2: Specifying your custom redirect domain and HTTPS policy through a configuration set

After you configure your domain to handle open and click tracking redirects, you must specify your custom domain and HTTPS policy in a configuration set.

When you send an email using a configuration set, if that configuration set is configured to use a custom redirect domain, the open and click links in that email automatically use the custom domain and HTTPS policy options specified in the configuration set.

You can complete this using the SES console or the `CreateConfigurationSet` v2 API operation.

To specify a custom redirect domain and HTTPS policy using the console

- While creating or editing a configuration set, use the Tracking options in Step 4 of Create configuration sets to specify your custom redirect domain and HTTPS policy options.

To specify a custom redirect domain and HTTPS policy using the AWS CLI

You can use the CreateConfigurationSet operation in the SES API v2 and use the TrackingOptions property to specify your custom redirect domain and the HTTPS policy. You can call this operation from the AWS CLI as shown in the following example.

- Create the configuration set in the AWS Region where you want to send and track email:

```
aws sesv2 create-configuration-set --cli-input-json file://create.json
```

- In this example, the input file is using parameters of the TrackingOptions property —CustomRedirectDomain specifies the custom domain to use for tracking open and click links, and HttpsPolicy specifies an HTTPS policy option:

```
{
    "ConfigurationSetName": "my-config-set",
    "TrackingOptions": {
        "CustomRedirectDomain": "marketing.example.com",
        "HttpsPolicy": "REQUIRE"
    },
    "SendingOptions": {
        "SendingEnabled": true
    }
}
```

  For the HttpsPolicy parameter, the following values can be specified to set the protocol of the open and click tracking links for your custom redirect domain:

  - OPTIONAL – (Default behavior) Open tracking links will be wrapped using HTTP. Click tracking links will be wrapped using the original protocol of the link.
  - REQUIRE – Open and Click tracking links will both be wrapped using HTTPS.
  - REQUIRE_OPEN_ONLY – Open tracking links will be wrapped using HTTPS. Click tracking links will be wrapped using the original protocol of the link.

Part 3: Specifying open and click event types through a configuration set

After specifying your custom domain and HTTPS policy in the configuration set in the previous step, you must specify open and/or click event types to track in an event destination through a configuration set.

You can complete this using the SES console or the CreateConfigurationSetEventDestination v2 API operation.

To select open and/or click event types using the console

- While creating or modifying an event destination, use Open and click tracking in Step 6 of the section called "Creating an event destination" to specify the event types.

Part 4 (Optional): Configuring custom path segments for Universal Links and App Links

This section is optional. Complete it only if you have a mobile app and want iOS Universal Links or Android App Links to open your app directly when recipients click tracking links. Complete the subsection for each mobile platform your app supports—iOS, Android, or both. Before you begin, complete Part 1 to set up a custom redirect domain and Part 2 to configure your configuration set to use that domain.

If you use iOS Universal Links or Android App Links, the operating system checks a verification file hosted on the click-tracking domain to decide which app should open the link. If the app is installed, the link opens directly in the app. If the app is not installed, the link falls back to the web browser. Because these verification files use path-prefix matching, you need control over the path segment in your tracking URLs.

To configure deep linking with custom path segments, complete these steps:

1. Host a verification file on your custom redirect domain.
2. Add the `ses:custom-path` attribute to links in your email HTML.

In the following instructions, `{your-custom-redirect-domain}` is the custom redirect domain you configured in Part 1 (for example, `click.mail.yourcompany.com`).

Step 1: Host a verification file

iOS Universal Links

To authorize your iOS app to handle Universal Links on your custom redirect domain, host the following file at `https://{your-custom-redirect-domain}/.well-known/apple-app-site-association`:

```
{
  "applinks": {
```

```
    "apps": [],
    "details": [
      {
        "appID": "TEAMID.com.example.myapp",
        "paths": ["/CL1/myapp/*"]
      }
    ]
  }
}
```

- Replace *TEAMID* with your Apple Team ID (find it at `https://developer.apple.com/account/#/membership/`)
- Replace *com.example.myapp* with your app's bundle identifier
- Replace *myapp* in the path with the value you will use in `ses:custom-path`
- The file must be served over HTTPS with content type `application/json`

After hosting this file, add your custom redirect domain to your app's Associated Domains in Xcode (`applinks:{your-custom-redirect-domain}`). For full setup details, see the Apple Universal Links documentation on the Apple Developer website.

Android App Links

To authorize your Android app to handle App Links on your custom redirect domain, host the following file at `https://{your-custom-redirect-domain}/.well-known/assetlinks.json`:

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "com.example.myapp",
    "sha256_cert_fingerprints": ["certificate-fingerprint"]
  }
}]
```

- Replace *com.example.myapp* with your app's package name
- Replace *certificate-fingerprint* with your app's signing certificate SHA-256 fingerprint

After hosting this file, add an intent filter in your app's `AndroidManifest.xml` with your custom redirect domain and `android:pathPrefix="/CL1/myapp/"`, where *myapp* is the value you will use in `ses:custom-path`. For full setup details, see the Android App Links documentation on the Android Developers website.

Step 2: Add ses:custom-path to your email HTML

Use the `ses:custom-path` attribute on individual `<a>` tags in your email HTML to add a fixed path segment to the tracking URL:

```
<a href="https://example.com/product/123" ses:custom-path="myapp">View product</a>
```

The following examples show the difference between a tracking URL without and with `ses:custom-path`:

- Without `ses:custom-path` (default): `https://{your-custom-redirect-domain}/CL0/{encodedUrl}/{index}/{messageId}/{hmac}`
- With `ses:custom-path="myapp"`: `https://{your-custom-redirect-domain}/CL1/myapp/{encodedUrl}/{index}/{messageId}/{hmac}`

The `ses:custom-path` value must meet the following requirements:

- Letters (A–Z, a–z), digits (0–9), hyphens (-), periods (.), or underscores (_) only
- 1–32 characters, case-sensitive
- SES removes this attribute from the email before delivery
- Falls back to the default format if the value is invalid (same as omitting `ses:custom-path`)

The `ses:custom-path` value does not appear in click event data. To attach custom metadata to click events for link identification or analytics, use the optional `ses:tags` attribute:

```
<a href="https://example.com/product/123"
   ses:custom-path="myapp"
   ses:tags="category:product">View product</a>
```

Tag values appear in the click event's `linkTags` field. For details, see Can I tag links with unique identifiers?.

Specifying a configuration set when you send email

To use a configuration set when sending an email, you must pass the name of the configuration set in the headers of the email. All of the Amazon SES email sending methods—including the AWS CLI, the AWS SDKs, and the Amazon SES SMTP interface—allow you to pass a configuration set in the headers of the email you send.

If you are using the SMTP interface or the `SendRawEmail` API operation, you can specify a configuration set by including the following header in your email (replacing *`ConfigSet`* with the name of the configuration set you want to use):

```
X-SES-CONFIGURATION-SET: ConfigSet
```

This guide includes code examples for sending email using the AWS SDKs and the Amazon SES SMTP interface. Each of these examples includes a method of specifying a configuration set. To see step-by-step procedures for sending emails that include references to configuration sets, see the following:

- Sending email through Amazon SES using an AWS SDK
- Using the Amazon SES SMTP interface to send email

Viewing and exporting reputation metrics

Amazon SES automatically exports information about the overall bounce and complaint rates for your entire account to Amazon CloudWatch. You can use these metrics to create alarms in CloudWatch, or to automatically pause email sending using a Lambda function.

You can also export reputation metrics for individual configuration sets to CloudWatch. Exporting reputation data at the configuration set level gives you more control over your sender reputation.

This section includes procedures for exporting reputation data for individual configuration sets to CloudWatch by using the Amazon SES API.

Enabling the export of reputation metrics

To start exporting reputation metrics for a configuration set, use the `UpdateConfigurationSetReputationMetricsEnabled` API operation. To access the Amazon SES API, we recommend using the AWS CLI or one of the AWS SDKs.

This procedure assumes that the AWS CLI is installed on your computer and properly configured. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To enable the exporting of reputation metrics for a configuration set

- At the command line, type the following command:

```
aws ses update-configuration-set-reputation-metrics-enabled --configuration-set-
name ConfigSet --enabled
```

  Replace *ConfigSet* in the preceding command with the name of the configuration set for which you want to start exporting reputation metrics.

Disabling the export of reputation metrics

You can also use the `UpdateConfigurationSetReputationMetricsEnabled` API operation to disable the exporting of reputation metrics for a configuration set.

To disable the exporting of reputation metrics for a configuration set

- At the command line, type the following command:

```
aws ses update-configuration-set-reputation-metrics-enabled --configuration-set-
name ConfigSet --no-enabled
```

  Replace *ConfigSet* in the preceding command with the name of the configuration set for which you want to disable the exporting of reputation metrics.

Using Global endpoints in Amazon SES

Amazon SES Global endpoints is a feature that enhances the continuity and reliability of your email sending operations. This chapter will guide you through the concept, setup, and usage of Global endpoints, helping you leverage multi-region sending (MRS) to achieve higher availability and improved disaster recovery capabilities for your email workloads.

What are Global endpoints?

Global endpoints are resources that allow you to distribute your SES outbound workloads across two AWS Regions. Once configured, SES automatically divides your sending traffic between the selected primary and secondary regions. If either region experiences an impairment, SES will automatically shift traffic away from the affected region to maintain continuity of your sending operations.

Key benefits of using Global endpoints include:

- Improved email sending continuity
- Automatic failover between regions
- Simplified multi-region configuration

How Global endpoints work

When you set up a Global endpoint, you select a primary region (where the endpoint is created) and a secondary region. SES then creates a multi-region endpoint (MREP) that serves as the entry point for your email sending requests.

The Global endpoint setup process synchronizes key artifacts and sending limits from your primary region to your secondary region. This ensures that both regions have equivalent verified identities, configuration sets, and approved sending limits sufficient for all of the expected volume.

Once the Global endpoint is ready and its Endpoint ID is specified in the SendEmail API call, SES automatically routes your outbound traffic between your primary and secondary regions. If either region becomes impaired, traffic will be weighted away from that region towards the other one until the impairment is resolved.

Setting up Global endpoints

Topics

- Prerequisites
- Creating a Global endpoint
- Global endpoint states

Prerequisites

Before creating a Global endpoint you'll first need to grant SES permission to create Service-Linked Roles (SLRs) in your account. These roles enable essential service functionalities and resource access needed to create, use, and monitor Global endpoints. This can be done by implementing the following policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "ses.amazonaws.com"
        }
      }
    }
  ]
}
```

Creating a Global endpoint

To create a new Global endpoint:

1. Open the SES console at https://console.aws.amazon.com/ses/.

2. In the navigation pane, choose **Global Endpoints**.
3. Choose **Create global endpoint** and enter a name in the **Name** field.
4. Select a secondary region from the dropdown menu. (Your primary region defaults to the region you signed into the console with.)
5. (Optional) Add one or more **Tags** to your Global endpoint.
6. Review the configuration and choose **Create global endpoint**.

The creation process may take a few seconds. Once completed, the status of your Global endpoint will change to "Ready."

Using the AWS CLI:

```
aws sesv2 create-multi-region-endpoint --primary-region us-west-2 --secondary-
region us-east-1 --endpoint-name MyGlobalEndpoint
```

In the preceding example:

- Replace *us-west-2* with the primary region for your Global endpoint.
- Replace *us-east-1* with the secondary region for your Global endpoint.
- Replace *MyGlobalEndpoint* with the friendly name to give your Global endpoint.

Global endpoint states

Global endpoints can have the following states:

- *Creating* – The resource is being provisioned
- *Ready* – The resource is ready to use
- *Failed* – The resource failed to be provisioned
- *Deleting* – The resource is being deleted as requested

Preparing the secondary region

Now that you've created your global endpoint, you must now ensure that the your email sending configuration, including all its components (identities, configuration sets, email templates, and sending limits), is consistent across the primary and secondary regions before utilizing the Global

endpoint for sending emails. This alignment is crucial to avoid potential issues and ensure proper email delivery and tracking.

The Region duplication feature in the console assists you by automatically duplicating resources and duplicating account-level settings from the primary to the secondary region that will quickly help you to ensure that both regions have equivalent configurations.

Based on resource dependencies, the order in which you duplicate resources matters. To avoid conflicts, follow the topic order below:

Topics

- Duplicating configuration sets
- Duplicating verified domain identities
- Duplicating production limits

Duplicating configuration sets

You can select multiple configuration sets from your primary region to be duplicated, along with their settings, in the secondary region.

The "Duplicate configuration sets" feature allows you to:

- Duplicate multiple configuration sets into the secondary region at once.
- Check for differences between configuration sets in the primary region and the secondary region.

To duplicate configuration sets:

1. On the **Global endpoints** page, choose the Global endpoint you want to duplicate by selecting it from the **Name** column.
2. In the **Duplicate configuration sets** card, expand **Configuration sets actions** and choose **Duplicate**.
3. Select up to 10 configuration sets followed by **Confirm**.
4. If the status is not successful, choose **View report** to identify the problem.
5. (Optional) For previously duplicated configuration sets, you can check for differences between the primary and secondary regions by selecting **Check differences** while repeating the last three steps.

Note

- If the configuration set you duplicated contains *Event destinations*, *Reputation options*, *Archiving options*, or is referenced in an *Email template*, these settings will need to be manually configured in the secondary region.
- If you have archiving enabled for sent (outbound) email in a configuration set in the primary region, you must manually enable archiving for sent (outbound) email in the secondary region's configuration set using an archive created in the secondary region with the same name.

Duplicating verified domain identities

To ensure the Global endpoint configuration works effectively, your sending domain identity needs to be verified in both the primary and secondary regions. SES uses Deterministic Easy DKIM (DEED) to simplify this process.

Deterministic Easy DKIM (DEED) is a feature that generates consistent DKIM tokens across all AWS Regions based on a parent domain that is configured with Easy DKIM. This consistency allows SES to automatically verify a domain in the secondary region once it's verified in the primary region, without requiring additional DNS record updates. Therefore, you must ensure that the domain identity you want to duplicate, that is the parent, is already configured with Easy DKIM.

The "Duplicate verified domain identities" feature allows you to:

- Duplicate multiple domain identities into the secondary region at once.
- Verify them automatically with Deterministic Easy DKIM (DEED).
- Check for differences between identities in the primary region and the secondary region.

To duplicate identities from the SES console:

1. On the **Global endpoints** page, choose the Global endpoint you want to duplicate by selecting it from the **Name** column.
2. In the **Duplicate verified domain identities** card, expand **Identities actions** and choose **Duplicate**.
3. Select up to 10 identities followed by **Confirm**.

4. If the status is not successful, choose **View report** to identify the problem.
5. (Optional) For previously duplicated identities, you can check for differences between the primary and secondary regions by selecting **Check differences** while repeating the last three steps.

> **Note**
>
> - Domain identities verified with BYODKIM, or are self-signed, will need to be created manually in the secondary region, as DEED is not applicable in this case.
> - Domain identities using *Mail-from attributes*, *Policies*, or *Feedback forwarding and notifications* will need to have these features manually configured in the secondary region.

Duplicating production limits

SES checks if sending limits are aligned between regions and allows you to request limit increases in the secondary region if needed.

The "Duplicate production limits" feature allows you to:

- Check if production limits are aligned between primary and secondary regions.
- Request a limit increase in the secondary region if needed.

To duplicate production limits:

1. On the **Global endpoints** page, choose the Global endpoint you want to duplicate by selecting it from the **Name** column.
2. In the **Duplicate production limits** card, if the status displays *Sending limits not aligned*, expand **Sending limits actions**.
3. Select **Managing sending limits** for the secondary region.
4. The **Service Quotas** page opens in the secondary region where you can request increases to "Sending quota" and "Sending rate" to match the values from the primary region.

> **Tip**
>
> It's recommended that you request the maximum quota you're eligible for in both regions. While email traffic is distributed amongst both regions under normal operating conditions, during a failover event, the full volume of email traffic will be sent to one region and its limits should be enough to handle the full volume load.

5. (Optional) You can also request production increases for your primary region by selecting **Managing sending limits** for the primary region while repeating the previous two steps.

> **Important**
>
> It's crucial that both regions have the equivalent verified identities and configuration sets that you intend to send email with, along with matching sending limits to ensure proper functionality of the Global endpoint. Any discrepancies could cause delivery failures, diminished failover reliability, and missing metrics.

Using Global endpoints

Topics

- Integrating with your application
- Monitoring and metrics

Integrating with your application

Using a Global endpoint in your application requires that you obtain its Endpoint ID.

To retrieve a Global endpoint's Endpoint ID:

1. From the SES console, go to the **Global endpoints** page and choose the Global endpoint you want to use by selecting it from the **Name** column.
2. Select the copy icon under **Endpoint ID** on the Global endpoint details page.

Using the AWS CLI:

```
aws sesv2 get-multi-region-endpoint --endpoint-name MyGlobalEndpoint --region us-west-2
```

In the preceding example:

- Replace *MyGlobalEndpoint* with the friendly name you gave your Global endpoint during creation.
- Replace *us-west-2* with the primary region where you created your Global endpoint.
- The API response will include the value of your Endpoint ID such as, `"EndpointId": "abcdef12.g3h"`.

Once you've obtained the Endpoint ID of your Global endpoint, you can update your SendEmail or SendBulkEmail API calls to include the Endpoint ID value for the `endpoint-id` parameter. Here's an example of how to specify the Endpoint ID in a SendEmail API call using the AWS CLI:

```
aws sesv2 send-email \
                --from-email-address "sender@example.com" \
                --destination "ToAddresses=recipient@example.com" \
                --content "Subject={Data=Test
 email,Charset=UTF-8},Body={Text={Data=This is a test email sent using Amazon SES
 Global endpoints.,Charset=UTF-8}}" \
                --endpoint-id "abcdef12.g3h"
```

Replace *abcdef12.g3h* with the actual Endpoint ID you obtained either through the console or API.

Monitoring and metrics

The Global endpoints feature provides a unified view of your email sending volume across both the primary and secondary regions. You can access these metrics through the Cross-region metrics tab on the Global endpoint details page in the SES console.

To access sending metrics across both regions:

1. From the SES console, go to the **Global endpoints** page, choose the Global endpoint you want to see metrics for by selecting it from the **Name** column.
2. Select the **Cross-region metrics** tab on the Global endpoint details page and enter a date range of up to 31 days. Metrics for both regions will be displayed for the given date range.

Using the AWS CLI:

```
aws cloudwatch get-metric-statistics \
                --namespace AWS/SES \
                --metric-name SendCount \
                --dimensions Name=ses:multi-region-endpoint-id,Value=abcdef12.g3h \
                --start-time 2024-10-01T00:00:00Z \ --end-time 2024-10-31T23:59:59Z \
                --period 86400 \
                --statistics Sum
```

Replace *abcdef12.g3h* with your actual Endpoint ID.

Best practices and considerations

Following these best practices and considerations helps ensure effective utilization, monitoring, and cost optimization of Global endpoints across multiple AWS Regions for improved availability and reliability of email sending capabilities.

- Regularly synchronize any changes made to artifacts (e.g., configuration sets, verified identities) between regions to maintain sending integrity.
- Monitor the Cross-region metrics to ensure balanced traffic distribution and identify any potential issues.
- Be aware that there may be fractional increases in API latency when making calls to MREP-enabled distant regions.
- To ensure balanced traffic distribution, adopt an active-active architecture that mirrors the regions of your Global endpoint. Alternatively, choose two regions that are approximately equally distant from your client application.
- Configure your client application to detect routing changes quickly. Global endpoints publish current routing weights through DNS. Set the positive DNS cache TTL to approximately 60 seconds and the negative DNS cache TTL to approximately 10 seconds in your runtime and resolver configuration. On the JVM, these are controlled by the `networkaddress.cache.ttl` and `networkaddress.cache.negative.ttl` security properties.
- Set HTTP connection lifetime limits in your client. Long-lived keep-alive connections continue to reach the originally resolved IP address even after DNS refreshes. Set a short Connection TTL of approximately 60 seconds on your HTTP client so that pooled connections are replaced regularly. Most AWS SDKs expose this setting.
- Global endpoints do not support SMTP or VPC endpoint access.

- Consider potential egress charges if routing traffic through a NAT Gateway.

Pricing

While exact pricing details are subject to change, Global endpoints are expected to carry a price premium over single-region sending for an equal volume of mail. Despite this increase, the overall cost is anticipated to remain competitive compared to other email service providers.

For the most up-to-date pricing information, please refer to the Amazon SES Pricing page.

Dedicated IP addresses for Amazon SES

When you create a new Amazon SES account, by default your emails are sent from IP addresses that are shared with other SES users. You can also use dedicated IP addresses that are reserved for your exclusive use by leasing them for an additional cost. This gives you complete control over your sender reputation and enables you to isolate your reputation for different segments within email programs. Amazon SES offers two ways to provision and manage a dedicated IP address:

- **Standard**—refers to dedicated IP addresses that you manually set up and manage, including the option to manually warm them up and scale them out, and to manually move them in and out of IP pools. (These were formerly referred to as *dedicated IP addresses* in SES.)
- **Managed**—refers to dedicated IP addresses that are automatically set up on your behalf by SES to provide a quick and easy way to start using dedicated IP addresses that are managed by SES; they automatically warm up for each ISP individually and auto-scale based on your sending volume to help ensure that your dedicated IP addresses are used optimally based on how you send email.

When deciding between shared IP addresses or the two types of dedicated IP addresses defined above, choose the one that provides the most benefits for the type, volume, and patterns of email that you send. To help you make your decision, these benefits are summarized in the following table. Choose an item in the **Benefit** column for additional information.

Benefit	Shared IP addresses	Dedicated IP addresses (standard)	Dedicated IP addresses (managed)
Ready to use immediately	Yes	No	No
Additional setup required	No	Yes	Yes
IP addresses & reputation isolated from other SES customers	No	Yes	Yes

Benefit	Shared IP addresses	Dedicated IP addresses (standard)	Dedicated IP addresses (managed)
Capacity increases automatically as traffic increases	No	No	Yes
Good for customers with continuous, predictable sending patterns	Yes	Yes	Yes
Good for customers with less predictable sending patterns	Yes	No	Yes
Good for high-volume senders	Yes	Yes	Yes
Good for low-volume senders	Yes	No	No
Additional monthly costs	No	Yes	Yes
Complete control over sender reputation	No	Yes	Yes
Isolate reputation by email type, recipient, or other factors	No	Yes	Yes
Provides known IP addresses that never change	No	Yes	No

Important

If you don't plan to send large volumes of email on a regular and predictable basis, we recommend that you use shared IP addresses. If you want to use dedicated IP addresses in situations where your sending patterns are highly irregular, using *Dedicated IPs (managed)* is the better option.

Ease of setup

Shared IP addresses—you don't need to perform any additional configuration. Your SES account is ready to send email as soon as you verify an email address and move out of the sandbox.

Dedicated IP addresses (standard)—you must submit a request through the AWS Support Center and optionally configure dedicated IP pools.

Dedicated IP addresses (managed)—you don't need to submit a request for dedicated IP addresses. They'll automatically be allocated when you opt in and do a one-time walkthrough to create your managed dedicated pool.

Reputation management

IP address reputations are based largely on historical sending patterns and volume. An IP address that sends consistent volumes of email over a long period of time typically has a good reputation.

Shared IP addresses—shared between several SES customers, these addresses collectively send a large volume of email and AWS carefully manages the outbound traffic to maximize the reputations of the shared IP addresses.

Dedicated IP addresses (standard)—after warmup, your IP addresses are isolated from the SES shared pool and you maintain your own sender reputation by sending consistent and predictable volumes of email.

Note

For information about Smart Network Data Services (SNDS) data for your dedicated IPs (standard), see SNDS metrics for dedicated IPs.

Dedicated IP addresses (managed)—after warmup of your new IPs, they're isolated from the SES shared pool and you maintain your own sender reputation. There's the added benefit of tracking the reputation for each ISP and optimally scheduling outgoing sending accordingly. So while you still maintain your sender reputation, this automation helps to improve overall deliverability and reduce bounce rates when compared to equivalent workloads on manually configured dedicated IP addresses.

Predictability of sending patterns

An IP address with a consistent history of sending email has a better reputation than one that suddenly starts sending out large volumes of email with no prior sending history.

Shared IP addresses—good for email sending patterns that don't follow a predictable pattern. With shared IP addresses, you can increase or decrease your email sending patterns as the situation demands.

Dedicated IP addresses (standard)—you must warm up addresses by sending an amount of email that gradually increases every day. The process of warming up new IP addresses is described in Warming up dedicated IP addresses (standard). After your dedicated IP addresses are warmed up, you must then maintain a consistent sending pattern.

Dedicated IP addresses (managed)—your dedicated IP addresses are warmed up automatically for each IP in the managed pool by using an adaptive warmup strategy (in conjunction with the SES shared pool) that takes into account actual sending patterns to optimize the warmup for each ISP individually. The managed IP pool automatically scales out per ISP based on usage and consideration of ISP-specific policies.

Volume of outbound email

Shared IP addresses —best for customers who send low volumes of email.

Dedicated IP addresses (standard) | Dedicated IP addresses (managed)—both are suited for customers who send large volumes of email. Most ISPs only track the reputation of a given IP address if they receive a significant volume of mail from that address. For each ISP with which you want to cultivate a reputation, you should send several hundred emails within a 24-hour period at least once per month. In some cases, both types of dedicated IP addresses may also work for smaller volumes of email. For example, they may work well if you send to a small, well-defined group of recipients whose mail servers accept or reject email using a list of specific IP addresses, rather than IP address reputation.

Additional costs

Shared IP addresses—included in the standard SES pricing.

Dedicated IP addresses (standard)—are available for an additional monthly fee per IP address that you lease. For pricing information, see the SES pricing page.

Dedicated IP addresses (managed)—are available for a standard monthly fee (regardless of the amount of IPs needed) and a per message usage charge. For pricing information, see the SES pricing page.

Control over sender reputation

Shared IP addresses—your sender reputation is controlled by SES.

Dedicated IP addresses (standard) | Dedicated IP addresses (managed)—your sender reputation is completely under your control. Your SES account is the only one that is able to send email from those addresses. For this reason, the sender reputation is determined by your email sending practices. Additionally, dedicated IPs (managed) actively monitors outbound IP addresses used for email sending by using the highest performing IP addresses to improve email deliverability to your recipients. Utilization data can be surfaced by using additional services such as Amazon CloudWatch metrics and the built-in dashboards that are in Amazon SES.

Ability to isolate sender reputation

Shared IP addresses—your sender reputation is set at the account level and can't be isolated.

Dedicated IP addresses (standard) | Dedicated IP addresses (managed)—you can isolate your sender reputation for different components within your email program by creating *dedicated IP pools*—groups of dedicated IP addresses that can be used for sending specific types of email. For example, you can create one pool of dedicated IP addresses for sending marketing email, and another for sending transactional email.

Known, unchanging IP addresses

Shared IP addresses—you don't know the IP addresses that SES uses to send your mail, and they can change at any time.

Dedicated IP addresses (standard)—you can find the values of the addresses that send your mail in the **Dedicated IPs** page of the SES console. This is because dedicated IP addresses are static.

Dedicated IP addresses (managed)—SES will automatically configure the optimal number of dedicated IP addresses based on your sending patterns. While SES manages the automatic scaling of your IP pool, you can view all dedicated IP addresses currently allocated to your account through the SES console or API. The number of IPs in your pool will continue to dynamically increase or decrease based upon your sending demand.

Dedicated IP addresses (standard) in Amazon SES

Dedicated IP addresses (standard) are dedicated IP addresses that you manually set up and manage in SES. They are different from those that are set up and managed automatically using the SES feature the section called "Dedicated IP addresses (managed)". In addition to allowing you full control over your sending reputation using dedicated IP addresses, dedicated IPs (standard) enable you to fully manage your dedicated IPs, including warming them up, scaling them out, and IP pool management.

Dedicated IPs (standard) and *Dedicated IPs (managed)* both refer to dedicated IP addresses that you lease in SES for additional pricing, but differ in how they're implemented and managed. While there are shared benefits common to both, they each have unique advantages to offer depending on your type of email sending, as discussed in *Dedicated IP addresses*.

The topics is this section explain how to manually set up and manage dedicated IPs (standard) in SES.

Topics

- Requesting and relinquishing dedicated IP addresses (standard)
- Warming up dedicated IP addresses (standard)
- Creating standard dedicated IP pools for dedicated IPs (standard)

Requesting and relinquishing dedicated IP addresses (standard)

To use *dedicated IP addresses (standard)*, you must first request them. When you no longer need them, you must relinquish them. Request and relinquish dedicated IPs (standard) through the AWS Support Center. Your account is charged an additional monthly fee for each standard dedicated IP address that you lease for use with Amazon SES. There's no minimum commitment when using dedicated IPs (standard).

For more information about the costs associated with dedicated IPs (standard), see Amazon SES Pricing.

For a list of all of the Regions where Amazon SES is currently available, see AWS Region and Endpoints in the *Amazon Web Services General Reference*. To learn more about the number of Availability Zones that are available in each AWS Region, see AWS Global Infrastructure.

Request or relinquish dedicated IPs (standard)

You can request as many dedicated IPs (standard) as you need by creating a service quota increase case in the AWS Support Center.

To request or relinquish dedicated IPs (standard)

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Do one of the following:

   a. If you *don't* have existing dedicated IPs in your account:

      - The **Dedicated IPs** onboarding page is displayed. In the **Dedicated IPs (standard) overview** panel, choose **Request dedicated IPs**.

   b. If you have existing dedicated IPs in your account:

      i. Select the **Standard IP pools** tab on the **Dedicated IPs** page.

      ii. In the **Standard overview** panel, choose **Request or relinquish Standard dedicated IPs**.
4. The **Hello! We're here to help** page opens in the AWS Support Console. All the fields on this page will have the following values preselected:

   - **Choose the related issue for your case** – *Account and billing*
   - **Service** – *Service Quotas*
   - **Category** – *Amazon SES*
   - **Severity** – *General question*

   After verifying these values, choose **Next step: Additional information**.
5. Under **Additional information**, complete the following selections:

- For **Region**, select the AWS Region that your request applies to.
- For **Quota Title**, select **Desired Dedicated IP**.
- For **Value**, select the number of dedicated IPs you are requesting or relinquishing in the region selected.
- If you want to request or relinquish dedicated IPs (standard) in another AWS Region, choose **Add another limit**, and fill in the fields accordingly. Repeat for each additional AWS Region.
- For **Description**, make it clear what you have and what you want to do in each region specified as in the following examples:

  Request – *"I have two DIPs in the Milan region, but would like to add one more for a total of three"*

  Relinquish – *"I have two DIPs in the Ohio region, but want to remove one of them—please remove the DIP that has the address 23.251.228.95"*.

  > **Important**
  >
  > The process of relinquishing a dedicated IP address can't be reversed. If you relinquish a *dedicated IP address* in the middle of a month, we prorate the monthly dedicated IP usage fee, based on the number of days that have elapsed in the current month.

- Choose **Next step: Solve now or contact us**.
- On the **Solve now or contact us** page, select your preferred contact language, and choose **Submit**.

After you submit the form, we'll evaluate your request. If we grant your request, we'll reply to your case in the Support Center to confirm that the dedicated IP addresses have been added to or removed from your account according to your request.

Warming up dedicated IP addresses (standard)

When determining whether to accept or reject a message, email service providers consider the reputation of the IP address that sent it. One of the factors that contributes to the reputation of an IP address is whether the address has a history of sending high-quality email. Email providers are less likely to accept mail from new IP addresses that have little or no history. Email sent from IP

addresses with little or no history might end up in recipients' junk mail folders, or might be blocked altogether.

When you start sending email from a new dedicated IP address, you should gradually increase the amount of email that you send from that address before using it to its full capacity. This process is called *warming up* the IP address.

The amount of time that's required to warm up an IP address varies between email providers. For some email providers, you can establish a positive reputation in around two weeks, while for others it may take up to six weeks. When warming up a new dedicated IP address, you should send emails to your most active users to ensure that your complaint rate remains low. You should also carefully examine your bounce messages and send less email if you receive a high number of blocking or throttling notifications. For information about monitoring your bounces, see Monitoring your Amazon SES sending activity.

Automatic warmup for dedicated IPs (standard)

When you request *dedicated IP addresses (standard)*, Amazon SES automatically warms them up to improve the delivery of emails that you send. The automatic IP address warmup feature is enabled by default. SES automatically warms up your dedicated IPs by gradually increasing the number of emails you send through your dedicated IPs based on a predefined warmup plan. This gradual increase helps your IPs build a positive reputation with internet service providers (ISPs).

The steps that happen during the automatic warmup process depend on whether you already have dedicated IP addresses.

- When you request dedicated IPs (standard) for the first time, SES distributes your email sending between your dedicated IP addresses and a set of addresses that are shared with other SES customers. SES gradually increases the number of messages that are sent from your dedicated IP addresses over time.
- If you already have dedicated IP addresses, SES distributes your email sending between your existing dedicated IPs (which are already warmed up) and your new dedicated IPs (which are not warmed up). SES gradually increases the number of messages that are sent from your new dedicated IP addresses over time.

> **Note**
>
> Automatic IP warmup is a time-based process. The warmup percentage steadily increases over 45 days, independently from your sending volume.

After you warm up a dedicated IP address, you should send around 1,000 emails every day to each email provider that you want to maintain a positive reputation with. You should perform this task on each dedicated IP address that you use with SES.

You should avoid sending large volumes of email immediately after the warmup process is complete. Instead, slowly increase the number of emails you send until you reach your target volume. If an email provider sees a large, sudden increase in the number of emails that are sent from an IP address, they may block or throttle the delivery of messages from that address.

Disable the automatic warmup process on dedicated IPs (standard)

When you purchase new standard dedicated IP addresses, Amazon SES automatically warms them up for you because the automatic IP address warmup feature is enabled by default for your account. If you prefer to warm up dedicated IP addresses yourself, you can disable the automatic warmup feature at the account level for all of your IP addresses.

If you disable the automatic warmup feature, any subsequently leased dedicated IPs will be added to your account with a warmup status of *Complete* which makes them available for use without having been warmed up—this means you are responsible for ensuring these IPs are properly warmed up before using them for regular sending. Any IPs that were currently in the middle of warmup at the time you disabled the automatic warmup feature will not be effected.

> **Important**
>
> If you disable the automatic warm up feature, you're responsible for warming up your dedicated IP addresses yourself. If you send email from addresses that haven't been warmed up, you may experience poor delivery rates.

To disable (or re-enable) the automatic warmup feature for all dedicated IPs (standard) in your account

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Select the **Standard IP pools** tab on the **Dedicated IPs** page.
4. Choose **Disable auto warm-up** in the **Standard overview** panel to disable automatic warmup, or choose **Enable auto warm-up** to re-enable automatic warmup.

Manually warm up dedicated IPs (standard)

You can manually increase or decrease your dedicated IPs (standard) current sending volume by editing its warmup percentage, end its warmup process prematurely, and set its current sending volume to 0% and restart the warmup process.

To manually warm up dedicated IPs (standard)

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Select the **Standard IP pools** tab on the **Dedicated IPs** page.
4. In the **All Standard dedicated IPs** panel, select an IP address and choose **Edit warm up** and select one of the following options:

    a. **Edit percentage**—enter a value in the **Warm-up percentage** field to increase or decrease your IP's current sending volume by editing its warmup percentage followed by **Save changes**.

    The **Warm-up status** column will say `In progress` and the **Warm-up percentage** column will display the value that you entered.

    b. **Mark as Complete**—read the **Mark warm-up as Complete?** dialogue to confirm that you understand the implications of ending the auto warm-up process prematurely, then choose **Mark as Complete**.

The **Warm-up status** column will say `Complete` and the **Warm-up percentage** column will say `100%`.

c. **Reset percentage**—read the **Reset warm-up percentage?** dialogue to confirm you're setting the IP's current sending volume to 1% and will have to either restart the automatic warmup process or set the warmup percentage manually, then choose **Reset**.

The **Warm-up status** column will say `In progress` and the **Warm-up percentage** column will say `1%`.

Creating standard dedicated IP pools for dedicated IPs (standard)

If you purchased several dedicated IP addresses (standard) to use with Amazon SES, you can create groups of those addresses, called *dedicated IP pools*. Grouping dedicated IPs (standard) together in a pool makes them easier to manage. A common scenario is to create one pool for sending marketing communications, and another for sending transactional emails. Your sender reputation for transactional emails is then isolated from that of your marketing emails. In this scenario, if a marketing campaign generates a large number of complaints, the delivery of your transactional emails is not impacted.

This section contains procedures for creating dedicated IP pools.

> **Note**
>
> You can also create configuration sets that use a pool of IP addresses that are shared by all SES customers. The shared IP pool is useful for situations in which you need to send email that doesn't align with your usual sending behaviors. For information about using the shared IP pool with a configuration set, see Assigning IP pools in Amazon SES.

To create a dedicated IP pool for dedicated IPs (standard) using the SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.

> **Note**
>
> If you currently don't have any dedicated IPs (standard) in your account, the **Dedicated IPs** onboarding page is displayed giving you the opportunity to purchase dedicated IPs (standard). For more information see the section called "Request or relinquish dedicated IPs (standard)".

3. Select the **Standard IP pools** tab on the **Dedicated IPs** page.
4. In the **All Dedicated IP (standard) pools** panel, choose **Create Standard IP pool**.

   The **Create IP Pool** page opens.
5. In the **Pool details** panel,

   a. Choose **Standard (self managed)** in the **Scaling mode** field.

   b. Enter a name for your IP pool in the **IP pool name** field.

   > **Note**
   >
   > The IP pool name must be unique and can't be a duplicate of a managed IP pool name in your account.

   c. (Optional) If you have existing standard dedicated IP addresses that you want to add to this IP pool, select them from the dropdown list in the **Dedicated IP addresses** field.

   > **Note**
   >
   > If you select an IP address that's already associated with an IP pool, it will now only be associated with this IP pool.
6. (Optional) You can associate this IP pool with a configuration set by selecting one from the dropdown list in the **Configuration sets** field.

> **Note**
>
> - If you select a configuration set that's already associated with an IP pool, it will now only be associated with this IP pool.

- To add or remove associated configuration sets after this IP pool is created, edit the configuration set's **Sending IP pool** parameter.
- If you haven't created any configuration sets yet, see *Configuration sets*.

7. (Optional) You can add one or more **Tags** to this IP pool by including a tag key and an optional value for the key.

   a. Choose **Add new tag** and enter the **Key**. You can also add an optional **Value** for the tag.

   b. To add the tag, choose **Save changes**.

   You can add up to 50 tags. You can remove a tag by choosing **Remove**.

8. Select **Create pool**.

> **Note**
>
> After you create a standard IP pool, you have the option to convert it to a managed IP pool. See Creating a managed IP pool.

Dedicated IP addresses (managed) for Amazon SES

Dedicated IP addresses (managed) is an Amazon SES feature that automatically sets up and manages dedicated IP addresses on your behalf to provide a quick and easy way to start using dedicated IP addresses that are managed by SES. This helps to ensure that your dedicated IP addresses are used efficiently and optimally for how you send email.

To enable dedicated IPs (managed) in your account, you just create a managed IP pool and SES does all the rest. SES will determine how many dedicated IPs you require based on your sending patterns, create them for you, and then manage how they scale based on your sending requirements.

Once enabled, you can utilize dedicated IPs (managed) in your email sending by associating the managed IP pool with a configuration set, and then specifying that configuration set when sending email. The configuration set can also be applied to a sending identity by using a default configuration set.

Benefits and features of dedicated IPs (managed)

The dedicated IP addresses that you create with dedicated IPs (managed) automate management tasks to help ensure that your dedicated IP addresses are used in a way that's optimal for how you send email:

- **Easy onboarding** – To get started with dedicated IPs (managed), you create a managed IP pool directly from the SES console. Dedicated IP addresses are automatically allocated to the pool. You can start sending with the managed IP pool without having to open a request case through the AWS Support Center.
- **Auto-scaling per ISP** – You don't have to manually monitor or scale your dedicated IP pools because the managed IP pool scales out automatically based on usage. It also takes into consideration ISP-specific policies. For example, if SES detects that an ISP supports a low daily send quota, the pool scales out to better distribute traffic to that ISP across more IP addresses.
- **Intelligent warmup** – Dedicated IPs (managed) start to send mail to ISPs based on their capacity. That is, how much they are currently warmed up. They automatically keep track of the level of warmup for each ISP individually. Additionally, the dedicated IPs (managed) feature provides information about your reputation at an effective daily rate with top ISPs in the form of Amazon CloudWatch metrics and built-in dashboards.
  - **Warmup per ISP** – SES tracks the reputation for each IP in the managed IP pool for each ISP individually. For example, if you've been sending all your traffic to Gmail, the IP addresses are considered warmed up only for Gmail and cold for other ISPs. If you change your traffic pattern by ramping up email sent to Hotmail, SES ramps up traffic slowly for Hotmail, as the IP addresses are not warmed up yet.
  - **Adaptive warmup & Shared pool transitioning** – The warmup adjustment is adaptive and takes into account actual sending patterns. When sending volume to an ISP drops, the warmup percentage also drops for that ISP. In the early phase of warmup, any sending that's excessive based on the current level of warmup *is sent through the IP addresses that are shared with other Amazon SES users—the SES shared pool*. In later stages of warmup, any sending that's excessive is proactively slowed down and retried later.

> **⚠ Important**
>
> While dedicated IPs (managed) automatically warms up your dedicated IP addresses, part of that automatic process is working interactively with the SES shared IP pool.

- If your sending rate is too aggressive for your new dedicated IPs while they're being warmed up, SES will automatically spill part of your sending over into the SES shared IP pool to protect the reputation of your new dedicated IPs.
- Even after your new dedicated IPs are fully warmed up, it isn't guaranteed that all of your sending will go through them 100% of the time. For example, if your sending rate suddenly rises and dedicated IPs (managed) determines it must allocate an additional dedicated IP address, it will initiate the warmup process which includes using the shared pool. Likewise, if your sending rate suddenly drops very low, all of your sending could switch over to the SES shared IP pool, see the section called "Importance of warmup".

- **Automatic request & relinquish of dedicated IP addresses** – You don't need to request or relinquish managed dedicated IP addresses through the AWS Support Center, as is required when using dedicated IPs (standard). When onboarding with dedicated IPs (managed) directly from the SES console, CLI, or API, you are automatically allocated dedicated IP addresses and charged a fee based on the volume of messages that you send. When you delete an IP pool that's created by dedicated IPs (managed) or opt out of dedicated IPs (managed), your allocated IP addresses are automatically relinquished and charges cease immediately.
  - **Getting your first dedicated IP address** – The dedicated IPs (managed) feature will automatically allocate your first dedicated IP address once your sending volume reaches hundreds of emails over a period of a few days. This ensures that the IP you send from can build a sending reputation and improve deliverability. (If you don't expect your sending volume to be at this level, you should be using shared IP addresses. See the comparison table in *Dedicated IP addresses* to review the type of IP addresses that are best for how you send email.)

Why proper IP warmup is important

To ensure that your email will be delivered through your dedicated IP address, it must have a good reputation with the receiving ISP. ISPs will only accept a small volume of email from an IP that they don't recognize. When you're first allocated an IP, it's new and won't be recognized by the receiving ISP because it doesn't have a reputation associated with it. In order for an IP's reputation to be established, it must gradually build trust with receiving ISP—this gradual trust building process is referred to as *warming-up*. Immediately after dedicated IPs (managed) allocates an IP, it starts the Intelligent warmup process.

With the Warmup per ISP and Adaptive warmup features of dedicated IPs (managed), business continuity is maintained throughout the warmup cycle by ensuring that your email will be delivered. Once the warmup phase is complete, any excess capacity is queued and sent only through the dedicated IP pool. However, if you have one dedicated IP address and your sending falls below the minimum volume required to maintain IP reputation, *dedicated IPs (managed) may remove your dedicated IP and your sending will be routed through the SES shared IP pool*.

> **Note**
>
> If you send small volumes of email (less than a few hundred per day over a few days), it would be more beneficial to send through the SES shared IP pool. See if dedicated IPs (managed) is right for how you send mail by reviewing the comparison table in *Dedicated IP addresses*.

Understanding the shared responsibility between you and SES when using dedicated IPs (managed)

While dedicated IP addresses (managed) offers numerous automated features for dedicated IP management, scaling, and warmup, the extent of this automation and SES' responsibilities needs to be understood. It would be incorrect to assume that "managed" means SES completely handles all aspects of IP reputation and listing issues. To clarify these misconceptions, we need to emphasize that while the service automates technical aspects like scaling and warmup, you remain responsible for maintaining your sending reputation and managing any reputation related issues such as getting listed in a Reputation Block List (RBL).

The following FAQs clarify the shared responsibility model between you and SES and address common misconceptions about the service's scope. These FAQS highlight that while the "managed" aspect refers to technical infrastructure management, you must still actively monitor and maintain your sending reputation, keep bounce rates low, and handle most RBL delisting requests yourself:

- the section called “Managed DIPS FAQs”

Creating a managed IP pool to enable dedicated IPs (managed)

To enable dedicated IPs (managed), you first create a managed IP pool. After you create a managed pool, the feature determines how many dedicated IPs you require based on your sending patterns and will dynamically scale them to your requirements.

To use your managed pool to send email, you must associate the managed pool with a configuration set, and then specify that configuration set when sending email. The configuration set can also be applied to a sending identity by using a default configuration set.

There are two ways you can create a managed IP pool:

- Create a new pool.
- Convert an existing pool from standard to managed.

In the following procedures, instructions are provided for either method.

To create or convert to a managed IP pool using the SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Depending on whether you want to create a new managed IP pool or convert a standard dedicated IP pool to a managed one, follow the respective instructions:

   Create new pool

   To create a new managed IP pool

   1. Do one of the following:

      a. If you *don't* have existing dedicated IPs in your account:

         - The **Dedicated IPs** onboarding page is displayed. In the **Dedicated IPs (managed) overview** panel, choose **Enable dedicated IPs**.

           The **Create IP Pool** page opens.

      b. If you have existing dedicated IPs in your account:

         i. Select the **Managed IP pools** tab on the **Dedicated IPs** page.

ii. In the **All Dedicated IP (managed) pools** panel, choose **Create Managed IP pool**.

The **Create IP Pool** page opens.

2. In the **Pool details** panel,

   a. Choose **Managed (auto managed)** in the **Scaling mode** field.

   b. Enter a name for your managed pool in the **IP pool name** field.

> **Note**
>
> - The IP pool name must be unique. It can't be a duplicate of a standard dedicated IP pool name in your account.
> - You can't have more than 50 dedicated IP pools per AWS Region in your account inclusive of both managed and standard IP pools.

3. (Optional) You can associate this managed IP pool with a configuration set by choosing one from the dropdown list in the **Configuration sets** field.

> **Note**
>
> - If you choose a configuration set that's already associated with an IP pool, it will become associated with this managed pool, and no longer be associated with the previous pool.
> - To add or remove associated configuration sets after this managed pool is created, edit the configuration set's **Sending IP pool** parameter in the **General details** panel.
> - If you haven't created any configuration sets yet, see *Configuration sets*.

4. (Optional) You can add one or more **Tags** to your IP pool by including a tag key and an optional value for the key.

   a. Choose **Add new tag** and enter the **Key**. You can also add an optional **Value** for the tag. You can add up to 50 tags, if you make a mistake, choose **Remove**.

   b. To add the tags, choose **Save changes**.

After you create the pool, you can add, remove, or edit tags by selecting the managed pool and choosing **Edit**.

5. Select **Create pool**.

> **Note**
>
> - After you create a managed IP pool, it can't be converted to a standard IP pool.
> - When using dedicated IPs (managed), you can't have more than 10,000 sending identities (domains and email addresses, in any combination) per AWS Region in your account.

Convert standard to managed

To convert a standard dedicated IP pool to managed

1. Select the **Standard IP pools** tab on the **Dedicated IPs** page.
2. In the **All Dedicated IP (standard) pools** panel, select the checkbox of the dedicated IP pool you want to convert from standard to managed.
3. Choose **Convert to managed pool**—read the **Convert to managed IP pool** dialogue to confirm that you understand the conditions of converting your standard dedicated IP pool to a managed one.

> **Note**
>
> Before converting your dedicated IP pool from standard to managed, note the following:
>
> 1. All of your current dedicated IPs (standard) will be moved to the managed pool.
> 2. If you're currently leasing too many dedicated IPs (standard) for your sending volume, then dedicated IPs (managed) will remove the redundant IPs.

3. If any of your dedicated IPs (standard) are part of an allow-list for other applications, you should not transfer them to the managed pool as they will be removed if they become redundant—*refer to point 2*.
4. You will no longer be charged per IP, but instead will be charged based on the volume you send through the managed pool. See Amazon SES pricing.

4. If you agree to the conditions as stated, choose **Confirm**—a banner appears, confirming that your standard dedicated IP pool has been converted to a managed pool.

> **Note**
>
> Any configurations sets or tags you had associated with the standard pool before conversion will now be associated with the managed pool providing a seamless transition for any email sending using the configuration set.

Event publishing can be used to track the managed pool's sending performance. For more information, see the section called “Monitor email sending using event publishing”.

Viewing managed IP pool sending and capacity in the Amazon SES console

For the managed IP pools you've created, the SES console provides an easy way for you to observe how they're being used for your email sending through the use of cards and time series graphs that show sending metrics and ISP utilization and capacity.

To view managed IP pool sending and capacity using the SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Select the **Managed IP pools** tab on the **Dedicated IPs** page.
4. Depending on whether you want to view sending and capacity metrics in the Amazon SES console or the Amazon CloudWatch console, follow the respective instructions:

Amazon SES console

To view sending and capacity metrics in the Amazon SES console

1. In the **All Dedicated IP (managed) pools** table, select the name of a managed IP pool listed in the **IP pool** column to view its details.

   The selected IP pool's detail page opens with the following cards and time series graphs:

   a. Cards:

   - *Sending status* – Indicates if your sending volume and frequency is enough to utilize dedicated IPs by displaying one of two statuses:
     - *Insufficient volume* – Your sending volume is too low.
     - *Sending via Dedicated IPs* – One or more dedicated IPs are being used in your managed pool.
   - *Managed dedicated IP send volume* – The volume of email sent through dedicated IPs in your managed pool in the last 7 days.
   - *Managed dedicated IP send percentage* – The percentage of email sent through dedicated IPs in your managed pool in the last 7 days.

   b. Graphs:

   - *Sent volume* – The volume of email sent in the last 7 days through managed dedicated IPs as compared to shared IPs.
   - *Percentage of sent volume* – The percentage of email sent in the last 7 days through managed dedicated IPs as compared to shared IPs.
   - *ISP capacity* – Displays how much email is being sent through dedicated IPs in your managed pool per the top 10 most widely used ISPs and their available capacity during your sending:
     - *Sends for ISP (red bars)* – The volume of email you sent in the last 24 hours through the selected ISP.
     - *Capacity for ISP (blue line)* – The selected ISP’s available capacity during the last 24 hours.

2. To filter on a specific ISP for the **ISP capacity** graph, choose the **ISP** list box and select an ISP—the graph will update with metrics for the selected ISP. (If you don't filter on an ISP, Gmail is displayed by default).

Amazon CloudWatch console

To view sending and capacity metrics in the Amazon CloudWatch console

- In the **All Dedicated IP (managed) pools** table, select the **See** *<pool_name>* **CloudWatch metrics** link in the **CloudWatch metrics** column to view its details.

  The selected IP pool's page opens in the CloudWatch console displaying the following metrics:

  - *Send* – The volume of email sent through both managed dedicated IPs and shared IPs.
  - *ApproximateDedicatedSendingPercentage* – Indicates the approximate percentage of traffic that has been delivered through a dedicated IP.
  - *SentLast24Hours* – The volume of email you sent in the last 24 hours through the selected ISP. (Labeled *Sends for ISP* in the SES console.)
  - *Available24HourSend* – The selected ISP's available capacity during the last 24 hours. (Labeled *Capacity for ISP* in the SES console.)

Deleting a managed IP pool and opting out of dedicated IPs (managed)

When you delete a managed IP pool, all of its allocated IP addresses are automatically relinquished. If you only have one managed IP pool and you delete it, or you delete your last remaining managed IP pool, you'll be opting out of the dedicated IPs (managed) feature and charges will cease immediately.

To delete a managed IP pool using the SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dedicated IPs**.
3. Select the **Managed IP pools** tab on the **Dedicated IPs** page.

4. In the **All Dedicated IP (managed) pools** table, select the radio button next to the **IP pool** name of the managed pool you want to remove and choose **Delete**.
5. In the pop-up modal, you'll have the opportunity to confirm your choice by selecting **Delete**, or **Cancel** to keep your managed pool.

> **Note**
>
> If you only have one managed pool or you're removing your last managed pool, the pop-up modal will remind you that by deleting your remaining managed pool, you'll be opting out of the dedicated IPs (managed) feature and will no longer be charged for it. You will be required to enter `Disable` in the confirmation field before you can choose **Delete**.

Using your own IP addresses to send email using Amazon SES

Amazon SES includes a feature called *Bring Your Own IP* (*BYOIP*), which makes it possible to use your own IP addresses to send email through Amazon SES. If you already use a range of IP addresses to send email, you can request that we make your IP range available for sending email through Amazon SES.

> **Note**
>
> BYOIP is only available for dedicated IP addresses that you configure manually—it can't be used with *Dedicated IPs (managed)*.

BYOIP is helpful, for example, when you've developed a positive IP reputation using an in-house email sending system, but you want to migrate to Amazon SES. By using BYOIP, you can start sending email through Amazon SES immediately, without having to re-establish the reputations of your IP addresses.

Requirements

To use BYOIP, your IP address range has to meet the following requirements:

- The address range has to be registered with your Regional internet registry (RIR), such as the American Registry for Internet Numbers (ARIN), Réseaux IP Européens Network Coordination

Centre (RIPE NCC), or Asia-Pacific Network Information Centre (APNIC). The address range has to be registered to a business or institutional entity and can't be registered to a person.

- You have to be able to provide proof that you own the address range by submitting a signed authorization message.
- The addresses in the IP address range have to have a clean history. We might investigate the reputation of the IP address range, and we reserve the right to reject an IP address range if it contains IP addresses that have poor reputations or are associated with malicious behavior.
- The IP address range cannot include IP address ranges that were brought into another AWS service for BYOIP, such as Amazon EC2.

Considerations

There are several factors that you should consider before you request the transfer of your IP ranges to Amazon SES:

- The most specific address range that you can specify is /24. In other words, if you transfer the IP range 203.0.113.0/24 to your Amazon SES account, then you can send from a total of 256 addresses, ranging from 203.0.113.0 to 203.0.113.255. You have to transfer the entire range—Amazon SES doesn't currently allow you to transfer individual IP addresses.
- If you use BYOIP for a specific range of IP addresses, you can only access that range from a single AWS Region.
- You can bring five address ranges per Region to your AWS account.
- If you use your own IP addresses, you can't use the addresses in the pool of shared Amazon SES IP addresses. If you need to use these shared IP addresses, you can use Amazon SES in a different AWS Region, or create a new AWS account.
- There is a monthly charge for each IP address that you use with BYOIP. For more information, see Amazon SES Pricing.

Using your own IP addresses with Amazon SES

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each BYOIP request carefully.

If you want to use your own IP range with Amazon SES, send the following information to ses-byoip-request@amazon.com:

- Your AWS account ID.
- The AWS Region that you want to use the IP range in, such as ap-south-1.
- A description of your use case.
- The IP range that you want to use with Amazon SES.
- The name of the internet registry that the range is registered with.

We'll respond to your request within 48 business hours. In our communications with you, we might request additional information, including documents that prove your ownership of the IP range.

Virtual Deliverability Manager for Amazon SES

Deliverability—ensuring your emails reach recipient inboxes instead of spam or junk folders—is a core element of a successful email strategy. *Virtual Deliverability Manager* is an Amazon SES feature that helps you enhance email deliverability by providing insights into your sending and delivery data, and giving advice on how to fix issues that are negatively affecting your delivery success rate and reputation.

Virtual Deliverability Manager offers two complementary sets of deliverability capabilities:

SES deliverability

SES deliverability provides insights and recommendations for email sent through Amazon SES—helping you monitor delivery rates, track engagement, identify reputation issues, and resolve configuration problems that affect inbox placement. It includes a dashboard for viewing deliverability data at the account, ISP, sending identity, configuration set, and tenant levels, and an advisor that flags infrastructure issues and recommends solutions. See the section called “SES deliverability”.

Global deliverability

Global deliverability extends your visibility beyond Amazon SES with analytics based on a representative sample of your sending—across every provider your domains send through, not just Amazon SES. It includes campaign analytics, inbox placement rates, inbox placement testing, and blocklist monitoring. See the section called “Global deliverability”.

Topics

- SES deliverability
- Global deliverability

SES deliverability

SES deliverability provides insights and recommendations for email sent through Amazon SES—helping you monitor delivery rates, track engagement, identify reputation issues, and resolve configuration problems that affect inbox placement.

SES deliverability is enabled at the account level when you enable Virtual Deliverability Manager through the onboarding wizard in the Amazon SES console.

Key features

SES deliverability includes the following features:

- **Dashboard** – Provides insights into your deliverability data focusing on account, ISP, sending identity, and configuration set levels. This helps you quickly see problematic areas and trends, and catch possible problems before they turn into larger deliverability issues like deferrals or blocks. See the section called “Dashboard”.
- **Advisor** – Provides recommendations to improve your email sending by flagging configuration issues that are negatively affecting your email deliverability and reputation. It recommends solutions to resolve specific issues in the infrastructure of your sending domain, IP space, and authentication records. See the section called “Advisor”.

Getting started

To start using SES deliverability, enable Virtual Deliverability Manager from the onboarding wizard in the Amazon SES console. See the section called “Getting started”.

Topics

- Getting started with Virtual Deliverability Manager
- Virtual Deliverability Manager dashboard
- Virtual Deliverability Manager advisor
- Virtual Deliverability Manager settings

Getting started with Virtual Deliverability Manager

To start using Virtual Deliverability Manager with your account, you must enable it using the onboarding wizard in the Amazon SES console, where you'll set up *engagement tracking* and *optimized shared delivery*. Virtual Deliverability Manager uses engagement tracking and optimized shared delivery to monitor your sending and to help you make improvements to your deliverability and reputation.

- **Engagement tracking** – The ability to monitor recipient engagement behavior through open and click events by using a tracking pixel within a wrapped link. When triggered, the tracking pixel provides a timestamp of when a message was opened, and indicates which links were clicked by the recipient. *Turning this on alters your URLs and links to include Amazon SES engagement tracking wrappers.*

- **Optimized shared delivery** – Automatically chooses the optimal IP to use when sending emails, improving end-point delivery of messages to the target email recipients.

While both engagement tracking and optimized shared delivery are turned on by default in the onboarding wizard, you have the option to turn them off. We highly recommend that you keep both features enabled to get the most out of Virtual Deliverability Manager.

Getting started with Virtual Deliverability Manager using the Amazon SES console

The following procedure shows you how to get started with Virtual Deliverability Manager using the Amazon SES console.

To get started with Virtual Deliverability Manager using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Virtual Deliverability Manager**.
3. Choose any of the **Get started with Virtual Deliverability Manager** buttons on the **Virtual Deliverability Manager overview** page.
4. On the **Select Engagement tracking** page, accept the default or choose **Turn off engagement tracking**, then choose **Next**.

   > **Note**
   >
   > Turning on engagement tracking alters your URLs and links to include Amazon SES engagement tracking wrappers.

5. On the **Select Optimized shared delivery** page, accept the default or choose **Turn off optimized shared delivery**, then choose **Next**.

   > **Important**
   >
   > Optimized shared delivery might result in preemptive delays to your emails being sent in an attempt to protect your sending reputation. If you have a critical workload that must be sent without delay, we recommend that you don't enable this setting. Instead,

use configuration sets for sending, and only enable optimized shared delivery for those configuration sets where you can afford delays.

6. Review your choices for engagement tracking and optimized shared delivery on the **Review and enable** page. Choose **Previous** if you want to go back and make changes; otherwise, choose **Enable Virtual Deliverability Manager**.

   The **Virtual Deliverability Manager settings** page opens. The **Subscription overview** panel indicates the status of Virtual Deliverability Manager and the **Additional settings** panel indicates the status of **Engagement tracking** and **Optimized shared delivery**.

Once you've enabled Virtual Deliverability Manager for your account, you can define custom settings for how a configuration set will use engagement tracking and optimized shared delivery by overriding how they've been defined in Virtual Deliverability Manager. This gives you the flexibility to tailor your email sending for specific email campaigns. For example, you can enable engagement tracking and optimized shared delivery for your marketing email and disable them for your transactional email. See Virtual Deliverability Manager options while creating or editing a configuration set.

You can also enable global deliverability from the Virtual Deliverability Manager Settings page to extend your visibility with analytics across all providers. See the section called "Getting started".

Getting started with Virtual Deliverability Manager using the AWS CLI

The following examples show you how to get started with Virtual Deliverability Manager using the AWS CLI.

To get started with Virtual Deliverability Manager using the AWS CLI

You can use the `PutAccountVdmAttributes` operation in the Amazon SES API v2 to get started with Virtual Deliverability Manager. You can call this operation from the AWS CLI, as shown in the following examples.

- Enable Virtual Deliverability Manager in your account:

```
aws --region us-east-1 sesv2 put-account-vdm-attributes --vdm-attributes
 VdmEnabled=ENABLED
```

- Enable both engagement tracking and optimized shared delivery using an input file:

```
aws --region us-east-1 sesv2 put-account-vdm-attributes --cli-input-json file://
attributes.json
```

The input file looks similar to this:

```
{
    "VdmAttributes": {
        "VdmEnabled": "ENABLED",
        "DashboardAttributes": {
            "EngagementMetrics": "ENABLED"
        },
        "GuardianAttributes": {
            "OptimizedSharedDelivery": "ENABLED"
        }
    }
}
```

Parameter values and related data types can be found by linking from the `VdmAttributes` data type in the Amazon SES API v2 reference.

> **Note**
>
> Turning on engagement tracking alters your URLs and links to include Amazon SES engagement tracking wrappers.

> **Important**
>
> Optimized shared delivery might result in preemptive delays to your emails being sent in an attempt to protect your sending reputation. If you have a critical workload that must be sent without delay, we recommend that you don't enable this setting. Instead, use configuration sets for sending, and only enable optimized shared delivery for those configuration sets where you can afford delays.

- To verify the outcome:

```
aws --region us-east-1 sesv2 get-account
```

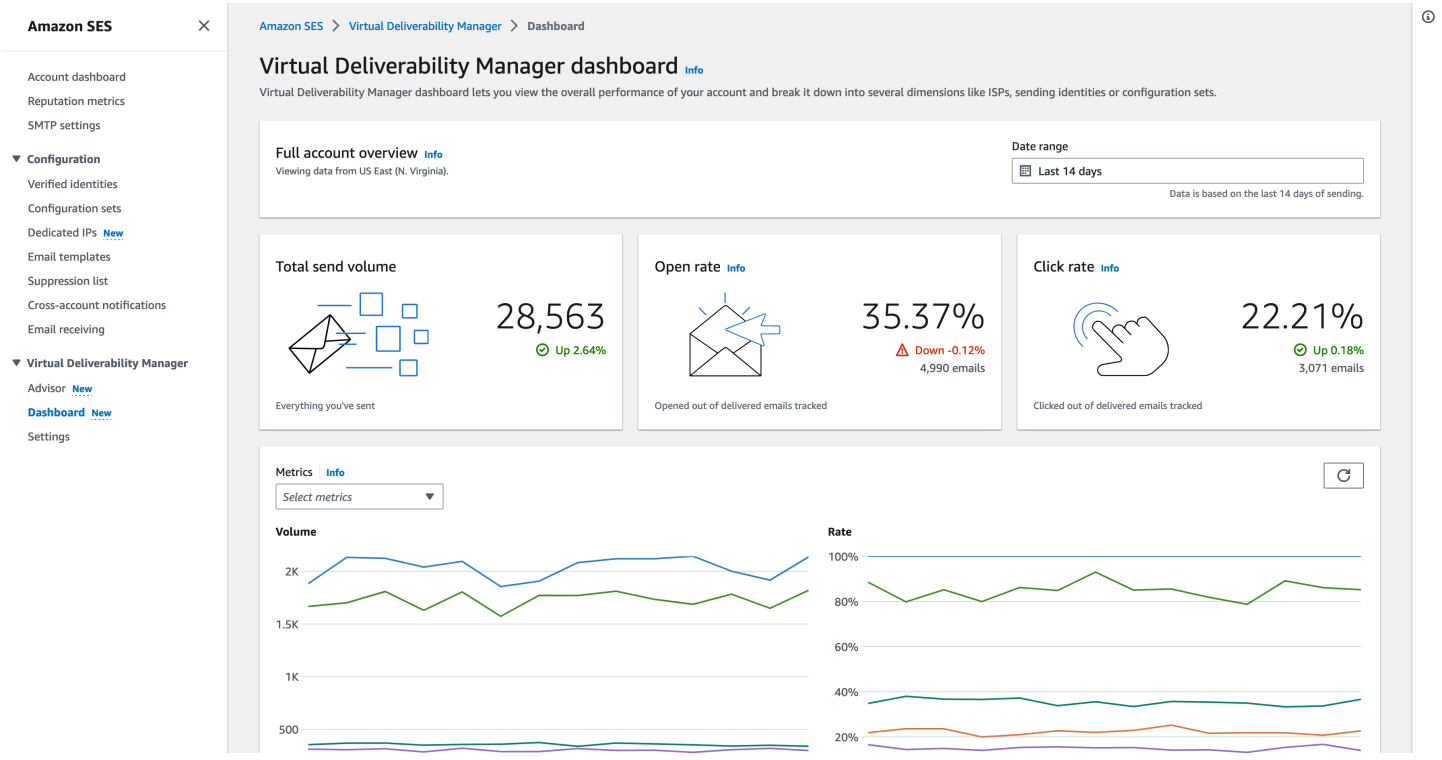
- To define custom settings for how a configuration set will use engagement tracking and optimized shared delivery by overriding how they've been defined in Virtual Deliverability Manager, see the AWS CLI example in the section called "Settings".

Virtual Deliverability Manager dashboard

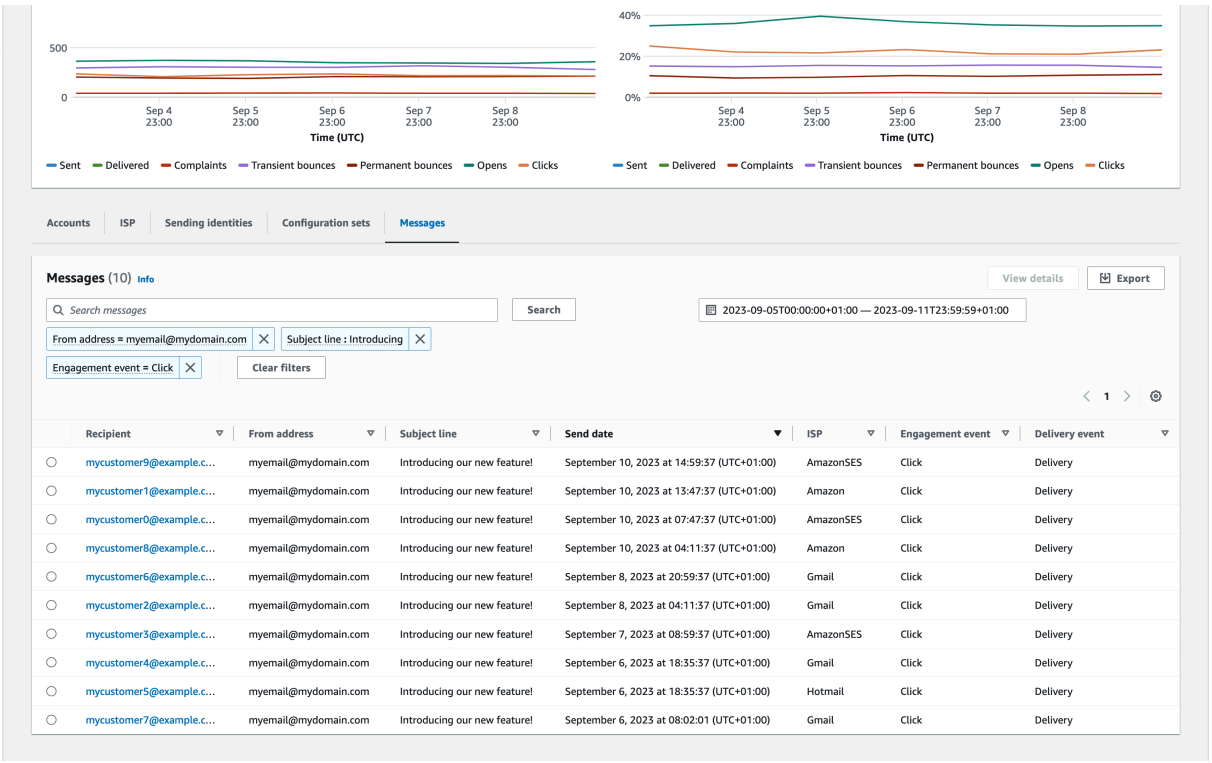
The dashboard offers high level views of your account's deliverability program, such as easy to read cards and time series graphs that show deliverability and reputation through open/click and delivery rates and bounce/complaint stats. The dashboard also offers a more detailed view, enabling you to drill down to more detailed specific table data when there's an issue that's tied to a particular ISP, sending identity, configuration set, or tenant that's associated with an email campaign.

Being able to see things from a high overall level with the ability to also view the specific details allows you to focus on the problematic areas of your deliverability rather than needing to review your email program as a whole. This level of insight also gives you the ability to catch trends and possible problems before they turn into larger deliverability problems, like deferrals or blocks.

An account overview in the Virtual Deliverability Manager dashboard showing the cards and time series graphs.



The *Messages* table selected in the Virtual Deliverability Manager dashboard showing sent messages matching the date range and filter criteria.

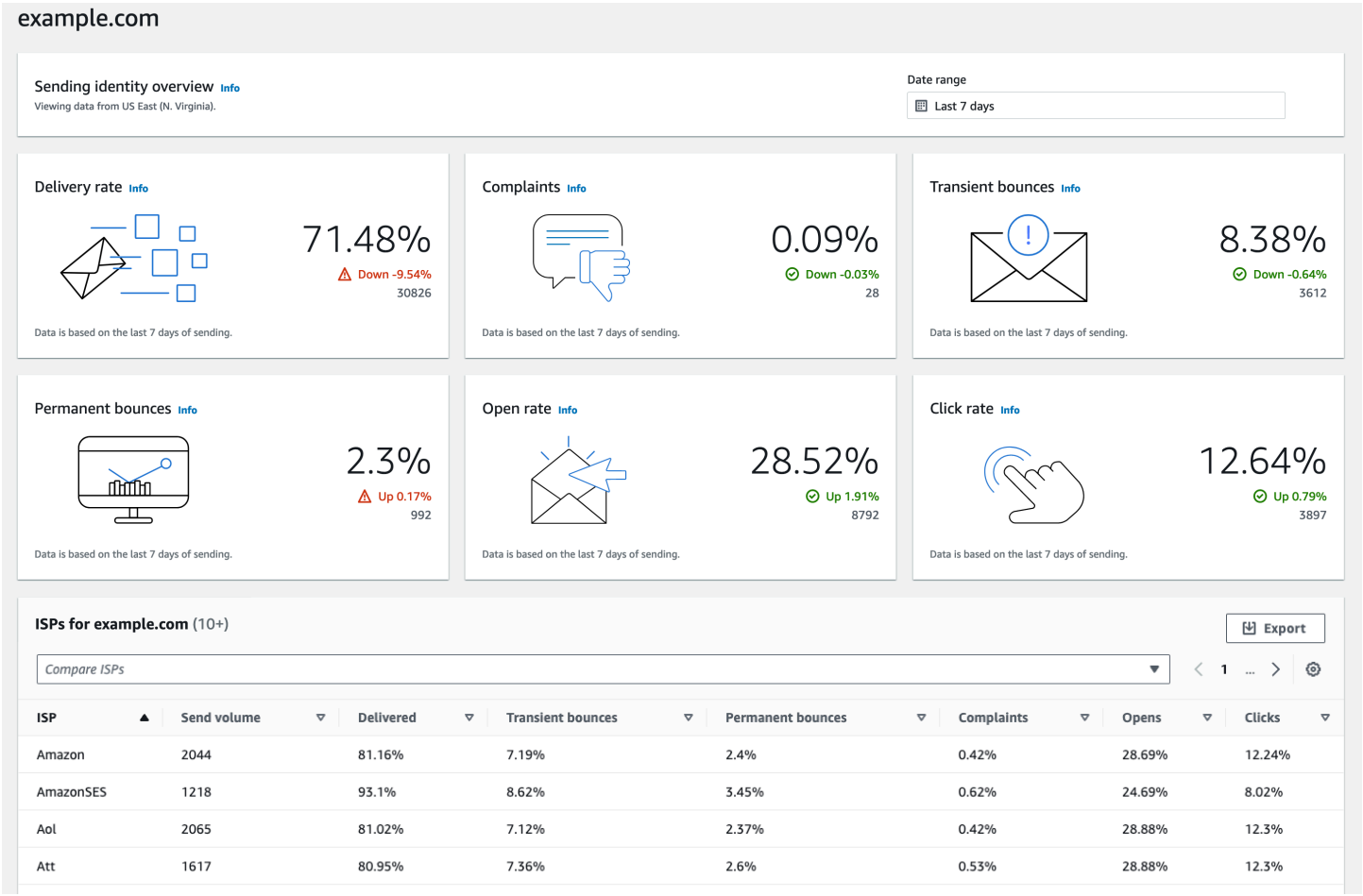


Granular data provided by the dashboard can help you to improve your sender reputation and calculate ideal times and dates for better engagement and conversions for your email program with the ability to drill-down to specific data sets:

- **ISP data** – Valuable when you have a deliverability issue to a specific ISP or mailbox provider—instead of trying to adjust your entire account, which may otherwise be doing well, you can focus on the problematic endpoint and align with its best practices to improve sender reputation to that ISP and restore good inbox deliverability to reach your recipients. It's also important to understand your ISP distribution—as you may send more heavily to one ISP or mailbox provider than to others. You need to ensure that traffic is always being delivered and engaged by the end recipients to have a positive impact your email conversion.
- **Sending identity & configuration set data** – Useful in helping you to identify sending identities and configuration sets that are contributing to your overall account deliverability issue. You can focus on those specifically, adjust your configurations, and possibly reduce sending with a particular identity until the issue is resolved. For example, a sending identity accidentally sent to a suppression list, resulting in all traffic going through that identity. That identity is associated with a configuration set, causing deliverability issues. It’s valuable in such cases to be able to identify the sending identity or configuration set so that you can focus on rectifying that

problem specifically, rather than combing through your entire account to try to identify the root cause of the deliverability issue.

Drill-down data displayed in the Virtual Deliverability Manager dashboard for the selected sending identity, *example.com*—cards display deliverability and reputation metrics. The table displays all of the ISPs that the sending identity sent mail to with metric rates for each ISP within the date range entered.



Using the Virtual Deliverability Manager dashboard in the Amazon SES console

The following procedure shows you how to use the Virtual Deliverability Manager dashboard in the Amazon SES console to view your overall deliverability and reputation statistics and to drill-down into problematic areas.

To use the Virtual Deliverability Manager dashboard to view high level and more detailed data of your account's deliverability metrics

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Dashboard** under **Virtual Deliverability Manager**.

> **Note**
>
> - **Dashboard** will not be visible if you haven't enabled Virtual Deliverability Manager for your account. For more information, see the section called "Getting started".
> - Dashboard metrics are displayed in near real-time.
> - Dashboard messages are displayed within a few minutes from send time.

3. In the **Full account overview** panel, choose a date range to be used for all metrics in the cards, time series graphs, and drill-down tables.

- In the **Date range** field, choose **Relative range** (default) or **Absolute range**.
  - **Relative range** – Select the radio button that corresponds with the number of days desired.
    - *Custom range* – Enter a range in either days (up to 60), weeks (up to 8), or months (up to 2).
  - **Absolute range** – The first date you choose will be the *Start date*, the second date will be the *End date*, not to exceed 60 days total. To specify a single day, choose it for both the *Start* and *End date*.

> **Note**
>
> The following applies to all date ranges in the dashboard:
>
> - All dates & times are UTC.
> - For **Relative range** dates, the last day ends on its UTC midnight timestamp. For example, if you choose *Last 7 days*, the seventh day would be yesterday, ending at midnight.

- If the date range is greater than 30 days, the *% Difference* column in the *Account statistics* table and the change percentages in the cards will not have a value (indicated by dash -).

4. The cards, time series graphs, and all of the drill-down tables, *Accounts statistics*, *ISP*, *Sending identities*, *Configuration sets*, and *Tenants*, display metric totals calculated from the date range entered, and use the metric math described in How dashboard metrics are calculated.

   - To create a local `.csv` file of the data you're currently viewing in either the *ISP*, *Sending identities*, or *Configuration sets* table, select its **Export** button.

5. Time series graphs charting **Volume** and **Rate** progression for the date range you entered are shown in the **Metrics** pane. Hovering over a date interval in the graphs will show the exact volume count or rate percentage based on a daily aggregation. You can filter the metrics you want to see using the *Select metrics* dropdown.

6. Choose the **Accounts** tab to display the **Accounts statistics** table.

   - This table gives an overview of your deliverability and reputation metrics, showing the total **Volume**, **% Rate**, and **% Difference** for *Sent*, *Delivered*, *Complaints*, *Transient & Permanent bounces*, *Opens & Clicks* as calculated from the date range entered.

   Note

   If the date range is greater than 30 days, the *% Difference* column will not have a value (indicated by dash -).

7. Choose the **ISP** tab to display the **ISP** table.

   - This table displays metrics for *Send volume*, *Delivered*, *Transient & Permanent bounces*, *Complaints*, *Opens & Clicks* for each ISP you've sent to as calculated from the date range entered.
   - To filter specific ISPs, inside the *Compare ISPs* search box, choose the corresponding check box for each ISP to include.
   - To create a local `.csv` file of the data you're currently viewing in this table, select its **Export** button.

8. Choose the **Sending identities** tab to display the **Sending identities** table.

- This table displays metrics for *Send volume*, *Delivered*, *Transient & Permanent bounces*, *Complaints*, *Opens & Clicks* for each sending identity you've used as calculated from the date range entered.
- To filter specific sending identities, inside the *Compare identities* search box, choose the corresponding check box for each identity to include.
- To drill-down on a specific sending identity, choose its name in the **Sending identity** column.
  - Cards will appear displaying *Delivery rate*, *Complaints*, *Transient & Permanent bounces*, *Open & Click rates* for the selected sending identity as calculated from the date range entered.
  - The time series graphs will refresh displaying all the metrics for the selected sending identity as calculated from the date range entered.
  - An ISP table will be displayed listing all the ISPs the sending identity sent mail to with metrics given for each ISP as calculated from the date range entered.
- To create a local `.csv` file of the data you're currently viewing in this table, select its **Export** button.

9. Choose the **Configuration sets** tab to display the **Configuration sets** table.

- This table displays metrics for *Send volume*, *Delivered*, *Transient & Permanent bounces*, *Complaints*, *Opens & Clicks* for each configuration set that's been used to send mail as calculated from the date range entered.
- To filter specific configuration sets, inside the *Compare configuration sets* search box, choose the corresponding check box for each configuration set to include.
- To drill down on a specific configuration set, choose its name in the **Configuration set** column.
  - Cards will appear displaying *Delivery rate*, *Complaints*, *Transient & Permanent bounces*, *Open & Click rates* for the selected configuration set as calculated from the date range entered.
  - The time series graphs will refresh displaying all the metrics for the selected configuration set as calculated from the date range entered.
  - An ISP table will be displayed listing all the ISPs the configuration set was used to send mail to with metrics given for each ISP as calculated from the date range entered.
- To create a local `.csv` file of the data you're currently viewing in this table, select its **Export** button.

10. Choose the **Tenants** tab to display the **Tenants** table.

> **Note**
>
> The **Tenants** tab is only applicable if you're managing your email sending through tenants. For more information, see *Tenants*.

- This table displays metrics for *Send volume*, *Delivered*, *Transient & Permanent bounces*, *Complaints*, *Opens & Clicks* for each of your tenants as calculated from the date range entered.
- To filter specific tenants, inside the *Compare tenants* search box, choose the corresponding check box for each tenant to include. You can also use the sending status filter to list tenants that share a sending status.
- To drill-down on a specific tenant, choose its name in the **Tenant** column.
  - Cards will appear displaying *Delivery rate*, *Complaints*, *Transient & Permanent bounces*, *Open & Click rates* for the selected tenant as calculated from the date range entered.
  - The time series graphs will refresh displaying all the metrics for the selected tenant as calculated from the date range entered.
  - An ISP table will be displayed listing all the ISPs the tenant sent mail to with metrics given for each ISP as calculated from the date range entered.
  - Tables will be displayed listing the sending identities and configuration sets associated with the tenant, and any reputation findings for the tenant.
- To create a local `.csv` file of the data you're currently viewing in this table, select its **Export** button.

11. Choose the **Messages** tab to display the **Messages** table.

    This is an interactive table that provides a way for you to search and find your sent messages. For each message, you can track its current delivery and engagement status, event history, and see the response returned by the mailbox provider. The following points cover the ways you can search for particular messages:

- Selecting inside the date range picker, you can filter on messages you've sent within the last 30 days. If you don't select a date range, your search will default to the last 7 days including the current day within your timezone.

- In the *Search messages* field you can filter on *Recipient*, *From address*, *Subject line*, *ISP*, *Tenant name*, *Engagement event*, *Delivery event*, and *Message ID* — the following properties apply:
  - Depending on the filter type, you either enter a case sensitive text string, or select a value from a list.
  - *Engagement event* is limited to a single value, *Subject line* can have up to two values, and all other filters can have up to five values per search. Filtering by *Message ID* will exclude any other filters you may have selected including the date range.
  - The *Message ID* column is hidden by default, but can be displayed by selecting the gear icon to customize how you view the **Messages** table.
- After you've selected your filters and date range, choose **Search** and the table will be populated with messages matching your search criteria. The table can load up to 100 messages. *If your search returns more than 100 messages, the 100 messages in the table are a random sample of the total returned.*
- Selecting a message's radio button followed by selecting **View details** will produce a **Message info** sidebar containing details of the message's full event history, the most recent at top, and any responses or diagnostic codes returned by the mailbox provider.
- To create a local `.csv` file of the data you're currently viewing in this table, select its **Export** button.

Accessing your Virtual Deliverability Manager metric data using the AWS CLI

The following example shows you how to access your Virtual Deliverability Manager metric data using the AWS CLI. This is the same data used in the Virtual Deliverability Manager dashboard in the console.

To access your deliverability metric data using the AWS CLI

You can use the `BatchGetMetricData` operation in the Amazon SES API v2 to access your deliverability metric data. You can call this operation from the AWS CLI as shown in the following examples.

- Access your deliverability metric data:

```
aws --region us-east-1 sesv2 batch-get-metric-data --cli-input-json file://sends.json
```

- The input file looks similar to this:

```
{
 "Queries": [
   {
     "Id": "Retrieve-Account-Sends",
     "Namespace": "VDM",
     "Metric": "SEND",
     "StartDate": "2022-11-04T00:00:00",
     "EndDate": "2022-11-05T00:00:00"
    }
 ]
}
```

More information about parameter values and related data types can be found by linking from the `BatchGetMetricDataQuery` data type in the Amazon SES API v2 reference.

Filtering and exporting your deliverability metric data using the AWS CLI

This example shows you how to use the `CreateExportJob` operation to filter and export your deliverability metric data to a .csv or .json file using the AWS CLI. This is the same data used in the Virtual Deliverability Manager dashboard's **ISP**, **Sending identities**, and **Configuration sets** tables.

To filter and export your deliverability metric data to a .csv or .json file using the AWS CLI

You can use the `CreateExportJob` operation along with the `MetricsDataSource` data type in the Amazon SES API v2 to filter and export your metric data to a .csv or .json file. You call this operation from the AWS CLI as shown in the following example.

> **Note**
>
> In `ExportDestination`, specify only `DataFormat`. Amazon SES writes the export file to a location that it manages. After the job completes, retrieve the file with `GetExportJob`. To store a copy in your own Amazon S3 bucket, upload the downloaded file to your bucket.

- Filter and export your deliverability metric data using an input file:

```
aws --region us-east-1 sesv2 create-export-job --cli-input-json file://metric-export-input.json
```

- In this example, the input file is using MetricsDataSource parameters to filter on all the ISPs you've sent mail to, showing the rate of successful delivery within the given date range, and a .csv format specified for the output file:

```
{
    "ExportDataSource": {
        "MetricsDataSource": {
            "Dimensions": {
                "ISP": ["*"]
            },
            "Namespace": "VDM",
            "Metrics": [
                {
                    "Name": "DELIVERY",
                    "Aggregation": "RATE"
                }
            ],
            "StartDate": "2023-06-13T00:00:00",
            "EndDate": "2023-06-20T00:00:00"
        }
    },
    "ExportDestination": {
        "DataFormat": "CSV"
    }
}
```

  More information about parameter values and related data types can be found in MetricsDataSource as an object of the type ExportDataSource in the Amazon SES API v2 reference.

Finding your sent messages, their delivery & engagement status, and exporting the results using the AWS CLI

These examples show you how to use the CreateExportJob operation to search and find particular messages you've sent, see their current delivery and engagement status, and export the results of your search to a .csv or .json file using the AWS CLI. This is the same data used in the Virtual Deliverability Manager dashboard's **Messages** table.

To find sent messages, their delivery and engagement status, and export the results to a .csv or .json file using the AWS CLI

You can use the `CreateExportJob` operation along with the `MessageInsightsDataSource` data type in the Amazon SES API v2 to apply filters in order to find particular messages you've sent, see their delivery and engagement status, and export the results to a .csv or .json file. You call this operation from the AWS CLI as shown in the following examples.

Note

If your filtered search returns more than 10,000 messages, the 10,000 messages in the API's result set are a random sample of the total returned.

Note

In `ExportDestination`, specify only `DataFormat`. Amazon SES writes the export file to a location that it manages. After the job completes, retrieve the file with `GetExportJob`. To store a copy in your own Amazon S3 bucket, upload the downloaded file to your bucket.

- Find sent messages, see their current status, and export results using an input file:

```
aws --region us-east-1 sesv2 create-export-job --cli-input-json file://message-
insights-export-input.json
```

- In this example, the input file is using `MessageInsightsDataSource` parameters to filter on a subject equal to "Sale Ends Tonight!", and a .csv format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "Subject": [
                    "Sale Ends Tonight!"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "CSV"
```

```
    }
}
```

- In this example, the input file is using MessageInsightsDataSource parameters to filter on a subject that starts with “Hello”, sent with a FromEmailAddress containing “information” to destinations ending with “@example.com”, and a .json format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "Subject": [
                    "Hello*"
                ],
                "FromEmailAddress": [
                    "*information*"
                ],
                "Destination": [
                    "*@example.com"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "JSON"
    }
}
```

- In this example, the input file is using MessageInsightsDataSource parameters to filter on a subject that starts with “Hello”, exclude results that have "noreply@example.com" as a FromEmailAddress, and a .csv format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
```

```
                "Subject": [
                    "Hello*"
                ]
            },
            "Exclude": {
                "FromEmailAddress": [
                    "noreply@example.com"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "CSV"
    }
}
```

- In this example, the input file is using `MessageInsightsDataSource` parameters to filter on a subject that starts with “Hello”, sent with a FromEmailAddress containing “information” to destinations ending with “@example.com”, using Gmail as the ISP, a last delivery event of “DELIVERY”, a last engagement event that’s either “OPEN” or “CLICK”, and a .json format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "Subject": [
                    "Hello*"
                ],
                "FromEmailAddress": [
                    "*information*"
                ],
                "Destination": [
                    "*@example.com"
                ],
                "Isp": [
                    "Gmail"
                ],
                "LastDeliveryEvent": [
```

```
                    "DELIVERY"
                ],
                "LastEngagementEvent": [
                    "OPEN", "CLICK"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "JSON"
    }
}
```

- In this example, the input file is using `MessageInsightsDataSource` parameters to filter on messages sent by the tenant named "example-tenant" that had a last delivery event of "DELIVERY", and a .csv format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "TenantName": [
                    "example-tenant"
                ],
                "LastDeliveryEvent": [
                    "DELIVERY"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "CSV"
    }
}
```

- In this example, the input file is using `MessageInsightsDataSource` parameters to filter on destinations ending with “@example1.com”, or “@example2.com”, or “@example3.com”, exclude

messages with a LastDeliveryEvent equal to “SEND” or “DELIVERY”, and a .csv format specified for the output file:

```
{
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "Destination": [
                    "*@example1.com",
                    "*@example2.com",
                    "*@example3.com"
                ]
            },
            "Exclude": {
                "LastDeliveryEvent": [
                    "SEND",
                    "DELIVERY"
                ]
            }
        }
    },
    "ExportDestination": {
        "DataFormat": "CSV"
    }
}
```

More information about parameter values and related data types can be found in `MessageInsightsDataSource` as an object of the type `ExportDataSource` in the Amazon SES API v2 reference.

Managing your export jobs using the AWS CLI

These examples show you how to manage your export jobs by listing them, getting information about them, and canceling them using the AWS CLI.

To list your export jobs using the AWS CLI

You can use the `ListExportJobs` operation in the Amazon SES API v2 to list your export jobs. You can call this operation from the AWS CLI as shown in the following examples.

- List your export jobs:

```
aws --region us-east-1 sesv2 list-export-jobs --export-source-type=METRICS_DATA
```

```
aws --region us-east-1 sesv2 list-export-jobs --job-status=CREATED
```

```
aws --region us-east-1 sesv2 list-export-jobs --cli-input-json file://list-export-jobs-input.json
```

- The input file looks similar to this:

```
{
  "NextToken": "",
  "PageSize": 0,
  "ExportSourceType": "METRICS_DATA",
  "JobStatus": "CREATED"
}
```

More information about parameter values for the `ListExportJobs` operation can be found in the Amazon SES API v2 reference.

To get information about your export job using the AWS CLI

You can use the `GetExportJob` operation in the Amazon SES API v2 to get information about your export job. You can call this operation from the AWS CLI as shown in the following examples.

- Get information about your export job:

```
aws --region us-east-1 sesv2 get-export-job --job-id=<JobId>
```

```
aws --region us-east-1 sesv2 get-export-job --cli-input-json file://get-export-job-input.json
```

- The input file looks similar to this:

```
{
    "JobId": "e2220d6b-dce5-45f2-bf60-3287a465b732"
}
```

More information about parameter values for the `GetExportJob` operation can be found in the Amazon SES API v2 reference.

- Export jobs run asynchronously. Call `get-export-job` until `JobStatus` is `COMPLETED`, `FAILED`, or `CANCELLED`. When the status is `COMPLETED`, the response includes a pre-signed URL in `ExportDestination.S3Url` that you use to download the export file. When the status is `FAILED`, see `FailureInfo`. The URL expires five minutes after `get-export-job` returns it. Call `get-export-job` again to get a new URL. A completed job's response looks similar to the following example:

```
{
    "JobId": "e2220d6b-dce5-45f2-bf60-3287a465b732",
    "ExportSourceType": "MESSAGE_INSIGHTS",
    "JobStatus": "COMPLETED",
    "ExportDestination": {
        "DataFormat": "CSV",
        "S3Url": "https://presigned-url"
    },
    "ExportDataSource": {
        "MessageInsightsDataSource": {
            "StartDate": "2023-07-01T00:00:00",
            "EndDate": "2023-07-10T00:00:00",
            "Include": {
                "Subject": [
                    "Sale Ends Tonight!"
                ]
            }
        }
    },
    "CreatedTimestamp": "2023-07-10T10:00:00",
    "CompletedTimestamp": "2023-07-10T10:05:00",
    "Statistics": {
        "ProcessedRecordsCount": 5,
        "ExportedRecordsCount": 5
    }
}
```

To upload the export file to your own Amazon S3 bucket, download it locally first and then use `aws s3 cp`:

```
set -euo pipefail
```

```
S3_URL=$(aws --region us-east-1 sesv2 get-export-job \
    --job-id=<JobId> \
    --query 'ExportDestination.S3Url' \
    --output text)

if [ -z "$S3_URL" ] || [ "$S3_URL" = "None" ]; then
    echo "Export job is not COMPLETED yet; no download URL is available." >&2
    exit 1
fi

curl --fail --location --output export-file "$S3_URL"
aws s3 cp export-file s3://amzn-s3-demo-bucket/key
```

You must have `s3:PutObject` permission for the destination bucket.

To cancel your export job using the AWS CLI

You can use the `CancelExportJob` operation in the Amazon SES API v2 to cancel your export job. You can call this operation from the AWS CLI as shown in the following examples.

- Cancel your export job:

```
aws --region us-east-1 sesv2 cancel-export-job --job-id=<JobId>
```

```
aws --region us-east-1 sesv2 cancel-export-job --cli-input-json file://cancel-export-job-input.json
```

- The input file looks similar to this:

```
{
    "JobId": "e2220d6b-dce5-45f2-bf60-3287a465b732"
}
```

More information about parameter values for the `CancelExportJob` operation can be found in the Amazon SES API v2 reference.

Seeing a message's full event history and ISP responses using the AWS CLI

The following example shows you how to see details of a message's full event history and any responses or diagnostic codes returned by the mailbox provider using the AWS CLI. This is the same data used in the **Message info** sidebar after selecting a message's radio button in the Virtual Deliverability Manager dashboard's **Messages** table.

To see a message's event history and ISP responses using the AWS CLI

You can use the `GetMessageInsights` operation in the Amazon SES API v2 to see details of a sent message. You can call this operation from the AWS CLI as shown in the following example.

- See message details about a sent email identified by its message-id:

```
aws --region us-east-1 sesv2 get-message-insights --message-id
 01000100001000dd-2a19190d-99d4-0000-9f00-deb5bbf2bfbe-000001
```

  More information about parameter values for the `GetMessageInsights` operation can be found in the Amazon SES API v2 reference.

How Virtual Deliverability Manager dashboard metrics are calculated

All of the rate cards and drill-down tables displayed in the Virtual Deliverability Manager dashboard calculate metrics for the date range entered in the *Full account overview* panel.

The metric rate percentages displayed in the dashboard are calculated as described in the table. The last four columns represent qualifiers to the basic math that's used to derive the displayed metrics. For example, your *Open rate* is calculated as the open total divided by the delivered total for HTML messages that are delivered with engagement tracking turned on. They don't reflect any of the messages that you sent without engagement tracking and are not HTML encoded.

Rate %	How it's calculated	With engagement tracking enabled & HTML	And with at least 1 tracked link	Delivered to ISPs with an SES FBL	Excluded if on your account-level suppression list
Open rate	*open total / delivered total*	✓			
Click rate	*click total / delivered total*	✓	✓		
Complaint rate	*complaint total / delivered total*			✓	✓
Delivery rate	*delivered total / sent total*				
Transient bounce rat	*transient bounce total / sent tot*				✓
Permanent bounce ra	*permanent bounce total / sent to*				✓
Total send volume	*Rate % not displayed (everything you've sent; always 100%)*				

How the difference rate and volume totals are calculated for all metrics:

- **Difference %** – Difference in metric total as compared to previous metric total for the given date range. For example, if *Last 7 days* is the specified date range, *Metric rate of last 7 days - Metric rate of previous 7 days*.
  - The difference % for *Total send volume* is calculated differently. For example, *(Send volume of last 7 days - Send volume of previous 7 days) / Send volume of previous 7 days*.
- **Volume** – Total count of each metric.

Note

- The *Delivered* column in the drill-down tables displays the straight delivered volume without the delivered qualifiers used for calculating open, click, and complaint rates.

- Virtual Deliverability Manager only tracks metrics from emails that have one recipient —emails with multiple recipients are not counted in any of the Virtual Deliverability Manager dashboard metrics.
  - In these cases, your Virtual Deliverability Manager metric counts will be lower than your Amazon CloudWatch metric counts because CloudWatch metrics include emails with multiple recipients.
- Emails sent to the *SES mailbox simulator* are not counted in any of the Virtual Deliverability Manager dashboard metrics.
- Emails sent through a delegate sender's account (formerly cross-account sending) are not counted in any of the Virtual Deliverability Manager dashboard metrics.

Important

Apple Mail's Privacy Protection and its impact to engagement rates: As a result of Apple implementing their Mail Privacy Protection (MPP) feature for Apple devices as of iOS15, engagement numbers have become inflated as MPP triggers opens as the Apple Mail app is initiated, not necessarily when a recipient opens and/or clicks a message. This causes engagement data to look much higher than it typically would be and this is something email marketers will have to take into account when reviewing engagement. There are several other ways of identifying engagement, such as web activity, app/portal usage and also using proxy data from non-Apple devices to build an aggregate metric. The important thing to focus on is the trends of engagement as that can indicate if there's a problem with your email sending. For more information, see Apple Mail's Privacy Protection.

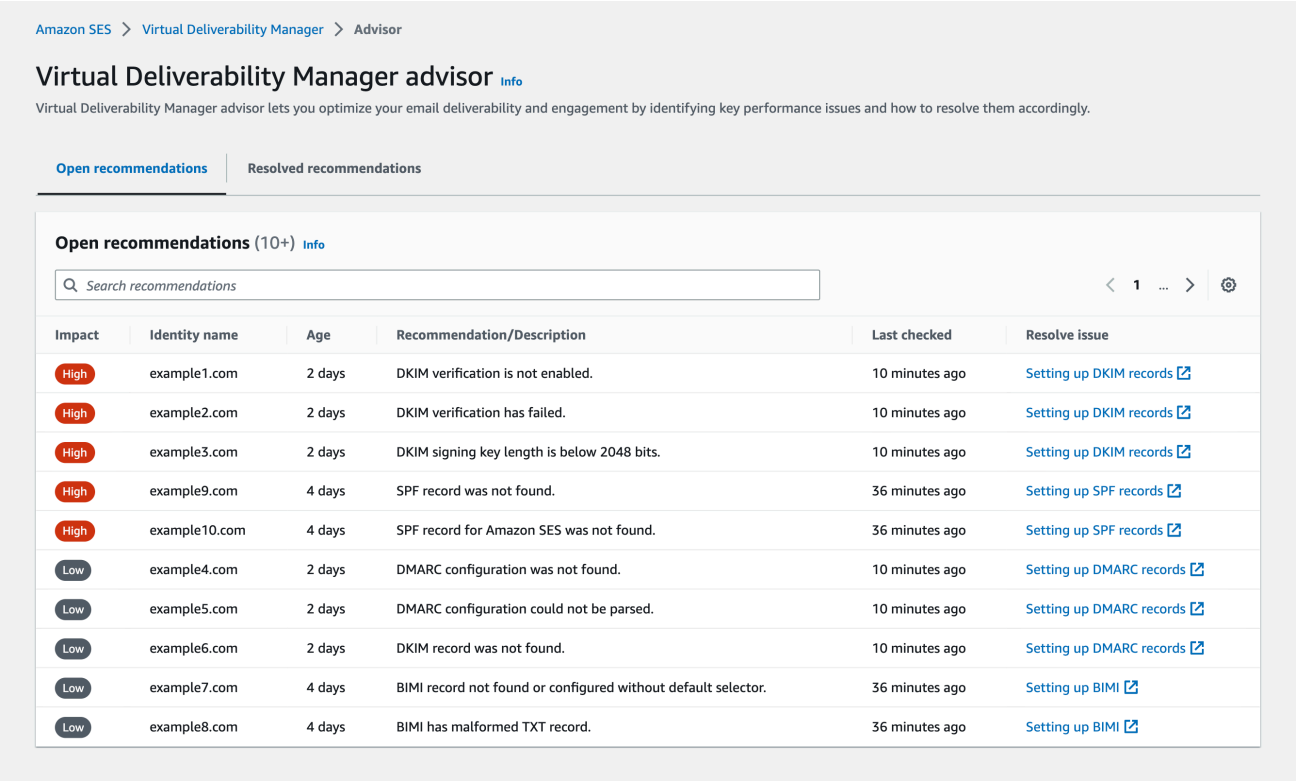
Virtual Deliverability Manager advisor

The *Virtual Deliverability Manager advisor* helps to optimize your email deliverability and engagement by identifying key performance and infrastructure issues at the account and sending identity levels that are adversely affecting your email deliverability and reputation. It provides solutions by providing specific guidance on how to resolve the identified issue.

Advisor's infrastructure recommendations are listed in the *Open recommendations* table. The recommendations identify standard email authentication problems, such as when SPF, DKIM, DMARC, or BIMI records don't exist or have problems with their configuration such as being

malformed or having a key length that's too short. They're categorized by severity of *Impact*, *Identity name* of the sending domain, and the *Age* of the alert. In the search bar, a list box provides the option to filter on impact level, infrastructure category, or sending identity name. The *Last checked* column shows a relative time of when the recommendation was last updated, such as "Just now" or "15 minutes ago". The last column, *Resolve issue*, provides a link to the relevant section in the Amazon SES Developer Guide with guidance about how to resolve the identified issue.

Open recommendations display in the Virtual Deliverability Manager advisor sorted by impact level.



If you don't have any ongoing advisor notifications, a message will indicate that you don’t have any open recommendations. We recommend that you check the advisor on a regular basis. Optionally, you can integrate these advisor notification events with Amazon EventBridge to build scalable event-driven applications as explained in *Monitoring using EventBridge*.

You can also access the *Resolved recommendations* table from the Virtual Deliverability Manager advisor page, which lists infrastructure issues that you’ve resolved by implementing the advisor's guidance. Resolved recommendations are listed with an initial status that describes the issue before it was resolved. Resolved recommendations expire after 30 days.

If you have global deliverability enabled, the advisor also generates recommendations when your monitored domains or IPs are listed on a blocklist. These recommendations include the

blocklist name, listing reason, and delisting guidance. For more information, see the section called "Blocklists".

What the Virtual Deliverability Manager advisor's looking for

In the previous section we discussed that Virtual Deliverability Manager's advisor performs checks against your sending domain to determine if you've configured a safely authenticated infrastructure to ensure you maintain a high rate of email deliverability and maintain a good sender reputation. Before you activate the Virtual Deliverability Manager advisor, we think it would be helpful for you to know exactly what the advisor's checking and what it's looking for in those checks.

You can use this table as a reference to go through your sending domain's configuration and correct any of these elements that are not aligned to the standards listed in this table before they become problems that the advisor has to alert you to.

Type of check	Advisor message	Why the advisor's alert	Learn more
Complaint rate check	*`ISP_name` ISP has `high/med/low` complaint rate.*	Identity has exceeded complaint recommendations threshold for this ISP.	Monitoring sender reputation
DKIM configuration	*DKIM verification is not enabled.*	DKIM is not enabled per identity.	Easy DKIM in SES
DKIM key strength	*DKIM signing key length is below 2048 bits.*	DKIM signing key length is not using at least 2048 bits.	Easy DKIM in SES
DKIM DNS record validation	*DKIM verification has failed.*	DKIM CNAME records determined invalid after looking up and trying to validate the key.	Verifying a DKIM domain identity with your DNS provider
DMARC configuration	*DMARC configuration was not found.*	DMARC TXT records are missing.	Setting up the DMARC policy on your domain

Type of check	Advisor message	Why the advisor's alert	Learn more
DMARC DNS record format check	*DMARC configuration could not be parsed.*	Invalid format found for DMARC TXT records.	Setting up the DMARC policy on your domain
DMARC's DKIM configuration	*DKIM record was not found.*	No DKIM record was found in order to comply with DMARC.	Complying with DMARC through DKIM
DMARC's DKIM configuration	*DKIM record is not aligned.*	The domain specified in the DKIM signature does not align (match) with the domain in the From address.	Complying with DMARC through DKIM
SPF configuration	*SPF record was not found.*	SPF TXT record missing for Custom MAIL FROM domain.	Configuring your custom MAIL FROM domain
SPF "include" configured	*SPF record for Amazon SES was not found.*	`include:amazonses.com` is missing from SPF TXT record.	Configuring your custom MAIL FROM domain
SPF enforcement configured	*SPF all qualifier is missing.*	`~all` is missing from SPF TXT record.	Configuring your custom MAIL FROM domain
SPF enforcement validation	*An SPF configuration issue was found.*	Attempts to detect the required SPF MX record within 72 hours failed.	Custom MAIL FROM domain setup states
BIMI configured	*BIMI record not found or configured without default selector.*	BIMI TXT records are missing or lack the selector attribute.	Setting up BIMI

Type of check	Advisor message	Why the advisor's alert	Learn more
BIMI format validation	*BIMI has malformed TXT record.*	BIMI TXT record determined as misconfigured after checking for the presence and valid format of: version, certificate URL, and logo URL.	Setting up BIMI

Using the Virtual Deliverability Manager advisor in the Amazon SES console

The following procedure shows you how to use the Virtual Deliverability Manager advisor in the Amazon SES console to resolve identified deliverability issues using the Amazon SES console.

To use the Virtual Deliverability Manager advisor to resolve deliverability and reputation issues

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Advisor** under **Virtual Deliverability Manager**.

   > **Note**
   >
   > **Advisor** will not be visible if you haven't enabled Virtual Deliverability Manager for your account. For more information, see the section called “Getting started”.

3. The **Open recommendations** table displays by default. Recommendations are categorized by **Impact** (High/Low), **Identity name** (sending domain), **Age** (of the alert), and **Recommendation/Description** (identified issue). In the search bar, filter on the **Impact** level, the infrastructure issue **Category**, or the **Identity name** of the sending domain.
4. To remediate a problem that's described in the **Recommendation/Description** column, choose the link in the **Resolve issue** column for that row, and implement the suggested solution.

> **Note**
>
> After you implement a solution, the resolved issue can take up to six hours to be reflected. You can view the resolved issue on the **Resolved recommendations** tab.

Accessing your Virtual Deliverability Manager recommendations using the AWS CLI

The following examples show you how to access your Virtual Deliverability Manager recommendations using the AWS CLI.

To access your Virtual Deliverability Manager recommendations using the AWS CLI

You can use the `ListRecommendations` operation in the Amazon SES API v2 to list your deliverability recommendations. You can call this operation from the AWS CLI, as shown in the following examples.

- List the recommendations to see deliverability issues:

```
aws --region us-east-1 sesv2 list-recommendations
```

- Apply filters to retrieve recommendations for a specific domain that you own:

```
aws --region us-east-1 sesv2 list-recommendations --cli-input-json file://list-
recommendations.json
```

- The input file looks similar to this:

```
{
  "PageSize":100,
  "Filter":{
    "RESOURCE_ARN": "arn:aws:ses:us-east-1:123456789012:identity/example.com"
   }
}
```

Virtual Deliverability Manager settings

The Virtual Deliverability Manager Settings page provides controls for both SES deliverability and global deliverability. You can view or change settings at any time using the Amazon SES console or the AWS CLI.

SES deliverability settings

SES deliverability settings control engagement tracking and optimized shared delivery at the account level. You can also define custom settings at the configuration set level to override account-level defaults.

Changing SES deliverability settings using the Amazon SES console

To change SES deliverability settings using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Settings** under **Virtual Deliverability Manager**.
3. To change **Engagement tracking** or **Optimized shared delivery** settings:
   a. In the **Additional settings** panel, choose **Edit**.
   b. Select the corresponding radio button to turn either feature on or off, and then choose **Submit settings**.

   > **Note**
   >
   > *Engagement tracking* options that you define here or in Virtual Deliverability Manager's configuration set overrides, control whether or not to report opens and clicks in the Virtual Deliverability Manager dashboard; they do not affect event destination configurations that publish open and click events. For example, if you have engagement tracking disabled here, it will not disable the open and click event publishing you have set up in SES event destinations.

4. (Optional) To define custom settings for how a configuration set uses engagement tracking and optimized shared delivery by overriding how they're defined in Virtual Deliverability Manager, reference Virtual Deliverability Manager options while creating or editing a configuration set.
5. To disable Virtual Deliverability Manager:

a. In the **Subscription overview** panel, choose **Disable Virtual Deliverability Manager**.

b. In the **Disable Virtual Deliverability Manager?** pop-up window, enter *Disable* in the confirmation field, and then choose **Disable Virtual Deliverability Manager**.

6. To reenable Virtual Deliverability Manager, see the section called “Getting started”.

Changing SES deliverability settings using the AWS CLI

You can use the `PutAccountVdmAttributes` and `PutConfigurationSetVdmOptions` operations in the Amazon SES API v2 to change your SES deliverability settings.

- Enable or disable engagement tracking, optimized shared delivery, or both using an input file:

```
aws --region us-east-1 sesv2 put-account-vdm-attributes --cli-input-json file://
attributes.json
```

In this example, where engagement tracking is `ENABLED` and optimized shared delivery is `DISABLED`, the input file looks similar to this:

```
{
    "VdmAttributes": {
        "VdmEnabled": "ENABLED",
        "DashboardAttributes": {
            "EngagementMetrics": "ENABLED"
        },
        "GuardianAttributes": {
            "OptimizedSharedDelivery": "DISABLED"
        }
    }
}
```

You can find more information about parameter values and related data types by linking from the `VdmAttributes` data type in the Amazon SES API v2 reference.

- Define custom settings for how a configuration set will use engagement tracking and optimized shared delivery by overriding how they've been defined in Virtual Deliverability Manager:

```
aws --region us-east-1 sesv2 put-configuration-set-vdm-options --cli-input-json
 file://config-set.json
```

In this example, where a configuration set named *example* has both engagement tracking and optimized shared delivery enabled, the input file looks similar to this:

```
{
    "ConfigurationSetName": "example",
    "VdmOptions": {
        "DashboardOptions": {
            "EngagementMetrics": "ENABLED"
        },
        "GuardianOptions": {
            "OptimizedSharedDelivery": "ENABLED"
        }
    }
}
```

For more information about parameter values and related data types, see the `VdmOptions` data type in the Amazon SES API v2 reference.

- To verify the outcome:

```
aws --region us-east-1 sesv2 get-configuration-set --configuration-set-name example
```

- Not specifying `DashboardOptions` or `GuardianOptions` options at the configuration set level results in your Virtual Deliverability Manager account-level settings applying to traffic sent through that configuration set.

Global deliverability settings

Global deliverability settings control domain monitoring, IP monitoring, and inbox placement testing. Global deliverability is enabled per region from the Virtual Deliverability Manager Settings page.

To manage global deliverability settings using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Settings** under **Virtual Deliverability Manager**.
3. In the **Global deliverability** panel, you can:

- Enable or disable global deliverability for the current region.
- Choose **Edit domains** to select which of your verified sending domains to monitor.

For detailed instructions on enabling or disabling global deliverability, see the section called "Getting started". For more information about global deliverability features, see the section called "Global deliverability".

Global deliverability

Global deliverability extends your visibility beyond Amazon SES. It provides analytics based on a representative sample of your sending—across every provider your domains send through, not just Amazon SES.

You can validate campaigns before they ship, track inbox placement at the ISP level, and monitor your IPs and domains against major blocklist operators—all from the Amazon SES console.

Global deliverability is enabled per region. In any region, domain monitoring tracks metrics across regions, accounts, and sending providers. IP monitoring only tracks the dedicated IPs available in that region. Inbox placement tests are not dependent on any specific region.

Key features

Global deliverability includes the following features:

- **Campaign analytics** – Surfaces per-campaign deliverability and engagement metrics based on a representative sample of emails sent from your monitored domains, even when they leave through a non-Amazon SES provider. See the section called "Campaign analytics".
- **Inbox placement rates** – Provides ISP-level inbox placement rates for your monitored domains, updated hourly based on analytics. See the section called "Inbox placement rates".
- **Inbox placement tests** – Sends your campaign to seed accounts across major mailbox providers before you send to real subscribers, so you can catch placement and authentication issues in advance. See the section called "Inbox placement tests".
- **Blocklists** – Monitors your dedicated IPs and sending domains against the industry's major blocklist operators with hourly checks and proactive alerts through Virtual Deliverability Manager Advisor when anything is listed (you can monitor these through EventBridge). See the section called "Blocklists".

Pricing

Global deliverability is sold as a single subscription package that includes monitored domains, dedicated IP monitoring, and inbox placement tests. For pricing details, see Amazon SES Pricing.

Getting started

To start using global deliverability, enable it from the Virtual Deliverability Manager Settings page in the Amazon SES console. See the section called “Getting started”.

Topics

- Getting started with global deliverability
- Campaign analytics
- Inbox placement rates
- Inbox placement tests
- Blocklists

Getting started with global deliverability

To start using global deliverability, you enable it from the Virtual Deliverability Manager Settings page in the Amazon SES console. After enabling, you select which of your verified sending domains to monitor.

Enabling global deliverability using the Amazon SES console

To enable global deliverability using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Settings** under **Virtual Deliverability Manager**.
3. In the **Global deliverability** panel, choose **Enable global deliverability**.
4. In the confirmation dialog, review the subscription details and choose **Enable global deliverability**.

   The subscription includes monitored domains, dedicated IP monitoring, and inbox placement tests. For pricing details, see Amazon SES Pricing.

5. After enabling, choose **Edit domains** to select which of your verified sending domains to monitor.

Disabling global deliverability

To disable global deliverability using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Settings** under **Virtual Deliverability Manager**.
3. In the **Global deliverability** panel, choose **Disable global deliverability**.
4. In the confirmation dialog, enter *disable* in the confirmation field, and then choose **Disable global deliverability**.

> **Important**
>
> Disabling global deliverability removes access to campaign analytics, inbox placement testing, and reputation monitoring.

Campaign analytics

Campaign analytics surfaces per-campaign deliverability and engagement metrics based on a representative sample of emails sent from your monitored domains, across every provider your domains send through—not just Amazon SES. Use campaign analytics to pinpoint which campaigns are performing well and where to make improvements.

What campaign analytics provides

For each detected campaign, campaign analytics displays the following information:

- *Subject line* – The subject line of the campaign.
- *From address* – The sender address used for the campaign.
- *Last send date* – The most recent date the campaign was sent.
- *Messages sent* – The number of messages sent in the campaign.
- *Inbox rate* – The percentage of messages that reached the inbox.

- *Spam rate* – The percentage of messages that were delivered to spam.

Viewing campaign details

When you select a campaign from the campaigns list, you can view detailed information including:

- *Campaign details* – Campaign ID, from address, sender address, messages sent, projected volume, first seen date, and last seen date.
- *Engagement metrics* – Inbox rate, spam rate, open rate, and delete rate for the campaign.
- *ISP breakdown* – Per-ISP metrics showing inbox rate, spam rate, open rate, and delete rate for each major mailbox provider (Gmail, Yahoo, Outlook, and others).

To view campaign analytics using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Campaign analytics** under **Global deliverability** in the **Virtual Deliverability Manager** section.
3. The **Campaigns** table displays all detected campaigns for your monitored domains.
4. Choose a campaign subject to view its detailed metrics and ISP breakdown.

Inbox placement rates

Inbox placement rates provides ISP-level inbox placement data for your monitored domains. This data is powered by analytics and updated hourly. These metrics are based on a representative sample of data and do not reflect the full volume of emails sent.

What inbox placement rates provides

For each monitored domain, inbox placement rates displays the following information:

- *ISP-level breakdown* – Inbox rate, spam rate, inbox count, and spam count for each major mailbox provider (Gmail, Outlook, Yahoo, and others).
- *Aggregate metrics* – Overall inbox rate and spam rate across all ISPs for the selected domain.
- *Trend data* – A time-series chart showing inbox placement rates by ISP over the selected date range.

Viewing inbox placement rates using the Amazon SES console

To view inbox placement rates using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Inbox placement rates** under **Global deliverability** in the **Virtual Deliverability Manager** section.
3. Select a monitored domain from the domain selector.
4. (Optional) Adjust the date range using the date range picker. You can select relative ranges (7, 14, 30, or 90 days) or specify an absolute date range.
5. Review the ISP breakdown table and the trend chart to identify any ISPs where inbox placement is lower than expected.

> **Note**
>
> Inbox placement data is provided by analytics and updated hourly. These metrics are based on a sample of data and do not reflect the full volume of emails sent.

Inbox placement tests

Inbox placement testing sends your campaign to seed accounts across major mailbox providers before you send to real subscribers, so you can catch placement and authentication issues before they reach your audience. Results are typically available in 2–4 hours.

What inbox placement tests provide

Each completed test provides the following information:

- *Overall results* – Inbox percentage, spam percentage, and missing percentage across all tested ISPs.
- *ISP breakdown* – Per-ISP inbox, spam, and missing percentages for Gmail, Outlook, Yahoo, and other major providers.
- *Test details* – Report ID, subject line, from address, creation date, and test status.

Your global deliverability subscription includes inbox placement tests each month. For pricing details including included quantities and overage charges, see Amazon SES Pricing.

Creating an inbox placement test

To create an inbox placement test using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Inbox placement tests** under **Global deliverability** in the **Virtual Deliverability Manager** section.
3. Choose **Create test**.
4. Select the identity to send from. You can choose either a verified email address or specify a local part with a verified domain.
5. Enter the **Subject** line for the test email.
6. Enter the email **Content** (HTML or plain text) that you want to test.
7. Choose **Create test**.

   The test appears in the tests list with a status of *In progress*. Results are typically available in 2–4 hours.

Viewing test results

To view inbox placement test results using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Inbox placement tests** under **Global deliverability** in the **Virtual Deliverability Manager** section.
3. Choose the test name from the tests list to view its details.
4. Review the **Test details** section for overall results and the **ISP breakdown** table for per-provider placement data.

Blocklists

Blocklist monitoring watches your dedicated IPs and sending domains against the industry's major blocklist operators, with hourly checks and proactive alerts through Virtual Deliverability Manager Advisor when anything is listed.

Monitored blocklist operators

Global deliverability monitors the following blocklist operators:

- *IP blocklists* – Monitors your dedicated IPs against major IP-based blocklist operators.
- *Domain blocklists* – Monitors your sending domains against major domain-based blocklist operators.

What blocklist monitoring provides

The blocklists page displays the following information:

- *Summary* – The number of monitored domains and IPs, and how many are currently listed on a blocklist.
- *Monitored domains* – A table showing each monitored domain, its current status (Listed or Clean), and the number of blocklists it appears on.
- *Monitored IPs* – A table showing each monitored IP address, its current status, and the number of blocklists it appears on.
- *Listing details* – For each listed entity, the blocklist name, listing date, reason, and delisting procedure with direct links to the blocklist operator's removal process.

Viewing blocklist status using the Amazon SES console

To view blocklist status using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Blocklists** under **Global deliverability** in the **Virtual Deliverability Manager** section.
3. Review the **Summary** section for an overview of your monitoring status.

4. In the **Monitored domains** or **Monitored IPs** tables, choose **Show details** for any entity to view its blocklist listings.
5. For listed entities, review the listing reason and follow the delisting procedure provided to request removal from the blocklist.

> **Note**
>
> When a domain or IP is listed on a blocklist, Virtual Deliverability Manager Advisor generates a recommendation with guidance on how to resolve the issue. See the section called “Advisor” for more information.

Email Validation for Amazon SES

Email list quality is a critical factor in maintaining high delivery rates and protecting your sender reputation. Invalid or risky email addresses can lead to high bounce rates, spam complaints, and mailbox providers may reject emails, blacklist sending domains, or throttle delivery rates.

Email Validation is an Amazon SES feature that helps you validate email addresses before you send to them. It checks email addresses for syntax errors, domain validity, and other checks, and identifies risky addresses that could harm your sender reputation, helping you maintain high deliverability rates and protect your sender reputation.

Why are email list quality and validation are important ?

Mailbox providers monitor key metrics such as bounce rates, complaint rates, and engagement levels to evaluate sender reputation. When these metrics exceed acceptable thresholds, providers implement protective measures such as rate limiting, temporary deferrals, or permanent blocks. High bounce rates (typically above 5-10%) signal poor list hygiene and trigger immediate reputation penalties.

Mailbox providers expect senders to maintain clean email lists and demonstrate good sending practices. They track hard bounces from invalid addresses, monitor spam trap hits, and measure recipient engagement to determine inbox placement. Poor list quality results in measurable impacts: reduced delivery rates, increased latency, and potential blacklisting that affects all future campaigns from your sending domain or IP address.

How Email Validation can help improve deliverability and reputation ?

Email Validation helps you improve both your deliverability and reputation with an *API* that checks email addresses on demand, and *Auto Validation* that automatically reviews and filters outbound emails based on deliverability thresholds you set.

- **API validation** – Validates email addresses through API calls, allowing you to validate addresses at the point of collection such as during sign-up forms or subscription processes, and for periodic list cleaning of existing databases. This helps prevent invalid addresses from entering your database, provides timely feedback to users about address validity, and enables regular maintenance of email list quality.
- **Auto Validation** – Automatically reviews all outbound email addresses and only delivers messages to recipients that meet your selected delivery likelihood threshold (high or medium).

This protects your sender reputation by preventing sends to invalid or risky addresses without requiring manual intervention.

Topics

- Email Validation API
- Auto Validation
- Email Validation Dashboard

Email Validation API

API validation allows you to validate individual email addresses through API calls, providing immediate feedback about address validity, deliverability, and risk factors. This feature is designed for validating addresses at the point of collection, such as during user registration, subscription forms, or any other scenario where you need timely validation results.

API validation performs multiple checks on each email address, including syntax validation, domain verification, mailbox existence checks, and others. The validation results include confidence verdicts (HIGH, MEDIUM, or LOW) for overall validity and individual evaluations.

Validation checks performed

API validation performs the following evaluations on each email address:

- **Syntax Validation ('HasValidSyntax')** – Checks that the email address follows proper RFC standards and contains valid characters in the correct format.
- **DNS Records ('HasValidDnsRecords')** – Checks that the domain exists, has valid DNS records, and is configured to receive email.
- **Mailbox Existence ('MailboxExists')** – Checks that the mailbox exists and can receive messages without actually sending an email.
- **Role Address ('IsRoleAddress')** – Identifies role-based addresses (such as admin@, support@, or info@) that may have lower engagement rates.
- **Disposable Domain ('IsDisposable')** – Checks disposable or temporary email addresses that could negatively impact your sender reputation.
- **Random String Patterns ('IsRandomInput')** – Checks randomly generated patterns.

Using API validation with the Amazon SES console

The following procedure shows you how to validate an email address using the Amazon SES console.

To validate an email address using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Email addresses validation** under **Email Validation**.
3. In the **Validate email address** section, enter the email address you want to validate in the **Email address** field.
4. Choose **Validate**.

   The validation results appear in the **Validation results** panel, showing:

   - **IsValid** – Overall validity with a confidence verdict (HIGH, MEDIUM, or LOW).
   - **Evaluations** – Individual evaluation results with confidence verdicts for syntax, DNS records, mailbox existence, and risk factors listed above.

Using API validation with the AWS CLI

The following examples show you how to validate email addresses using the AWS CLI.

To validate an email address using the AWS CLI

You can use the `GetEmailAddressInsights` operation in the Amazon SES API v2 to validate email addresses. You can call this operation from the AWS CLI, as shown in the following examples.

- Validate a single email address:

```
aws --region us-east-1 sesv2 get-email-address-insights --email-address
 user@example.com
```

- The response looks similar to this:

```
{
    "MailboxValidation": {
        "IsValid": {
```

```
            "ConfidenceVerdict": "HIGH"
        },
        "Evaluations": {
            "HasValidSyntax": {
                "ConfidenceVerdict": "HIGH"
            },
            "HasValidDnsRecords": {
                "ConfidenceVerdict": "MEDIUM"
            },
            "MailboxExists": {
                "ConfidenceVerdict": "MEDIUM"
            },
            "IsRoleAddress": {
                "ConfidenceVerdict": "LOW"
            },
            "IsDisposable": {
                "ConfidenceVerdict": "LOW"
            },
            "IsRandomInput": {
                "ConfidenceVerdict": "LOW"
            }
        }
    }
}
```

- For more information about response values and data types, see the `MailboxValidation` data type in the Amazon SES API v2 reference.
- Ensure your IAM identity has the required permissions for Email Validation API calls and CloudWatch metrics publishing:

```
{
  "Version": "2012-10-17       ",
  "Statement": [
    {
      "Sid": "EmailValidationPermissions",
      "Effect": "Allow",
      "Action": [
        "ses:GetEmailAddressInsights",
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

The `GetEmailAddressInsights` permission is required for API validation calls, and `CreateServiceLinkedRole` enables CloudWatch metrics publishing for validation activity.

Interpreting validation results

The validation response includes confidence verdicts to help you make decisions about email addresses:

- `IsValid` – Overall validity assessment with a confidence verdict of HIGH, MEDIUM, or LOW. HIGH validity confidence indicates high delivery likelihood for the email address, MEDIUM indicates moderate delivery likelihood and LOW indicates low delivery likelihood.
- `Evaluations` – Individual evaluation results, each with a confidence verdict:
  - `HIGH` – Strong indication of the specific check (e.g., HIGH for IsRandomInput means the email is very likely randomly generated).
  - `MEDIUM` – Moderate indication of the specific check (e.g., MEDIUM for IsRandomInput means that there is some likelihood that the email address is randomly generated).
  - `LOW` – Weak or no indication of the specific check (e.g., LOW for IsRandomInput means that the email address is less likely randomly generated).

Auto Validation

Auto Validation automatically reviews all outbound email addresses before sending and only delivers messages to recipients that meet your selected validation threshold. This helps you protect your sender reputation by preventing sends to likely invalid or risky addresses without requiring manual intervention or API integration.

When Auto Validation is enabled, Amazon SES validates each recipient address as part of attempting delivery. Addresses that don't meet your threshold are automatically suppressed. You can also set up configuration set event destinations to track which emails did not pass the validation threshold.

Validation thresholds

Auto Validation currently supports three validation thresholds:

- **SES managed** – Amazon SES automatically manages the threshold to suppress invalid addresses. This option allows Amazon SES to optimize the validation threshold based on your sending patterns and reputation.
- **High** – Allows emails to be sent only to addresses with high delivery likelihood. This provides maximum protection for your sender reputation but may suppress some legitimate addresses with medium delivery confidence.
- **Medium** – Allows emails to be sent to addresses with medium or high delivery likelihood. This balances reputation protection with delivery reach by allowing addresses with medium and high delivery confidence. This suppresses delivery to email addresses with low delivery confidence.

> **Important**
>
> If you choose High or Medium thresholds instead of SES managed, it is important to monitor your delivery metrics and validation results regularly.

Managing Auto Validation with the Amazon SES console

The following procedure shows you how to enable or change Auto Validation settings using the Amazon SES console.

To manage Auto Validation using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Auto Validation** under **Email Validation**.
3. Select **Enabled** checkbox to turn on the feature.
4. Choose a validation threshold.
5. Choose **Save changes**.

   The **Auto Validation** panel shows your updated settings.

> **Important**
>
> Auto Validation applies to all outbound emails sent through your account. Addresses that don't meet your threshold will be suppressed. You also have an option to enable auto

validation at the configuration set level. Suppressed sends still count towards your daily send quota and you will still be charged the standard outgoing message fee for suppressed sends in addition to the fee for auto validation. For pricing information, see the SES pricing page.

Amazon SES reports each suppressed send as a `Bounce` event with a `bounceType` of `Permanent` and a `bounceSubType` of `EmailValidationSuppressed`. These bounces don't count toward the *Reputation.BounceRate* metric for your account. They are counted under the *Bounce* metric in the AWS/SES namespace.

Enable Auto Validation at Configuration set level

You can override account-level Auto Validation settings for specific configuration sets. This allows you to apply different validation thresholds for different types of email campaigns.

To configure Auto Validation for a configuration set

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation pane, choose **Configuration sets** under **Configuration**.
3. Choose the configuration set you want to configure.
4. In the **Auto Validation options** section, choose **Edit**.
5. Select the **Override account level Auto Validation settings** checkbox.
6. Select the **Auto Validation enabled** checkbox to enable Auto Validation for this configuration set.
7. For **Validation threshold**, select either:
   - **SES managed** – Amazon SES automatically manages the threshold.
   - **High** – Only addresses with high delivery likelihood.
   - **Medium** – Addresses with medium delivery likelihood.
8. Choose **Save changes**.

If you don't override the account-level settings, the configuration set will use the Auto Validation settings defined at the account level. You can also set up event destinations to track which emails did not pass the validation threshold.

Managing Auto Validation with the AWS CLI

The following examples show you how to enable and configure Auto Validation using the AWS CLI.

To manage Auto Validation using the AWS CLI

You can use the PutAccountSuppressionAttributes operation in the Amazon SES API v2 to manage Auto Validation. You can call this operation from the AWS CLI, as shown in the following examples.

- Enable Auto Validation with a high threshold:

```
aws --region us-east-1 sesv2 put-account-suppression-attributes --cli-input-json
 file://auto-validation.json
```

  The input file looks similar to this:

```
{
    "SuppressedReasons": ["BOUNCE", "COMPLAINT"],
    "ValidationOptions": {
        "ConditionThreshold": {
            "ConditionThresholdEnabled": "ENABLED",
            "OverallConfidenceThreshold": {
                "Verdict": "HIGH"
            }
        }
    }
}
```

- Change the threshold to medium:

```
{
    "SuppressedReasons": ["BOUNCE", "COMPLAINT"],
    "ValidationOptions": {
        "ConditionThreshold": {
            "ConditionThresholdEnabled": "ENABLED",
            "OverallConfidenceThreshold": {
                "Verdict": "MEDIUM"
            }
        }
    }
}
```

- Use SES managed threshold:

```
{
    "SuppressedReasons": ["BOUNCE", "COMPLAINT"],
    "ValidationOptions": {
        "ConditionThreshold": {
            "ConditionThresholdEnabled": "ENABLED",
            "OverallConfidenceThreshold": {
                "Verdict": "MANAGED"
            }
        }
    }
}
```

- Disable Auto Validation:

```
{
    "SuppressedReasons": ["BOUNCE", "COMPLAINT"],
    "ValidationOptions": {
        "ConditionThreshold": {
            "ConditionThresholdEnabled": "DISABLED"
        }
    }
}
```

- To verify the outcome:

```
aws --region us-east-1 sesv2 get-account
```

For more information about parameter values and data types, see the SuppressionAttributes data type in the Amazon SES API v2 reference.

To configure Auto Validation for a configuration set using the AWS CLI

You can use the PutConfigurationSetSuppressionOptions operation to override Auto Validation settings for a specific configuration set.

- Override account-level settings for a configuration set:

```
aws --region us-east-1 sesv2 put-configuration-set-suppression-options --cli-input-
json file://config-set-auto-validation.json
```

The input file looks similar to this:

```
{
    "ConfigurationSetName": "my-config-set",
    "SuppressedReasons": ["BOUNCE", "COMPLAINT"],
    "ValidationOptions": {
        "ConditionThreshold": {
            "ConditionThresholdEnabled": "ENABLED",
            "OverallConfidenceThreshold": {
                "Verdict": "HIGH"
            }
        }
    }
}
```

- To verify the outcome:

```
aws --region us-east-1 sesv2 get-configuration-set --configuration-set-name my-
config-set
```

Email Validation Dashboard

The Email Validation Dashboard provides visibility into your email validation activity, showing validation results categorized by delivery likelihood confidence levels. You can monitor validation trends for API validations and console validations to understand the quality of your email lists and make informed decisions about your sending practices.

Accessing the dashboard

To view the Email Validation Dashboard:

1. Sign in to the Amazon SES console at https://console.aws.amazon.com/ses/.
2. In the navigation pane, choose **Email Validation**.
3. Choose the **Email Validation Dashboard** menu item to view validation metrics.

The dashboard provides a visual representation of your validation activity, helping you identify trends and make data-driven decisions about list hygiene and sending practices.

Email validation dashboard Info

Email validation dashboard lets you view metrics for the Email validation feature (API and Console) and break it down by validation confidence levels.

Email validation dashboard

Email validation metrics for US East (N. Virginia)

Date range

Last 7 days

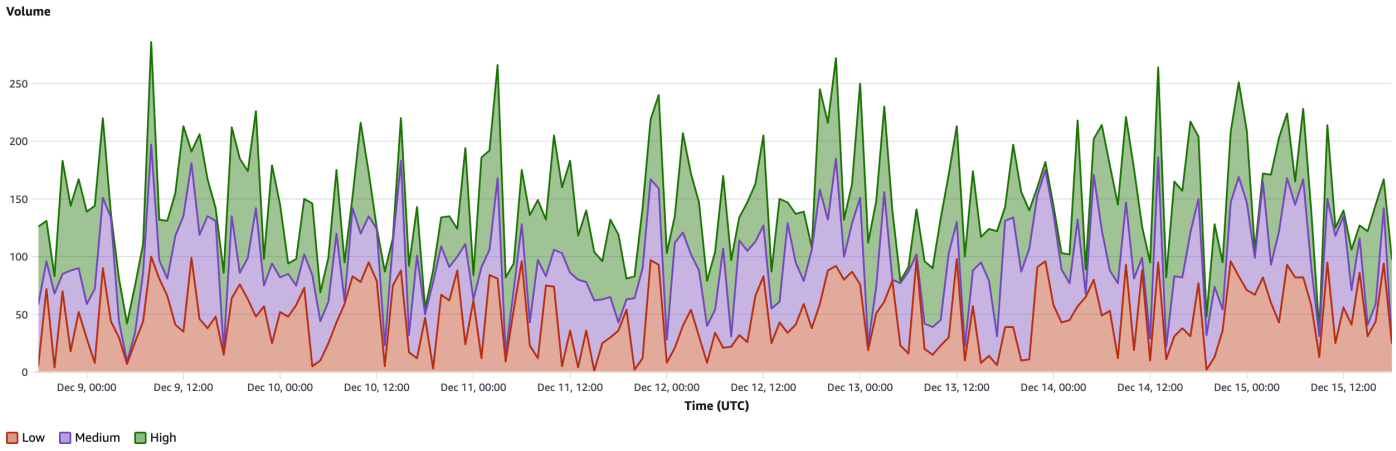
Email validation results counts

The following metrics show total volumes of validation confidence levels for the date range selected.

Low	Medium	High
8,026	8,322	8,757

Email validation results trends

High validity confidence indicates high delivery likelihood for the email address, medium indicates moderate delivery likelihood and low indicates low delivery likelihood. This metric is emitted when the GetEmailAddressInsights API is called or the Email addresses validation page is used.



Mail Manager for Amazon SES

Mail Manager is a set of Amazon SES email gateway features designed to help you strengthen your organization's email infrastructure, simplify email workflow management, and streamline email compliance control. It integrates with your existing infrastructure, can connect different business applications, and automates inbound email processing. Mail Manager also acts as a first line of defense in maintaining a healthy email system by efficiently managing your email traffic and enhancing compliance with its email archival capability.

Along with current Amazon SES capabilities, Mail Manager consists of the following features that support inbound traffic:

- **Ingress endpoint** – A key infrastructure component that utilizes filtering polices and rules that you can configure to determine which emails should be allowed into your organization and which ones should be rejected.
- **Traffic policies and rule sets** – Enable email administrators to define and enforce rules for managing inbound email traffic with highly customizable polices and rules that can sort, categorize, prioritize, and perform actions on emails based on a rich set of conditions and exceptions you define. This intelligent filtering combined with automated workflows helps to streamline email management, enhance efficiency, and ensure compliance with your organizational email policies.
- **SMTP relay** – Redirects email traffic to other SMTP servers based on criteria you define in rules by connecting internal email systems, and streamlines email management with automatic forwarding. Being able to distribute traffic across multiple servers and gateways enables your organization to manage high volume email traffic effectively, even in hybrid environments.
- **Email archiving** – Saves and protects your emails by storing data in persistent and secure long-term storage, and gives you a way to quickly search and archive email. It provides full-time, enterprise-level archiving without increasing the storage requirements of your mailbox server.
- **Email Add Ons** – A collection of specialized security tools from SES approved providers that can be used to manage email coming into your ingress endpoint as well as providing routing options based on security results. These tools are certified security intelligence and enforcement solutions that are ready to be integrated into your email workflow and can be activated directly from the Mail Manager console.

Getting started with Mail Manager

To start using Mail Manager, an onboarding wizard in the Amazon SES console will walk you through the steps of enabling Mail Manager for your account. See the section called “Getting started”.

Topics

- Getting started with Mail Manager
- Ingress endpoints
- Traffic policies and policy statements
- Rule sets and rules
- SMTP relay
- Address Lists
- Email archiving
- Email Add Ons
- Permission policies for Mail Manager
- Mail Manager logging

Getting started with Mail Manager

To start using Amazon SES Mail Manager you can use the *Get started with Mail Manager* wizard in the Amazon SES console, where you'll create an ingress endpoint and configure it with a traffic policy and rule set.

An ingress endpoint is your first building block in setting up Mail Manager—it’s a key infrastructure component that utilizes:

- **Traffic policies** – A traffic policy contains policy statements that you define to sort the incoming mail by allowing or blocking specific types of email when the policy statement’s conditions are met.
- **Rule sets** – A rule set contains rules that you define to perform actions on the email you allow in when the rule’s conditions are met.

However, part of creating an ingress endpoint is selecting a traffic policy and a rule set that have already been created and then assigning them to the ingress endpoint. The steps in the following procedure will walk you through the correct order of configuring your first ingress endpoint.

Getting started with Mail Manager using the SES console

The following procedure shows you how to get started with Mail Manager using the SES console.

To get started with Mail Manager using the Amazon SES console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Mail Manager** and select any of the **Get started with Mail Manager** buttons on the **Mail Manager overview** page.
3. On the *Get set up* page, select **Create traffic policy** on the *Create a traffic policy* card.

    a. Complete the workflow on the *Create a traffic policy* page. If you need additional information, see the section called “Creating traffic policies & policy statements (console)”.

    b. After creating your first traffic policy and policy statements, use your browser's back button to return to the *Get set up* page or select **Get set up** under *Mail Manager* in the left navigation panel.
4. On the *Get set up* page, select **Create rule set** on the *Create a rule set* card.

    a. Complete the workflow on the *Create a rule set* page. If you need additional information, see the section called “Creating rule sets & rules (console)”.

    b. After creating your first rule set and rules, use your browser's back button to return to the *Get set up* page or select **Get set up** under *Mail Manager* in the left navigation panel.
5. Now that you've created your first traffic policy and rule set, you'll be able to create your first ingress endpoint. On the *Get set up* page, select **Create ingress endpoint** on the *Create an ingress endpoint* card.

    - Part of the workflow on the *Email ingress endpoint* page will be to assign the traffic policy and rule set you just created to the ingress endpoint. If you need additional information, see the section called “Creating an ingress endpoint (console)”.

With your first ingress endpoint created, you can start using Mail Manager and utilize its other features such as SMTP relays and email archiving. You can also create additional ingress endpoints with unique traffic policies and rule sets to further customize how you manage all of your incoming email.

Ingress endpoints

An ingress endpoint is the key infrastructure component in Mail Manager that receives, routes, and manages your email by utilizing policies and rules you configure to determine which emails should be rejected, which ones should be allowed, and which ones should be acted upon.

Each ingress endpoint has its own traffic policy to determine which emails to block or allow, and its own rule set to perform actions on the email you do allow in; therefore, by creating multiple ingress endpoints, you can delegate each one to manage and route specific types of email. This level of granularity will help you to build an email management system that's tailored to your business needs.

Prerequisite workflow to create an ingress endpoint

At the time of creating your ingress endpoint, you must assign it a traffic policy and a rule set that have *already been created*. Therefore, the workflow for creating an ingress endpoint should be in the following order:

1. Start by creating a traffic policy to determine the email you want to block or allow. For details, see the section called “Creating traffic policies & policy statements (console)”.
2. Next, create a rule set to perform actions on the email you allow in. For details, see the section called “Creating rule sets & rules (console)”.
3. Finally, create your ingress endpoint and assign to it the traffic policy and rule set you just created or any others you previously created.

Once you create your ingress endpoint, you must configure it with the environment you're using to receive email, whether that be the configuration of an on-premise SMTP client or a web-based DNS domain host. This is discussed below in the section called “Public endpoint configuration”.

Configuring your environment to use an ingress endpoint

SES supports both public endpoints and Amazon Virtual Private Cloud (VPC) endpoints for ingress endpoints to accept incoming email. The following sections explain how to configure your ingress endpoint to use either of these options.

Topics

- Receiving email through the public endpoints
- Receiving email through Amazon VPC endpoints

Receiving email through the public endpoints

Using the "A" record

At the time you create an ingress endpoint, an "A" record for the endpoint will be generated and its value displayed on the ingress endpoint's summary screen in the SES console. The way you use the value of this record depends on the type of endpoint you created and your use case:

- **Open endpoint** – Mail sent to your domain will resolve directly to your ingress endpoint—no authentication required.
  - Copy and paste the value of the "A" record either directly into the SMTP configuration of an on-premise SMTP client or into an MX record for your domain in your DNS configuration.
  - Supported port: 25
  - Supports STARTTLS: Yes
- **Authenticated endpoint** – Mail sent to your domain has to come from authorized senders whom you've shared your SMTP credentials with, such as your on-premise email servers.
  - Copy and paste the value of the "A" record directly into the SMTP configuration of an on-premise SMTP client as well as your user name and password.
  - Supported ports: 25, 587 (RFC 2476)
  - Supports STARTTLS: Yes
- **mTLS endpoint** – Mail sent to your domain must come from clients that present a TLS client certificate signed by one of the certificate authorities (CAs) in the ingress endpoint's trust store. See the section called "mTLS authentication".
  - Copy and paste the value of the "A" record directly into the SMTP configuration of an on-premise SMTP client.
  - Supported port: 25
  - Supports STARTTLS: Yes

If you're using an MX record in your configuration, keep in mind that while every DNS provider has different procedures and interfaces for configuring records, the key pieces of information you need to put into you DNS settings are listed in the following example:

All email sent to *recipient@marketing.example.com* will go to your ingress endpoint because you entered the ingress endpoint's "A" record as the value for an MX record in your domain's DNS settings:

- **Domain** – `marketing.example.com`
- **MX record value** – `890123abcdef.ghijk.mail-manager-smtp.amazonaws.com` *(This is the "A" record value copied from your ingress endpoint.)*
- **Priority** – `10`

Connecting to the authenticated endpoint

For the authorized senders whom you've shared your SMTP credentials with in order to connect to your authenticated endpoint, the following protocols must be followed for the *username* and *password* in order to establish a successful connection to the server:

- **Username** – This is the ingress endpoint ID and must be encoded in Base64. *(See Step 11. in the console procedures to learn how to find the ingress endpoint ID.)*
- **Password** – This is the one used during ingress endpoint creation and must be encoded in Base64.

The following example shows a typical SMTP AUTH server and client exchange establishing connection:

```
S: 250 AUTH LOGIN PLAIN
C: AUTH LOGIN
S: 334 VXNlcm5hbWU6
C: SW5ncmVzc1BvaW50
S: 334 UGFzc3dvcmQ6
C: SW5ncmVzc1Bhc3N3b3Jk
S: 235 Authentication successful
```

This example contains the following properties:

- S means "Server"—the SMTP server accepting messages.
- C means "Client"—the SMTP client establishing connection with the server and sending messages to server.
- `250 AUTH LOGIN PLAIN` is a response from the server with AUTH methods supported, `AUTH LOGIN` or `AUTH PLAIN`, the sender could choose either of them, and send SMTP commands compliant with the SMTP Service Extension for Authentication specification RFC 2554. `AUTH LOGIN` is used here.
- `334 VXNlcm5hbWU6` – Server prompting for the username in Base64.

- `SW5ncmVzc1BvaW50` – Client responding with ingress endpoint ID in Base64.
- `334 UGFzc3dvcmQ6` – Server prompting for the password in Base64.
- `SW5ncmVzc1Bhc3N3b3Jk` – Client responding with ingress endpoint password in Base64.

Receiving email through Amazon VPC endpoints

In addition to public ingress endpoints, you can use VPC endpoints with SES ingress endpoints for secure, private email ingestion within your private network infrastructure.

Configuration differences compared to using public ingress endpoints

- The "A" Record typically available for public endpoints is not provided.
- You must connect to the ingress endpoint using DNS names provided by your VPC endpoint.
- All connections use private networking within your VPC.

Types of ingress endpoints supported through VPC endpoints

SES supports two types of ingress points through VPC endpoints:

- **Open ingress endpoint** – Email sent to your domain route directly through the VPC endpoint without requiring sender authentication.

  Configuration requirements:

  - Create a private open ingress endpoint by associating it with a VPC endpoint ID you own.
  - Supported ports: 25, 587
  - Supports STARTTLS: Yes
- **Authenticated ingress endpoint** – Mail sent to your domain has to come from authorized senders whom you've shared your SMTP credentials with, such as your on-premise email servers.

  Configuration requirements:

  - Create a private authenticated ingress endpoint by associating it with a VPC endpoint ID you own.
  - Supported ports: 25, 587
  - Supports STARTTLS: Yes

- Authentication uses the same base64-encoded username and password mechanism as public authenticated endpoints.

- **mTLS ingress endpoint** – Mail sent to your domain must come from clients that present a TLS client certificate signed by one of the CAs in the ingress endpoint's trust store. See the section called “mTLS authentication”.

  Configuration requirements:

  - Create a private mTLS ingress endpoint by associating it with a VPC endpoint ID you own.
  - Supported ports: 25, 587
  - Supports STARTTLS: Yes

VPC endpoint requirements

To use a VPC endpoint with an SES ingress endpoint, the following requirements must be met:

- The VPC endpoint must be active and available.
- The VPC endpoint must be owned by the same AWS account as the ingress endpoint (cross-account access is not supported).
- The VPC endpoint must be created for the appropriate service name based on the type of ingress endpoint:
  - **Open ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.open`
  - **Authenticated ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.auth`
  - **mTLS ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.mtls`
  - **FIPS open ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.open.fips`
  - **FIPS authenticated ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.auth.fips`
  - **FIPS mTLS ingress endpoint** – `com.amazonaws.region.mail-manager-smtp.mtls.fips`

Important configuration notes

- **One-to-one relationship** – Each VPC endpoint can only be associated with a single ingress endpoint. You cannot use the same VPC endpoint for multiple ingress endpoints.

- **No VPC endpoint policies** – Unlike other AWS services, VPC endpoints used with ingress endpoints do not support VPC endpoint policies. SES automatically verifies that the VPC endpoint owner and the ingress endpoint owner are the same AWS account.
- **Private DNS only** – All DNS names provided by the VPC endpoint will be private DNS names accessible only within your VPC.
- **Validation at creation time** – SES performs validation during resource creation to ensure the VPC endpoint meets all requirements.
- **TLS policy must match VPC endpoint service** – When creating a private ingress endpoint, the TLS policy value must match the VPC endpoint service type. An ingress endpoint with a `FIPS` TLS policy must use a FIPS VPC endpoint service, and an ingress endpoint with a `REQUIRED` or `OPTIONAL` TLS policy must use a non-FIPS VPC endpoint service. They cannot be mixed.

Connecting to your ingress endpoint through a VPC endpoint

After configuring your VPC endpoint and ingress endpoint:

1. Retrieve the DNS names generated for your VPC endpoint.
2. Configure your SMTP clients or email servers to use these DNS names for connection.
3. If using an authenticated endpoint, configure your SMTP clients with the appropriate base64-encoded credentials used with your authenticated ingress endpoint.

SMTP credentials rotation for authenticated ingress endpoints

When using an authenticated ingress endpoint, you may need to periodically rotate the SMTP password used by your sending clients. The rotation behavior depends on how you store your credentials: using AWS Secrets Manager or using SES managed storage.

AWS Secrets Manager-based password storage

If you configured your authenticated ingress endpoint with a secret stored in AWS Secrets Manager, Mail Manager uses the `AWSCURRENT` version of the secret for authentication and also accepts the `AWSPREVIOUS` version. This means that both the old and new passwords are valid during and after rotation, giving your sending clients time to update their credentials.

The `AWSPREVIOUS` version does not expire on the Mail Manager side—it remains valid as long as it exists in AWS Secrets Manager. It is your responsibility to manage secret versions and expiration through AWS Secrets Manager.

To set up automatic rotation:

1. Configure automatic rotation on your secret in AWS Secrets Manager.
2. When rotation occurs, AWS Secrets Manager moves the `AWSCURRENT` label to the new secret version and attaches `AWSPREVIOUS` to the former current version.
3. Mail Manager immediately accepts both versions, so sending clients can update their credentials without downtime.

For more information about AWS Secrets Manager rotation strategies, see Rotate secrets in the *AWS Secrets Manager User Guide*.

SES managed password storage

If you set the password directly when creating the ingress endpoint (or update it using the `UpdateIngressPoint` API), Mail Manager supports the previous password for 14 days after the update. After 14 days, sending clients using the old password receive `Authentication failed` errors.

Coordinating rotation with sending clients

When planning a credentials rotation, notify your sending application owners in advance and ensure they update their SMTP configurations within the grace period. For AWS Secrets Manager-based storage, the grace period lasts as long as the `AWSPREVIOUS` version exists. For SES managed storage, the grace period is 14 days.

> **Note**
>
> For seamless credentials rotation with minimal client disruption, use AWS Secrets Manager-based password storage with automatic rotation configured.

TLS policy for ingress endpoints

The TLS policy for an ingress endpoint controls whether connecting SMTP clients are required to use TLS encryption when sending email to your endpoint. You can specify a TLS policy when creating an ingress endpoint using the `CreateIngressPoint` API, and change it later using the `UpdateIngressPoint` API. The default TLS policy depends on your region: `FIPS` is the default in US and Canada regions, and `REQUIRED` is the default in all other regions.

All ingress endpoint connections use opportunistic TLS through the STARTTLS command. The connection begins as plaintext and is upgraded to TLS if the connecting client supports it. Implicit TLS (TLS Wrapper), where the connection starts encrypted, is not supported.

The following TLS policy values are available:

- **FIPS** – Requires TLS encryption using FIPS-validated cryptographic modules. This is the default in US and Canada regions and is only available in those regions.
- **REQUIRED** – Connecting SMTP clients must use TLS encryption. Connections that do not use TLS are rejected. This is the default in regions outside of the US and Canada.
- **OPTIONAL** – TLS encryption is supported but not required. Connecting SMTP clients can send email with or without TLS.

Availability by ingress endpoint type

Not all TLS policy values are valid for every combination of ingress endpoint type and network configuration:

- **FIPS** – Can be used with all ingress endpoint types (open, authenticated, and mTLS) on both public and private networks, but only in US and Canada regions. Once set, `FIPS` cannot be changed to another value through an update. If you need a different TLS policy, you must create a new ingress endpoint.
- **REQUIRED** – Can be used with all ingress endpoint types in all regions. However, for authenticated and mTLS ingress endpoints on public networks, `REQUIRED` can only be set at creation time—it cannot be changed through an update. For open ingress endpoints (public or private) and authenticated or mTLS ingress endpoints on private networks, `REQUIRED` can be set at creation time and changed through an update. Note that `REQUIRED` is not available for authenticated or mTLS ingress endpoints on public networks in US and Canada regions, where `FIPS` is used instead.
- **OPTIONAL** – Can be used with open ingress endpoints on both public and private networks, and with authenticated ingress endpoints on private networks. `OPTIONAL` is not available for mTLS ingress endpoints, and is not available for authenticated ingress endpoints on public networks.

Rules for changing TLS policy

The following rules apply when updating the TLS policy on an existing ingress endpoint:

- `FIPS` cannot be changed after creation.
- For open ingress endpoints and authenticated ingress endpoints on private networks, you can switch between `REQUIRED` and `OPTIONAL`.
- For mTLS ingress endpoints and authenticated ingress endpoints on public networks, the TLS policy cannot be changed after creation.

Mutual TLS (mTLS) authentication for ingress endpoints

Mutual TLS (mTLS) authentication requires connecting SMTP clients to present a TLS client certificate signed by one of the certificate authorities (CAs) in the ingress endpoint's trust store. Only clients with trusted certificates can send email to your endpoint.

To create an mTLS ingress endpoint, choose `MTLS` as the ingress endpoint type and provide a `TlsAuthConfiguration` containing a `TrustStore` in the `IngressPointConfiguration` parameter of the `CreateIngressPoint` API.

Trust store configuration

The trust store defines which client certificates are accepted by your ingress endpoint. It contains the following fields:

- **CAContent** (required) – A certificate authority (CA) certificate bundle in PEM format. This bundle contains the CA certificates used to validate client certificates. You can include multiple CA certificates in a single bundle, up to 500 KB.
- **CrlContent** (optional) – A certificate revocation list (CRL) in PEM format. If provided, client certificates that appear on the CRL are rejected even if they are signed by a trusted CA. Up to 500 KB.
- **KmsKeyArn** (optional) – The ARN of a AWS KMS customer managed key (CMK) used to encrypt the trust store data. If not specified, an AWS managed key is used. When using a CMK, the key policy must allow SES to use the key. See KMS customer managed key (CMK) key policy for mTLS trust store.

Expired certificates are considered invalid and are not accepted in connections. SES also filters out expired CA certificates and expired certificate revocation lists (CRLs) from your trust store. If a CRL expires, the CA certificate associated with that CRL is also removed from the trust store, which means clients signed by that CA will no longer be able to connect until you provide an updated CRL.

Using client certificate attributes in rule conditions

When a client connects to an mTLS ingress endpoint with a valid certificate, the certificate attributes (such as Common Name, serial number, and Subject Alternative Name fields) are made available for use in rule conditions as string expressions. This allows you to route, filter, or act on email based on the identity of the connecting client. For the full list of available attributes, see the Rule conditions reference.

Creating an ingress endpoint in the SES console

The following procedure shows you how to use the **Ingress endpoint** page in the SES console to create ingress endpoints and manage the ones you've already created.

To create an manage ingress endpoints using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Ingress endpoints** under **Mail Manager**.
3. On the **Ingress endpoints** page, select **Create ingress endpoint**.
4. On the **Create new ingress endpoint** page, enter a unique name for your ingress endpoint.
5. Choose whether it will be an **Open**, **Authenticated**, or **mTLS** endpoint.

   - If you choose **Authenticated**, select either **SMTP password** and enter a password (to be shared with authorized senders), or **Secret** and select one of your secrets from **Secret ARN**. *If you select a previously created secret, it must contain the policies indicated in the following steps for creating a new secret.*
   - If you choose **mTLS**, you must provide a trust store configuration containing your CA certificate bundle. Optionally, you can also provide a certificate revocation list and a AWS KMS key. See the section called “mTLS authentication”.
   - You have the option to create a new secret by choosing **Create new**—the AWS Secrets Manager console will open where you can continue to create a new key:

     a. Choose **Other type of secret** in **Secret type**.

     b. In **Key/value pair**, enter `password` for the key, and your actual password for the value.

> **Note**
>
> For **Key**, you must only enter `password` (anything else will cause authentication to fail).

c. Select **Add new key** to create a KMS customer managed key (CMK) in **Encryption key**—the AWS KMS console will open.

d. Choose **Create key** on the **Customer manged keys** page.

e. Keep the default values on the **Configure key** page and select **Next**.

f. Enter a name for your key in **Alias** (optionally, you can add a description and tag), followed by **Next**.

g. Select any users (other than yourself) or roles you want to permit to administer the key in **Key administrators** followed by **Next**.

h. Select any users (other than yourself) or roles you want to permit to use the key in **Key users** followed by **Next**.

i. Copy and paste the KMS CMK policy into the **Key policy** JSON text editor at the `"statement"` level by adding it as an additional statement separated by a comma. Replace the region and account number with your own.

j. Choose **Finish**.

k. Select your browser's tab where you have the AWS Secrets Manager **Store a new secret** page open and select the *refresh icon* (circular arrow) next to the **Encryption key** field, then click inside the field and select your newly created key.

l. Enter a name in the **Secret name** field on the **Configure secret** page.

m. Select **Edit permissions** in **Resource permissions**.

n. Copy and paste the Secrets resource policy into the **Resource permissions** JSON text editor and replace the region and account number with your own. (Be sure to delete any example code in the editor.)

o. Choose **Save** followed by **Next**.

p. Optionally configure rotation followed by **Next**.

q. Review and store your new secret by choosing **Store**.

r. Select your browser's tab where you have the SES **Create new ingress endpoint** page open and choose **Refresh list**, then select your newly created secret in **Secret ARN**.

6. Select a rule set containing the rule actions you want to perform on the email you allow in.

7. Select a traffic policy to determine the email you want to block or allow.
8. Choose whether it will be a **Public** or **Private** network.
   - For a public network, choose either **IPv4** only or **Dualstack** (IPv4 and IPv6) addressing.
   - For a private network, select or enter a VPC endpoint that you've shared with authorized senders in the same account, such as IAM users or roles. Optionally, you can create a new VPC endpoint by choosing **Create VPC endpoint** to open the Amazon VPC console.
9. Select a TLS policy for your ingress endpoint. The default depends on your region—see TLS policy for details on available values and restrictions.
10. Select **Create ingress endpoint**.
11. In **General details**, "Provisioning" will be displayed while your ingress endpoint is being created—refresh the page until "Active" is displayed and the **ARecord** field contains a value. Copy the "A" record value and paste it into your DNS configuration or your SMTP client as discussed in Public endpoint configuration.
12. Just above the **General details** container on the console, there is a large, unlabeled number prefixed by "inp" (also replicated in the breadcrumb trail at the top of the page), for example, **inp-1abc2de3fghi4jkl5mnop6qr**. This is referred to as the *ingress endpoint ID*, its value is used as the *username* to login to your ingress server. (You'll need to share this with your authorized senders to connect to your endpoint.)
13. You can view and manage the ingress endpoints you've already created from the **Ingress endpoints** page. If there's an ingress endpoint you want to remove, select it's radio button followed by **Delete**.
14. To edit an ingress endpoint, select its name to open its summary page:
    - You can change the endpoint's active status or TLS policy (for supported configurations) by choosing **Edit** in **General details** followed by **Save changes**.
    - You can select a different rule set or traffic policy by choosing **Edit** in either **Rule set** or **Traffic policy** followed by **Save changes**.

Traffic policies and policy statements

A traffic policy is a container for policy statements that you assign to an ingress endpoint so that it can sort the incoming mail by allowing or blocking specific types of email when the conditions of the policy statements are met. A traffic policy can be used by multiple ingress endpoints.

> **Tip**
>
> You can think of a traffic policy as a "filter set", and a policy statement as a "filter". The traffic policy (filter set) contains polices (filters) that you use to *filter* your incoming mail.

When you create a traffic policy, you have the option to set a maximum message size (in bytes). When a message exceeds that size, it is discarded. You can also set the default action of the policy to allow or block email that falls outside of the conditions of your policy statements—think of this as a “catch all” action for the traffic policy.

Policy statements are also created with either an allow or block action that is taken when the statements' conditions are met. You build the conditions by selecting an email property and a conditional operator for a value you enter that must be matched by the incoming message before the policy statement will allow or block it. Each policy statement can have multiple conditions.

A traffic policy can contain multiple policy statements and executes them in an order that's based on the implicit hierarchy of how it evaluates email:

- **Policy statements that block** – These statements are evaluated first and block any message that meets the statement's conditions.
- **Policy statements that allow** – These statements are evaluated next and allow any message that meets the statement's conditions.
- **Default action of traffic policy** – The remainder of messages that fall outside of the policy statements are allowed or blocked based on how you've defined this parameter.
- **Maximum message size** – If this optional parameter is set, any message greater than this size is discarded.

A traffic policy is an independent resource which can be used by more than one ingress endpoint, but policy statements belong exclusively to the traffic policy in which they were created. Thus, you must first create a traffic policy, or edit an existing one, before you can create policy statements to evaluate the email coming into your ingress endpoint.

The procedure in the next section explains how to create traffic policies and their policy statements in the SES console.

Creating traffic policies and policy statements in the SES console

The following procedure shows you how to use the **Traffic policies** page in the SES console to create traffic policies and their policy statements, and manage the ones you've already created.

To create and manage traffic policies and policy statements using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Traffic policies** under **Mail Manager**.
3. On the **Traffic policies** page, select **Create traffic policy**.
4. On the **Create a traffic policy** page, enter a unique name for your traffic policy.
5. (Optional) If you want to discard any messages above a certain size, enter a value in bytes in the **Maximum message size** field.
6. In **Default action**, choose whether the traffic policy is to either **Allow** or **Deny** (block) messages that fall outside of (are not addressed by) the conditions of your policy statements.
7. Select **Add new policy statement** to create a statement for your traffic policy.
8. Choose either **Allow** or **Deny** (block) for the action to be taken when the statement's conditions are met.
9. Build a condition by selecting an email property and a conditional operator for the value you enter. Select **Add new condition** if you want to add more conditions to this policy statement. *To learn more about a condition property and its operators and valid values, see the Policy statement conditions reference.*
   - If you're subscribed to an Email Add On, you'll be able to select it here as an email property.
   - Traffic policies associated with IPv6 or dual-stack ingress endpoints will not evaluate nor apply Email Add On conditions to messages received through IPv6 connections. Email Add On conditions will only apply to messages received through IPv4 connections on dual-stack endpoints.
10. If you want add more policy statements and conditions, repeat steps 7 - 9 above.
11. When you're done creating policy statements and their conditions, select **Create traffic policy**.
12. You can view and manage the traffic policies you've already created from the **Traffic policies** page. If there's an traffic policy you want to remove, select it's radio button followed by **Delete**.

13. To edit a traffic policy's properties or any of its policy statements, select its name to open its overview page, from here, select **Edit**.
14. In **Traffic policy details**, you can change the maximum message size and default action.
15. In any of the **Policy statement** containers, you can change the allow/deny property and edit any of the conditions. You can also remove policy statements and conditions, as well as add new ones.
16. When you're done with all your edits, save your changes by selecting **Save changes**.

Reference for policy statement conditions

Policy statement conditions

The following reference table lists all the policy statement properties that are available to build a policy statement condition. Selecting a property's expression type will take you to its reference page in the *SES Mail Manager API Reference* that lists all the available operators and valid values for that property.

Policy statement conditions: Properties, operators, and values

Property	Expression type
Recipient address Abusix Mail Intelligence *(if subscribed)* • Listed on Spamhaus Domain Block List *(if subscribed)* • Listed on	*Valid operators and values for string expressions*
Sender IP range	*Valid operators and values for IP expressions*
TLS protocol version	*Valid operators and values for TLS protocol expressions*
Abusix Mail Intelligence *(if subscribed)* • Is listed	*Valid operators and values for boolean expressions*

Property	Expression type
Spamhaus Domain Block List *(if subscribed)* • Is listed	

Recipient address matching and subaddressing

The *Recipient address* property uses the full SMTP envelope recipient address exactly as received, including any subaddress extension (also known as "plus addressing"). For example, if a message is sent to `user+tag@example.com`, the recipient address evaluated in the policy statement is `user+tag@example.com`, not `user@example.com`. This means that an `EQUALS` or `CONTAINS` rule targeting `user@example.com` will not match `user+tag@example.com`. If you need to match all subaddressed variants of an address, consider using an address list with wildcard entries (e.g., `user*@example.com`). Alternatively, `ENDS_WITH` with the domain (e.g., `@example.com`) can be used for broader domain-level matching.

Rule sets and rules

Rule sets are containers for rules that you assign to an ingress endpoint so that it can perform actions on email allowed in from the ingress endpoint's traffic policy. A rule set can be used by multiple ingress endpoints.

Rules tell your ingress endpoint how to handle incoming email by executing the actions defined in the rule when messages meet the rule's conditions. Each rule can have multiple conditions and actions. The rules you create within a rule set are executed in the order you specify within the rule set.

You build the rule's conditions by selecting an email property and a conditional operator for a value you enter that must be matched by the message before the rule will execute its actions—you define the actions to be taken as well as their order of execution.

For greater granularity, your rules can also contain exceptions that are defined similar to conditions, but here, you're defining a condition that the message must *not* match. Conditions and exceptions operate independently—you could build a rule with just exceptions if you wanted, as well as intermix conditions and exceptions.

Due to the fine granularity of how rules can be defined within a rule set, the following list is provided to help illustrate the relationship of rule set components:

- **Rule sets contain:**
  - **Rules** – You can define the order in which the rules are executed within the rule set.

    Rules contain:

    - **Conditions** – The rule applies if the message matches the evaluation of the condition(s); and if the rule has exceptions, see below.
    - **Exceptions** – The rule applies if the message does not match the evaluation of the exception(s); and if the rule has conditions, see above.
    - **Actions** – Actions are triggered when the rule applies—all of the conditions match and none of the exceptions.

      You can define the order in which the actions are executed within the rule.

Because each rule can have multiple conditions, exceptions, and actions, and the fact that you can define the order of how rules and actions are executed, this enables you to build a very customized and automated email handling solution tailored to your specific business requirements.

A rule set is an independent resource that can be used by more than one ingress endpoint, but rules belong exclusively to the rule set in which they were created. Thus, you must first create a rule set, or edit an existing one, before you can create rules to act upon the email coming into your ingress endpoint.

The procedure in the next section will walk you through creating rule sets and their rules in the SES console.

Creating rule sets and rules in the SES console

The following procedure shows you how to use the **Rule sets** page in the SES console to create rule sets and their rules, and manage the ones you've already created.

To create an manage rule sets and rules using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Rule sets** under **Mail Manager**.

3. On the **Rule sets** page, choose **Create rule set** and enter a unique name for your rule set.
4. On the rule set's overview page, select **Edit** , and then select **Create new rule** on the edit page.
5. In the **Rule details** sidebar, enter a unique name for your rule.
6. Select **Add new condition** to create a condition that the message must match; or check the **EXCEPT in the case of:** box followed by **Add new exception** to create a condition that the message must *not* match.
7. Build the condition or exception by selecting an email property and a conditional operator for the value you enter. Select **Add new condition** or **Add new exception** if you want to add more conditions or exceptions to this rule. *To learn more about a condition property and its operators and valid values, see the Rule conditions reference.*

   - If you're subscribed to an Email Add On, you'll be able to select it here as an email property.
8. Select **Add new action** to define the action to be taken when the rule's conditions are matched and/or exceptions are not matched. To add more actions to be taken, select **Add new action**. *When you create two or more actions, up/down arrows are displayed so that you can set the order of execution.*

   > **Note**
   >
   > To execute any of these Rule actions, you'll need to have their respective permission policy enabled for your account; otherwise, the rule action will fail.
9. Apply the permission policy for any of these actions directly from the **Rule details** panel after selecting the action:

   a. Choose **Create new role** in the **IAM role** field and enter a name followed by **Create role**. (The IAM trust policy for this role will automatically be generated in the background.)

   b. Because the IAM trust policy was automatically generated, you'll only need to add the action's permission policy to the role—select **View role** under the **IAM role** field to open the IAM console.

   c. Under the **Permissions** tab, choose **Add permissions** and select **Create inline policy**.

   d. On the **Specify permissions** page, select **JSON** in the **Policy editor**.

   e. Copy and paste the respective policy from Permission policies for Mail Manager into the **Policy editor** and replace the data in red text with your own. (Be sure to delete any example code in the editor.)

   f. Choose **Next**.

g. Review and create your permission policy for the IAM role by choosing **Create policy**.

h. Select your browser's tab where you have the SES Mail Manager **Edit rule set** page open and continue with the remaining steps for creating rules.

10. When you're done creating the conditions, exceptions, and actions for the rule, you save it to its rule set by choosing **Save rule set** located in the **Edit rule set** panel on the left.
11. If you want add more rules to the rule set, repeat steps 4 - 9 above.

    - When you create two or more rules, up/down arrows are displayed in the rule set's **Reorder** column so that you can set the order of execution.

12. You can view and manage the rule sets you've already created from the **Rule sets** page. If there's a rule set you want to remove, select it's radio button followed by **Delete**.
13. To edit a rule set, select its name to open its overview page, from here, select **Edit** where you can reorder the execution of its rules, add more rules by choosing **Create new rule**, or delete a rule by selecting it's radio button followed by **Delete**.
14. To edit a rule, select its radio button. In any of the containers on the **Rule details** sidebar, you can edit any of the conditions or exceptions and change or reorder any of the actions. You can also remove conditions, exceptions, and actions, as well as add new ones.
15. When you're done with all your edits, save your changes by selecting **Save rule set** located in the **Edit rule set** panel on the left.

Reference for rule conditions and actions

Rule conditions

The following reference table lists all the rule properties that are available to build a rule condition (or exception) and are categorized by their expression type. Rule properties that share the same expression type also share the same operators and values. Selecting a property's expression type will take you to its reference page in the *SES Mail Manager API Reference* that lists all the available operators and valid values for that property.

Rule conditions: Properties, operators, and values

Property	Expression type
From address To address	*Valid operators and values for string expressions*

Property	Expression type
CC address Mail from Recipient address Subject Helo MIME header Client certificate attribute *(mTLS ingress endpoints only)* • Serial number • Common name • Country • Locality • Organization • Organizational unit • State • Subject alternative name Vade Advanced Email Security *(if subscribed)* • Category • Verdict Trend Micro Virus Scanning *(if subscribed)* • Category	
IP range	*Valid operators and values for IP expressions*

Property	Expression type
Message max size	*Valid operators and values for number expressions*
DKIM SPF	*Valid operators and values for verdict expressions*
TLS TLS wrapped Read receipt Vade Advanced Email Security *(if subscribed)* • Is passed Trend Micro Virus Scanning *(if subscribed)* • Is passed	*Valid operators and values for boolean expressions*
DMARC policy	*Valid operators and values for DMARC expressions*

Recipient address matching and subaddressing

The *Recipient address* property uses the full SMTP envelope recipient address exactly as received, including any subaddress extension (also known as "plus addressing"). For example, if a message is sent to `user+tag@example.com`, the recipient address evaluated in the rule condition is `user+tag@example.com`, not `user@example.com`.
This means that an `EQUALS` or `CONTAINS` rule targeting `user@example.com` will not match `user+tag@example.com`. If you need to match all subaddressed variants of an address, consider using an address list with wildcard entries (e.g., `user*@example.com`). Alternatively, `ENDS_WITH` with the domain (e.g., `@example.com`) can be used for broader domain-level matching.

Rule actions

The following reference table lists all the rule actions that can be taken when a rule's conditions are met or its exceptions are not met. By selecting an action, you'll be taken to the action's reference page in the *SES Mail Manager API Reference* that lists the parameters and their formats for the action. The table uses the action names adopted in the Mail Manager console—the API names may differ slightly.

> **Note**
>
> In some of the API references, there will be an `ActionFailurePolicy` parameter that can be set to either *Continue* or *Drop* if the action fails—this only applies when using the API; when using the console, `ActionFailurePolicy` has been set to the default value of *Continue*.

Rule actions: Actions and parameters

Actions & their parameters	Description
Write to S3	Writes the MIME content of the email to an S3 bucket.
SMTP relay	Relays the email via SMTP to another specific SMTP server.
Archive	Archives the email by delivering it to an Amazon SES archive.
Add header	Adds a custom header to the received email.
Email recipients rewrite	Replaces the email envelope recipients with the given list of recipients. If the condition of this action applies only to a subset of recipient s, only those recipients are replaced.
Deliver to WorkMail mailbox	Delivers the email to an Amazon WorkMail mailbox.

Actions & their parameters	Description
Deliver to Q Business	Delivers an email to an Amazon Q Business application for ingestion into its knowledge base.
Publish to SNS	Publishes the email content to an Amazon SNS topic.
Send to internet	Uses SES to send the email to the recipient(s) on the email's recipient list.
Bounce	Bounces the email by returning a bounce response to the sender.
Invoke Lambda function	Invokes an AWS Lambda function to process the email. The Lambda payload format is the same as Amazon SES Email Receiving, as described in Notification contents. For more information, see Use case examples. For code examples, see Lambda function examples.
Drop	For email with multiple recipients, if this action applies to one or more (but not all) of those recipients, they will be dropped from the email's recipient list, and continued processing of rules will apply to remaining recipients. If this action applies to all recipient (s), rules processing stops as all recipients are dropped from the recipient list and will not receive the email.

SMTP relay

Because Mail Manager is deployed between your email environment (such as Microsoft 365, Google Workspace, or On-Premise Exchange) and the internet, Mail Manager uses SMTP relays to route incoming emails that are processed by Mail Manager to your email environment. It can also route

outbound emails to another email infrastructure such as another Exchange server or a third-party email gateway before sending to end recipients.

A SMTP relay is a vital component of your email infrastructure, responsible for efficiently routing emails between servers when designated by a rule action defined in a rule set.

Specifically, a SMTP relay can redirect incoming email between SES Mail Manager and an external email infrastructure such as Exchange, on-premise or third-party email gateways, and others. Incoming emails to an ingress endpoint will be processed by a rule that will route specified email to the designated SMTP relay, which in turn, will pass it on to the external email infrastructure defined in the SMTP relay.

When your ingress endpoint receives email, it uses a traffic policy to determine which emails to block or allow. The email you allow in passes to a rule set that applies conditional rules to execute the actions you've defined for specific types of email. One of the rule actions you can define is *SMTPRelay action*—if you select this action, the email will be passed along to the external SMTP server defined in your SMTP relay.

For example, you could use the *SMTPRelay action* to send email from your ingress endpoint to your on-premise Microsoft Exchange Server. You would set up your Exchange server to have a *public* SMTP endpoint that can only be accessed using certain credentials. When you create the SMTP relay, you enter the server name, port, and credentials of your Exchange server and give your SMTP relay a unique name, say, "RelayToMyExchangeServer". Then, you create a rule in your ingress endpoint's rule set that says, "When *From address* contains *'gmail.com'*, then perform *SMTPRelay action* using the SMTP relay called *RelayToMyExchangeServer*".

> **Note**
>
> All SMTP endpoints you configure with Amazon SES SMTP relay must be public as private SMTP endpoints are not supported.

Now, when email from *gmail.com* arrives to your ingress endpoint, the rule will trigger the *SMTPRelay action* and contact your Exchange server using the credentials you provided when creating your SMTP relay and deliver the email to your Exchange server. Thus, email received from *gmail.com* is *relayed* to your Exchange server.

You must first create an SMTP relay before it can be designated in a rule action. The procedure in the next section will walk you through creating an SMTP relay in the SES console.

Creating an SMTP relay in the SES console

The following procedure shows you how to use the **SMTP relays** page in the SES console to create SMTP relays and manage the ones you've already created.

To create and manage SMTP relays using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **SMTP relays** under **Mail Manager**.
3. On the **SMTP relays** page, select **Create SMTP relay**.
4. On the **Create SMTP relay** page, enter a unique name for your SMTP relay.
5. Depending on whether you want to configure an inbound (non-authenticated) or outbound (authenticated) SMTP relay, follow the respective instructions:

   Inbound

   To configure an inbound SMTP relay

   1. When SMTP relay is used as an inbound gateway to route incoming emails processed by Mail Manager to your external email environment, you will first need to configure the email hosting environment. While every email hosting provider has their own GUI and configuration workflow unique to them, the principals of configuring them to work with inbound gateways, such as your Mail Manager SMTP relay, will be similar.

      To help illustrate this, we have provided examples of how to configure Google Workspaces and Microsoft Office 365 to work with your SMTP relay as an inbound gateway in the following sections:

      - Setting up Google Workspaces
      - Setting up Microsoft Office 365

      > **Note**
      >
      > Ensure that your intended recipient destinations are SES verified email identities. For example, if you want to deliver email to recipients *abc@example.com*, *admin@example.com*, *postmaster@acme.com*, and *support@acme.com*, then we'd recommend that you verify the `example.com`

and `acme.com` domains in SES. If a recipient destination is not verified, SES will not attempt to deliver the email to the public SMTP server.

2. After you've configured Google Workspaces or Microsoft Office 365 to work with inbound gateways, enter the host name of the public SMTP server with the values below respective to your provider:

   - Google Workspaces: `aspmx.l.google.com`
   - Microsoft Office 365: `<your_domain>.mail.protection.outlook.com`

     Replace the dots with "-" in your domain name. For example, if your domain is *acme.com*, you would enter `acme-com.mail.protection.outlook.com`

3. Enter port number 25 for the public SMTP server.
4. Leave the Authentication section blank (do not select or create a secret ARN).

Outbound

To configure an outbound SMTP relay

1. Enter the host name of the public SMTP server you want your relay to connect to.
2. Enter the port number for the public SMTP server.
3. Setup authentication for your public SMTP server by selecting one of your secrets from **Secret ARN**. *If you select a previously created secret, it must contain the policies indicated in the following steps for creating a new secret.*

   - You have the option to create a new secret by choosing **Create new**—the AWS Secrets Manager console will open where you can continue to create a new key:

     a. Choose **Other type of secret** in **Secret type**.

     b. Enter the following keys and values in **Key/value pairs**:

Key	value
`username`	*my_username*
`password`	*my_password*

> **Note**
>
> For both of the keys, you must only enter `username` and `password` as shown (anything else will cause authentication to fail). For the values, enter your own username and password respectively.

c. Select **Add new key** to create a KMS customer managed key (CMK) in **Encryption key**—the AWS KMS console will open.

d. Choose **Create key** on the **Customer manged keys** page.

e. Keep the default values on the **Configure key** page and select **Next**.

f. Enter a name for your key in **Alias** (optionally, you can add a description and tag), followed by **Next**.

g. Select any users (other than yourself) or roles you want to permit to administer the key in **Key administrators** followed by **Next**.

h. Select any users (other than yourself) or roles you want to permit to use the key in **Key users** followed by **Next**.

i. Copy and paste the KMS CMK policy into the **Key policy** JSON text editor at the `"statement"` level by adding it as an additional statement separated by a comma. Replace the region and account number with your own.

j. Choose **Finish**.

k. Select your browser's tab where you have the AWS Secrets Manager **Store a new secret** page open and select the *refresh icon* (circular arrow) next to the **Encryption key** field, then click inside the field and select your newly created key.

l. Enter a name in the **Secret name** field on the **Configure secret** page.

m. Select **Edit permissions** in **Resource permissions**.

n. Copy and paste the Secrets resource policy into the **Resource permissions** JSON text editor and replace the region and account number with your own. (Be sure to delete any example code in the editor.)

o. Choose **Save** followed by **Next**.

p. Optionally configure rotation followed by **Next**.

q. Review and store your new secret by choosing **Store**.

r. Select your browser's tab where you have the SES **Create SMTP relay** page open and choose **Refresh list**, then select your newly created secret in **Secret ARN**.

6. Select **Create SMTP relay**.
7. You can view and manage the SMTP relays you've already created from the **SMTP relays** page. If there's an SMTP relay you want to remove, select it's radio button followed by **Delete**.
8. To edit an SMTP relay, select its name. On the details page, you can change the relay's name, the external SMTP server's name, port, and login credentials by selecting the corresponding **Edit** or **Update** button followed by **Save changes**.

Setting up Google Workspaces for inbound (non-authenticated) SMTP relay

The following walkthrough example shows you how to setup Google Workspaces to work with a Mail Manager inbound (non-authenticated) SMTP relay.

Prerequisites

- Access to the Google administrator console (Google administrator console > Apps > Google Workspace > Gmail).
- Access to the domain nameserver hosting the MX records for the domains which will be used for Mail Manager setup.

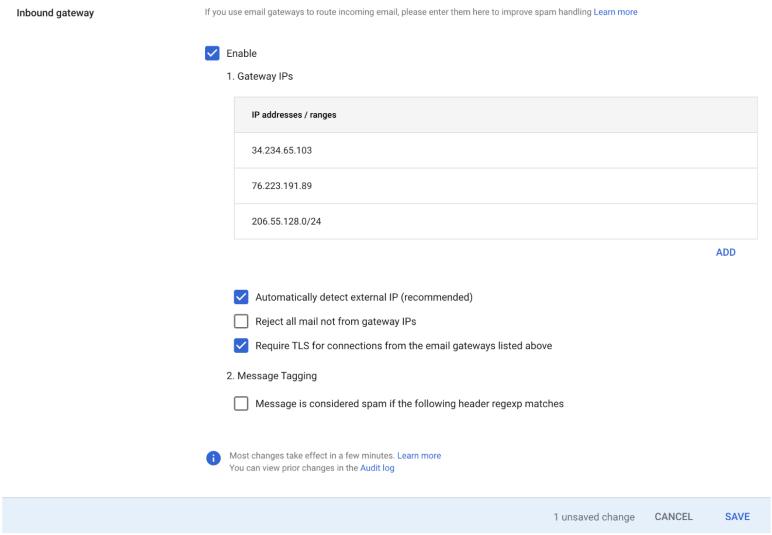
To setup Google Workspaces to work with an inbound SMTP relay

- **Add Mail Manager IP addresses to the Inbound gateway configuration**

  a. In the Google administrator console, go to **Apps > Google Workspace > Gmail**.

  b. Select **Spam, Phishing, and Malware**, then go to **Inbound gateway** configuration.

  c. Enable **Inbound gateway**, and configure it with the following details:



- In **Gateway IPs**, select **Add** , and add the ingress endpoint IPs specific to your region from the SMTP Relay IP Ranges table.
- Select **Automatically detect external IP**.
- Select **Require TLS for connections from the email gateways listed above**.
- Select **Save** at the bottom of the dialog box to save the configuration. Once saved, the administrator console will show the **Inbound gateway** as enabled.

Setting up Microsoft Office 365 for inbound (non-authenticated) SMTP relay

The following walkthrough example shows you how to setup Microsoft Office 365 to work with a Mail Manager inbound (non-authenticated) SMTP relay.

Prerequisites

- Access to the Microsoft Security admin center (Microsoft Security admin center > Email & collaboration > Policies & Rules > Threat policies).
- Access to the domain nameserver hosting the MX records for the domains which will be used for Mail Manager setup.

To setup Microsoft Office 365 to work with an inbound SMTP relay

1. **Add Mail Manager IP addresses to the Allow list**

a. In the Microsoft Security admin center, go to **Email & collaboration > Policies & Rules > Threat policies**.

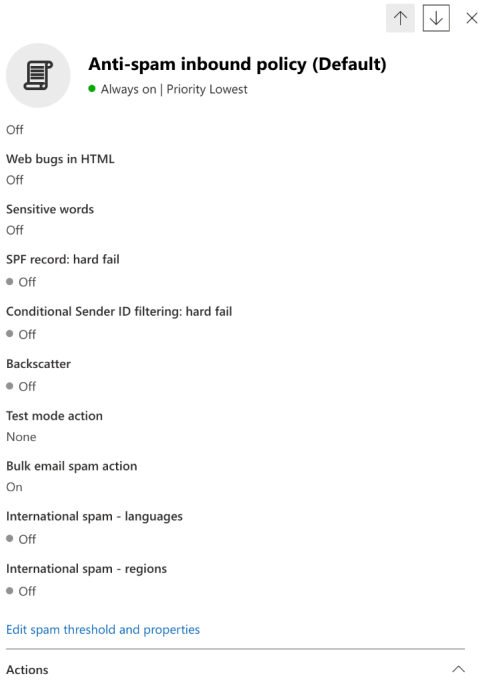
b. Select **Anti-spam** under **Polices**.

c. Select **Connection filter policy** followed by **Edit connection filter policy**.

- In the **Always allow messages from the following IP addresses or address range** dialog, add the ingress endpoint IPs specific to your region from the SMTP Relay IP Ranges table.
- Select **Save**.

d. Return to the **Anti-spam** option and choose **Anti-spam inbound policy**.

- At the bottom of the dialog, select **Edit spam threshold and properties**:



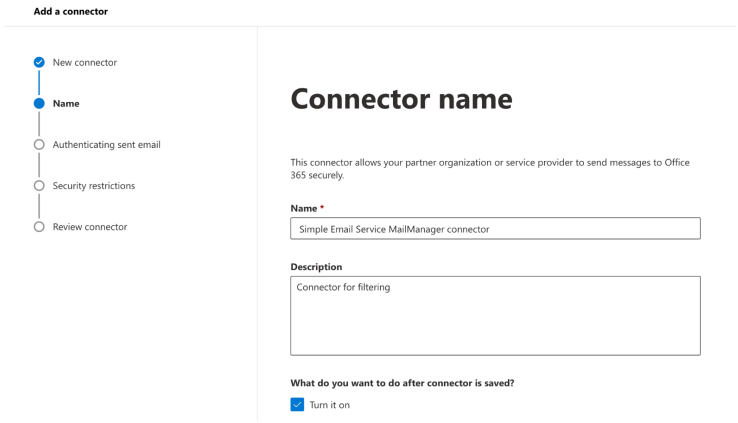
- Scroll to **Mark as spam** and ensure that **SPF record: hard fail** is set to **Off**.
- Select **Save**.

2. **Enhanced Filtering configuration (recommended)**

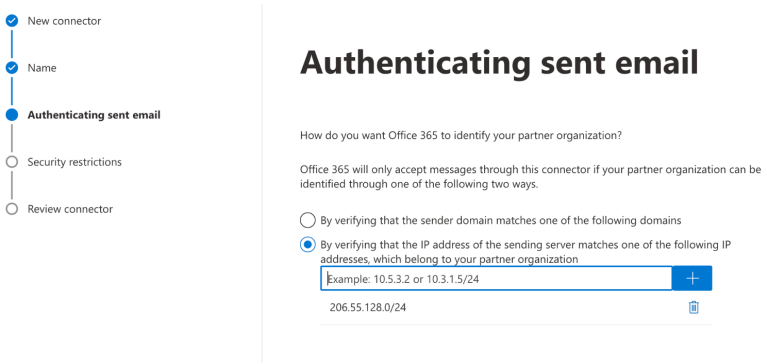
This option will allow Microsoft Office 365 to properly identify the original connecting IP before the message was received by SES Mail Manager.

a. **Create an inbound connector**

- Login to the new Exchange admin center and go to **Mail flow** > **Connectors**.
- Select **Add a connector**.
- In **Connection from**, select **Partner organization** followed by **Next**.
- Fill in the fields as follows:
  - **Name** – Simple Email Service Mail Manager connector
  - **Description** – Connector for filtering



- Select **Next**.
- In **Authenticating sent email**, select **By verifying that the IP address of the sending server matches one of the following IP addresses, which belong to your partner organization** and add the ingress endpoint IPs specific to your region from the SMTP Relay IP Ranges table.

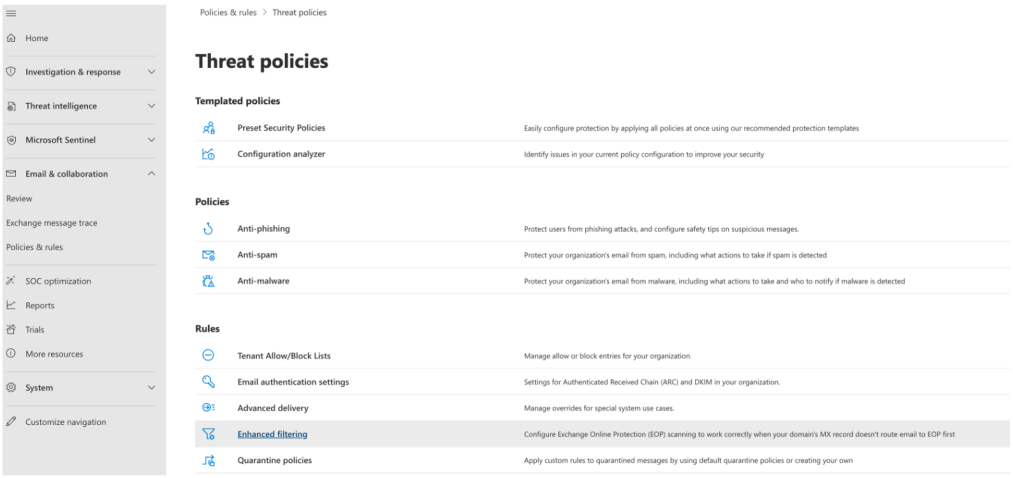


- Select **Next**.
- In **Security restrictions**, accept the default **Reject email messages if they aren't sent over TLS** setting, followed by **Next**.
- Review your settings and select **Create connector**.

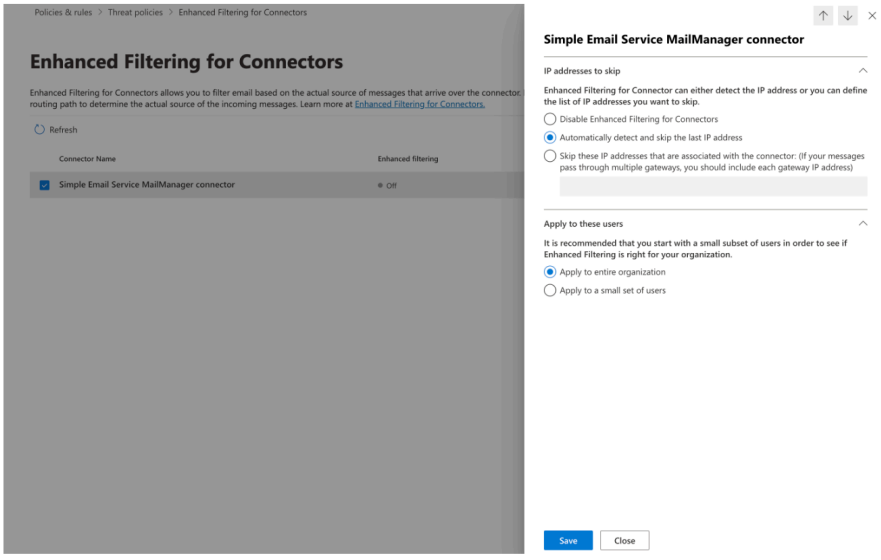
b. **Enable enhanced filtering**

Now that the inbound connector has been configured, you will need to enable the enhanced filtering configuration of the connector in the **Microsoft Security admin center**.

- In the Microsoft Security admin center, go to **Email & collaboration > Policies & Rules > Threat policies**.
- Select **Enhanced filtering** under **Rules**.



- Select the **Simple Email Service Mail Manager connector** that you created previously to edit its configuration parameters.
- Select both **Automatically detect and skip the last IP address** and **Apply to entire organization**.



- Select **Save**.

Address Lists

Address Lists is a Mail Manager feature that allows you to create and manage lists of email addresses and domains that you can use in traffic policies and rule sets to process incoming mail based on whether or not the recipient or sender of a message belongs to a specific list. Address Lists lends themselves to more granular control over email flows and help to simplify management of complex email routing scenarios.

What are Address Lists?

Address Lists are containers for email addresses and domains that you can use to filter and process email messages. They provide a convenient way to group related addresses and apply routing rules and traffic policies collectively.

Key use cases for Address Lists include:

- Deny lists for blocking known spam senders or domains
- Allow lists for ensuring delivery from trusted senders
- Recipient validation to reject emails to non-existent recipients early
- Role-based routing for applying different rules based on recipient roles
- Group-based policies for enforcing policies for specific user groups

How Address Lists work

Address Lists in SES streamline email management by allowing you to create and maintain collections of email addresses and domains. Once created, these lists are integrated into your email workflows through traffic policies and rules.

When SES processes an email, it checks the relevant Address List to determine if the sender or recipient is a member. Based on this membership and your configured policies and rules, SES then takes appropriate actions, such as routing, filtering, or rejecting the email. This process enables efficient and granular control over your email traffic.

Setting up Address Lists

Topics

- Creating and populating an Address List

- Managing Address Lists

Creating and populating an Address List

Part of creating an Address List in the console is to populate it with one or more addresses. Using the Mail Manager APIs, you can create empty Address Lists and populate them later. This section will show you how to do either with both console procedures and AWS CLI examples.

To create and populate an Address List:

1. Open the SES console at https://console.aws.amazon.com/ses/.
2. In the navigation pane under Mail Manager, choose **Address Lists**.
3. Choose **Create address list** and enter a name in the **Address list name** field.
4. Select either **Manual entry** or **Bulk upload** and follow the respective steps:
5. **For manual entry** – Enter one or more email addresses or domains in the console.

   If you use the asterisk (*) wildcard, the following formats apply:

   1. Only one `*` is allowed in the address:
      - The `*` should be either before or after @ when the entry is an email address.
      - When `*` is in the local part, the local part can be zero to 19 characters excluding the `*`.
      - When `*` is in the domain, the subdomain level can be 2 to 9 excluding the `*`.
   2. Examples of valid wildcard formats:
      - **.domain1.com to *.domain8.domain7...domain1.com*
      - **@domain.com to 123456789012345678 9*@domain.com*
      - *local@*.domain1.com to local@*.domain8.domain7...domain1.com*

   Wildcards can be used to match email addresses regardless of subaddress extensions (plus addressing). For example, adding `user*@example.com` to an address list will match any address whose local part starts with `user`, including `user@example.com`, `user+tag@example.com`, `username@example.com`, etc.
6. **For bulk upload** – Select **Choose file** and choose a CSV or JSON file from your computer containing the addresses to be uploaded.

   Use the format shown in the example for each file type:

1. CSV file example (Note that the header, `address`, is required.):

```
address
user1@domain.com
user2@*.domain.com
*@domain.com
```

2. JSON file example:

```
{
  "items": [
    {
      "address": "user1@domain.com"
    },
    {
      "address": "user2@*.domain.com"
    },
    {
      "address": "*@domain.com"
    }
  ]
}
```

7. Once you've finished adding addresses or have selected a bulk file, choose **Create address list**.

Using the AWS CLI:

Create the Address List

```
aws mailmanager create-address-list --address-list-name "MyDenyList"
```

Populate the Address List:

- **Single upload**

```
aws mailmanager register-member-to-address-list \
               --address-list-id al-123456789abc \
               --address "user@example.com"
```

- **Bulk upload**

For bulk uploads, you first have to create an import job specifying either a CSV or JSON format:

```
aws mailmanager create-address-list-import-job \
                --address-list-id "al-123456789abc" \
                --name "MyImportJob" \
                --import-data-format ImportDataType=CSV
```

This returns a job ID and a pre-signed URL. Use this pre-signed URL to upload your CSV or JSON file to an S3 bucket as shown in the following example using the curl command:

```
curl -X PUT -T "/path/to/file" "pre-signed URL"
```

After uploading, start the import job using the job ID returned in the previous command:

```
aws mailmanager start-address-list-import-job --job-id "job-123456789"
```

Managing Address Lists

You can update, view, and delete address lists as needed.

Topics

- Updating an Address List
- Viewing Address List details
- Deleting an Address List

Updating an Address List

You can update an address list by adding or removing addresses, and optionally, adding or removing tags.

To update an Address List:

1. On the **Address Lists** page, select the name of the Address List you want to edit.
2. To add addresses, choose **Add email address** and proceed with either the manual entry or bulk upload method as explained in Creating & populating an Address List.

3. To remove addresses, select the checkbox next to each address you want to remove followed by **Remove email address** and confirm deletion.
4. (Optional) Add or remove **Tags** to your Address List by choosing **Manage tags**.

Using the AWS CLI:

Add

```
aws mailmanager register-member-to-address-list \
                --address-list-id al-123456789abc \
                --address "user@example.com"
```

Remove

```
aws mailmanager deregister-member-from-address-list \
                --address-list-id al-123456789abc \
                --address "user@example.com"
```

Viewing Address List details

To view Address List details:

- On the **Address Lists** page, select the name of an Address List to view its details.

Using the AWS CLI:

```
aws mailmanager list-members-of-address-list --address-list-id al-123456789abc
```

Deleting an Address List

To delete an Address List:

1. On the **Address Lists** page, select the radio button next to the Address List you want to delete followed by **Delete**.
2. Confirm deletion of the list by typing *confirm* followed by **Delete**.

Using the AWS CLI:

```
aws mailmanager delete-address-list --address-list-id al-123456789abc
```

Using Address Lists in Traffic Policies and Rule Sets

Address Lists can be used in traffic policy statements and rule conditions to process emails based on list membership giving control over email flow. The following sections provide an example of Address Lists being used in each a traffic policy and a rule set.

Topics

- Using an Address List in a traffic policy statement
- Using an Address List in a rule

Using an Address List in a traffic policy statement

Address Lists can be selected when you build the condition of a traffic policy statement to either allow or deny email coming into your ingress endpoint.

The following console procedure and its AWS CLI equivalent are showing an example of creating a policy statement that allows messages into your ingress endpoint if the recipient is in the specified Address List.

To use an Address List in a traffic policy statement:

1. Create a new traffic policy or edit an existing one as explained in Creating traffic policies & policy statements (console)
2. In the **Policy statement** container, choose **Allow** for the action to be taken when the statement's conditions are met.
3. Build the statement's condition as follows:
   - Select **Recipient address** for the **Protocol** field.
   - Select **Is in address list** for the **Operator** field.
   - Select the name of your Address List for the **Value** field.
4. While this is just one example, you can add more policy conditions that can be based on a variety of operators with any of your Address Lists.

Using the AWS CLI:

```
aws mailmanager create-traffic-policy \
  --default-action ALLOW \
  --traffic-policy-name "testpolicy" \
  --policy-statements '[{
    "Action": "ALLOW",
    "Conditions": [{
      "BooleanExpression": {
        "Evaluate": {
          "IsInAddressList": {
            "Attribute": "RECIPIENT",
            "AddressLists": [
              "arn:aws:ses:eu-west-3:123456789012:mailmanager-address-
list/al-123456789abc"
            ]
          }
        },
        "Operator": "IS_TRUE"
      }
  }]
}]'
```

Using an Address List in a rule

Address Lists can be selected when you build the condition of a rule used in one of your rule sets to trigger the rule's action.

The following console procedure and its AWS CLI equivalent are showing an example of creating a rule that invokes the drop action if the recipient is in the specified Address List.

To use an Address List in a rule condition:

1. Create a new rule or edit an existing one as explained in Creating rule sets & rules (console)
2. In the **Rule conditions** container, build the rule's condition as follows.
   - Select **Recipient address** for the **Select property** field.
   - Select **Is in address list** for the **Select operator** field.
   - Select the name of your Address List for the **Value** field.
3. In the **Actions** container choose **Add new action** and select **Drop action**.
4. While this is just one example, you can add more rule conditions that can be based on a variety of operators with any of your Address Lists for a variety of actions to be taken.

Using the AWS CLI:

```
aws mailmanager create-rule-set \
  --rule-set-name "testruleset2" \
  --rules '[{
    "Name": "addresslist",
    "Conditions": [{
      "BooleanExpression": {
        "Evaluate": {
          "IsInAddressList": {
            "Attribute": "RECIPIENT",
            "AddressLists": [
              "arn:aws:ses:us-east-1:123456789012:mailmanager-address-
list/al-123456789abc"
            ]
          }
        },
        "Operator": "IS_TRUE"
      }
    }],
    "Actions": [{
      "Drop": {}
    }]
  }]'
```

Best Practices and Considerations

- Be mindful of list sizes—very large lists may impact performance.
- Address Lists are account-specific and can only be used within the same AWS account.
- Nested Address Lists are not currently supported.
- A maximum of 100 Address Lists per region is supported.
- A maximum of 100,000 addresses per Address List is supported.

Email archiving

Email archiving provides a way for you to archive the types of email you specify coming into your ingress endpoint, or emails you send through a configuration set, as well as providing a way to find your archived messages through a rich set of advanced search filters and the ability to export the results.

Email archiving saves and protects your emails by storing data in persistent and secure long-term storage, and gives you a way to quickly search and archive email. It provides full-time, enterprise-level archiving without increasing the storage requirements of your mailbox server.

An archive can contain a combination of both sent and received email:

- **Sent email archiving** – When you send (outbound) email through a configuration set with the archiving option enabled, all email sent using that configuration set will be archived to the email archive you designate.
- **Received email archiving** – When your ingress endpoint receives (inbound) email, it uses a traffic policy to determine which emails to block or allow. The email you allow in passes to a rule set that applies conditional rules to execute the actions you've defined for specific types of email. One of the rule actions you can define is *Archive action*—if you select this action, the email will be archived to the email archive you designate.

You must first create an archive before it can be specified in a rule action or configuration set. The procedure in the next section will walk you through creating an archive in the SES console.

Using email archiving in the Amazon SES console

The **Email archiving** page in the SES console consists of four interactive tables, **Search archive**, **Search history**, **Export history**, and **Manage archives**, that you can use to search for email in your archives, export the results, and manage your archives. In the following procedures, instructions are provided for each table.

To use the Email archiving page to search, export, and manage your archives

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Email archiving** under **Mail Manager**.
3. The **Email archiving** page consists of four tables **Search archive**, **Search history**, **Export history**, and **Manage archives**. For instructions specific to each of these tables, select its corresponding tab below:

Search archive

Search archive is an interactive table that provides a way for you to search and find your archived messages with a rich filter and date set offering detailed search criteria to find

anything from a specific email to many emails matching a broader category. Messages matching your search criteria can be downloaded individually or can be exported in bulk to an S3 bucket.

To search, download, or export archived email

1. On the **Email archiving** page, choose the **Search archive** tab to display the **Search archive** table.
2. Click inside the **Archive** field and choose an archive from the list followed by **Search**, or refine your search using the following steps.
3. Select the **Date range** field to expand date range options for your search:

   - **Relative range** (default) – Select the radio button that corresponds with the number of days desired, or choose a **Custom range** by selecting a time unit and a date range up to 30 days.
   - **Absolute range** – Enter a **Start date** and an **End date** *(and time if desired)* up to 30 days.

   > **Note**
   >
   > - Searching within an archive is limited to 30 days at a time. For example, if you want to search for messages from June 1st through July 31st, you would have to break it up into three searches as follows:
   >
   >   1. 30 days in June.
   >   2. The first 30 days in July.
   >   3. The 31st day of July.
   >
   > - For **Relative range** dates, the last day ends at midnight. For example, if you choose *Last 7 days*, the seventh day would be yesterday, ending at midnight.

4. (Optional) Select the **Filters** field to choose from among the following filters: *From*, *To* , *CC*, *Subject line*, and *Has attachments*—the following properties apply:

   - You can create up to 10 filters.
   - A filter can be edited by clicking on it, or removed by selecting the **X**.

5. Choose **Search** and the archived email matching your search criteria will be populated in the **Search results** table.

- The **Message ID** column is hidden by default, but can be displayed by selecting the gear icon to customize how you view the table.
- Every search you execute is automatically saved with a unique search id and will be listed in the **Search history** table.

6. To view the text of a message along with its envelope and header information, select the message's radio button followed by **View details** to open the **Message details** sidebar.
7. To create a local file of the message, select the message's radio button followed by **Download message**.
8. Your filtered search can be saved to an Amazon S3 bucket by selecting **Export to S3**.

   a. If you know the URI of the S3 bucket you want to use, enter it in the **S3 URI** field; otherwise, choose **Browse S3** and select an S3 bucket and folder to use on the S3 page.

   b. (Optional) You can encrypt your exported messages either by entering your own AWS KMS key into the **KMS key ARN** field, or by selecting **Create new key**. Otherwise, encryption will be set to whatever method is being used on the destination S3 bucket (even if none).

   c. Choose **Export** and all the messages found in your filtered search will be saved as individual files in the S3 folder you selected.

> **Note**
>
> While there's no limit on how many messages your archive can contain, search results are limited to 1000 rows in the **Search results** table.

Search history

A history of your searches is listed in this table so that you can restore the result set or access complex filter sets created previously. You can also create new searches based on the original search by editing the filters and dates. Any new searches are automatically saved with a unique search ID and will be listed in this table.

To view and work with your previous searches

1. On the **Email archiving** page, choose the **Search history** tab to display the **Search history** table which lists a history of all your archived email searches with the most recent on top. *This table loads data the first time you visit it—if you switch tabs and come back, use the refresh icon to retrieve the latest data.*
2. Click inside the **Archive** field and choose an archive from the list—all the searches belonging to that archive will be populated in the table. You can view and do more with individual searches in the steps below.
3. Select the radio button of a previous search followed by **View search results** to restore its original search results—the **Search archive** page will open displaying the filter set and date range used for the original search along with all the messages previously found based on that criteria. You can expand upon the original search in the following ways:
   - Create a new search by modifying the date range and filters followed by **Search**.
   - Any new searches you perform are automatically saved with a unique search ID and will be listed in the **Search history** table.

Export history

A history of your exports is listed in this table enabling easy access to the contents of the export folder in the S3 console.

To view your recent exports

1. On the **Email archiving** page, choose the **Export history** tab to display the **Export history** table which lists all of the archived email searches you exported to an S3 bucket within the last 30 days. *This table loads data the first time you visit it—if you switch tabs and come back, use the refresh icon to retrieve the latest data.*
2. If the status of an export is *Queued*, *Preprocessing* or *Processing*, you can cancel it by choosing **Cancel**.
3. Select an **S3 URI** to open the export's bucket folder in the S3 console where you can see the files it contains.

Manage archives

This table lists your archives where you have options to create a new archive, search for a particular archive and view its details, edit an archive, or delete an archive.

To create and manage archives

1. On the **Email archiving** page, choose the **Manage archives** tab to display the **Archives** table which lists all of your email archives. *This table loads data the first time you visit it—if you switch tabs and come back, use the refresh icon to retrieve the latest data.*
2. To search for a particular archive, begin typing in the **Archives** field.
3. To view details of an archive, select its name in the **Archive name** column.
4. To create an archive, select **Create archive**.

   a. Enter a unique name in the **Archive name** field.

   b. (Optional) Select a retention period in the **Retention period** field to override the default retention period of 180 days.

   c. (Optional) You can encrypt your archive either by entering your own AWS KMS key into the **KMS key ARN** field, or by selecting **Create new key**.

   Choose **Create archive**.
5. Now that you've created an archive, you can use it in a rule set to archive received (inbound) email, or designate it in a configuration set to archive sent (outbound) email.
6. To edit an archive, select its radio button followed by **Edit**.

   a. Edit or change the name in the **Archive name** field.

   b. Change the retention period in the **Retention period** field.

   Choose **Update archive**.
7. To delete an archive, select its radio button followed by **Delete**.

   - Type `delete` in the **Confirm** field followed by **Delete**.

     The archive state will switch to *Pending deletion* in the **Archives** table and will be automatically deleted after 30 days.

Email Add Ons

Email Add Ons is a collection of specialized security tools from SES approved providers that can be used to manage the type of email you allow into your ingress endpoint and to determine actions to be taken on certain types of email. These tools are certified security intelligence and enforcement solutions that are ready to be integrated into your email workflow and can be activated directly from the Mail Manager console.

These Add Ons offer the flexibility to choose from among vetted email security solutions appropriate to your individual use cases that can be used on a metered-price basis, as opposed to purchasing a large, single product solution that may not be optimized for any of your needs. Email Add Ons extends its core threat intelligence and security enforcement features on a per-workload basis, so there's no guessing about required capacity. These benefits allow you to focus on staying ahead of email security issues and maintain high service standards for your organization.

You can learn more about each Add On directly from the Email Add Ons page located in the Mail Manager console where you'll have access to product descriptions, key benefits, and pricing information. Once you decide on an Add On you'd like to use, simply subscribe to it from Mail Manager console. Once subscribed, you'll be able to select it as a traffic policy condition in determining email allowed into an ingress endpoint, or as a rule set condition to determine actions to be taken on specific emails. Primary support for all Add Ons is provided by AWS and can also be accessed from the Mail Manager console.

The procedure in the next section will walk you through subscribing to an Email Add On in the Mail Manager console.

Subscribing to Email Add Ons in the Mail Manager console

The following procedure shows you how to use the **Email Add Ons** page in the Mail Manager console to subscribe to an Add On so that it can be used in any of your traffic policies or rule sets.

To subscribe to an Email Add On using the console

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the left navigation panel, choose **Email Add Ons** under **Mail Manager**.
3. On the **Email Add Ons** page, select the title of any Add On card to open its overview page where you can learn more about what it does, its key benefits, and pricing information. If you'd like to use this Add On, choose **Subscribe**.

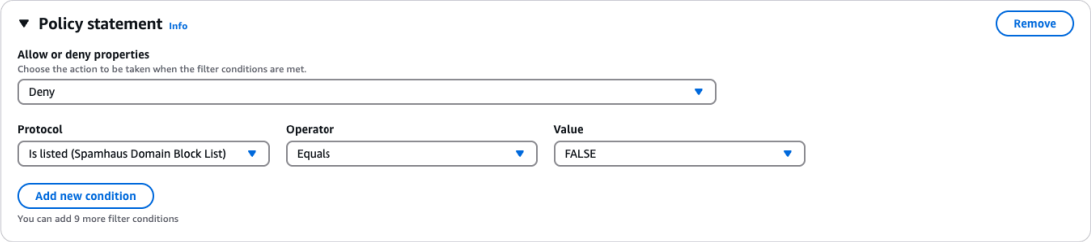
- Read the **Terms and Conditions** presented and check the **I accept** box followed by **Subscribe**.

4. Once you've subscribed to an Add On, you'll be able to integrate it into your email workflow by selecting it as a traffic policy condition to deny or allow email into your ingress endpoint, or a rule set condition to determine an action to take on qualifying messages.

   The following examples depict using an Add On in a policy statement condition and in a rule condition:

   Traffic policy example using an Add On

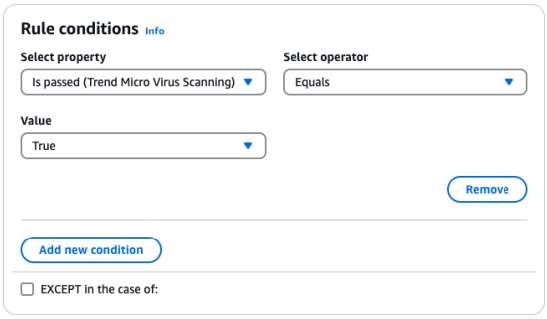
   Using the **Spamhaus Domain Block List** Add On in a policy statement condition to block email coming into your ingress endpoint that originates from a domain listed in Spamhaus:



   For details on how to create traffic policies and build policy statement conditions with Email Add Ons, see the section called "Creating traffic policies & policy statements (console)".

   Rule condition example using an Add On

   Using the **Trend Micro Virus Scanning** Add On in a rule condition to determine a rule action for email that passes the virus scan:



   For details on how to create rule sets and build rule conditions with Email Add Ons, see the section called "Creating rule sets & rules (console)".

5. To view general details or access support for any Add On you're subscribed to, select its name on the **Email Add Ons** page to open its overview page:
   - In **General details** you can view the date of when you subscribed and the Amazon Resource Name (ARN) of your Add On.
   - Select the **Support** tab to access links to AWS Support.
6. To unsubscribe from an Add On:
   a. You must first remove it from any of your traffic policies or rule sets where you have it defined in a condition; otherwise, the following unsubscribe steps will fail.
   b. Select its name on the **Email Add Ons** page to open its overview page followed by **Unsubscribe**.
   c. Type `confirm` in the **Confirm** field followed by **Unsubscribe**.

Permission policies for Mail Manager

The policies in this chapter are provided as a single point of reference for the policies necessary to utilize all the different features of Mail Manager.

In the Mail Manager feature pages, links are provide that will take you to the respective section on this page that contains the policies you need to utilize the feature. Select the copy icon of the policy you need and paste it as directed in the respective feature narrative.

The following policies give you permission to use the different features contained in Amazon SES Mail Manager through resource permission policies and AWS Secrets Manager policies. If you're new to permission policies, see the section called “Policy anatomy” and Permissions policies for AWS Secrets Manager.

Permission policies for Ingress endpoint

Both of the polices in this section are required to create an ingress endpoint. To learn how to create an ingress endpoint and where to use these policies, see the section called “Creating an ingress endpoint (console)”.

Secrets Manager secrets resource permission policy for ingress endpoint

The following Secrets Manager secrets resource permission policy is required to allow SES to access the secret using the ingress endpoint resource.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Id",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "000000000000"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws-eusc:ses:eusc-de-
east-1:000000000000:mailmanager-ingress-point/*"
                }
            }
        }
    ]
}
```

KMS customer managed key (CMK) key policy for ingress endpoint

It is required to use a customer managed key (CMK) for your secret. The following statement is required within your KMS key policy to allow SES to use your key for your secret.

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "ses.amazonaws.com"
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "secretsmanager.us-east-1.amazonaws.com",
```

```
            "aws:SourceAccount": "000000000000"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:ses:us-east-1:000000000000:mailmanager-ingress-
point/*"
        }
    }
}
```

KMS customer managed key (CMK) key policy for mTLS trust store

If you use a customer managed key (CMK) to encrypt your mTLS trust store, the following statement is required within your KMS key policy to allow SES to use your key.

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "ses.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "000000000000"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:ses:us-east-1:000000000000:mailmanager-ingress-
point/*"
        }
    }
}
```

Permission policies for SMTP relay

Both of the polices in this section are required to create an SMTP relay. To learn how to create an SMTP relay and where to use these policies, see the section called “Creating an SMTP relay (console)”.

Secrets Manager secrets resource permission policy for SMTP relay

The following Secrets Manager secrets resource permission policy is required to allow SES to access the secret using the SMTP relay resource.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Principal": {
                "Service": [
                    "ses.amazonaws.com"
                ]
            },
            "Resource": "*",
            "Condition": {
               "StringEquals": {
                   "aws:SourceAccount": "888888888888"
                },
               "ArnLike": {
                   "aws:SourceArn": "arn:aws-eusc:ses:eusc-de-
east-1:888888888888:mailmanager-smtp-relay/*"
                }
           }
        }
    ]
}
```

KMS customer managed key (CMK) key policy for SMTP relay

The following statement is required within your KMS key policy to allow SES to use your key for your secret.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.eusc-de-
east-1.amazonaws.com",
                    "aws:SourceAccount": "000000000000"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws-eusc:ses:eusc-de-
east-1:000000000000:mailmanager-smtp-relay/*"
                }
            }
        }
    ]
}
```

Permission policies for Email archiving

Archiving export

The IAM identity calling `StartArchiveExport` must have access to the destination S3 bucket configured by the following IAM policy:

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws-eusc:s3:::MyDestinationBucketName"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:PutObjectTagging",
                "s3:GetObject"
            ],
            "Resource": "arn:aws-eusc:s3:::MyDestinationBucketName/*"
        }
    ]
}
```

This is the S3 bucket policy for the destination bucket:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws-eusc:s3:::MyDestinationBucketName"
        },
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:PutObjectTagging",
                "s3:GetObject"
            ],
            "Resource": "arn:aws-eusc:s3:::MyDestinationBucketName/*"
        }
    ]
}
```

Note

Archiving doesn't support confused deputy condition keys (aws:SourceArn, aws:SourceAccount, aws:SourceOrgID, or aws:SourceOrgPaths). This is because Mail Manager's email archiving prevents the confused deputy problem by testing if the calling identity has write permissions to the export destination bucket using Forward Access Sessions before starting the actual export.

Archiving encryption at rest with KMS CMK

The IAM identity calling `CreateArchive` and `UpdateArchive` must have access to the KMS key ARN through the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:GenerateDataKey"
        ],
```

```
        "Resource": "arn:aws-eusc:kms:eusc-de-
east-1:111122223333:key/MyKmsKeyArnID"
    }
}
```

The following statements are required within your KMS key policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-
eusc:iam::111122223333:user/MyUserRoleOrGroupName"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "ses.eusc-de-east-1.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

Permission and trust polices to execute rule actions

The SES rules execution role is an AWS Identity and Access Management (IAM) role that grants the rules execution permission to access AWS services and resources. Before you create a rule in a rule set, you must create an IAM role with a policy that allows access to the required AWS resources. SES assumes this role when executing a rule action. For example, you might create a rules execution role that has permission to write an email message to a S3 bucket as a rule action to take when your rule's conditions are met.

Thus, the following trust policy is required *in addition to* the individual permission policies in this section required to execute each specific rule action.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "ses.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
                   "aws:SourceAccount": "888888888888"
          },
          "ArnLike": {
             "aws:SourceArn": "arn:aws-eusc:ses:eusc-de-
east-1:888888888888:mailmanager-rule-set/*"
          }
        }
      }
    ]
  }
```

Rule action policies

- Permission policy for Write to S3 rule action
- Permission policy for Deliver to mailbox rule action
- Permission policy for Send to internet rule action
- Permission policy for Deliver to Q Business rule action
- Permission policy for Publish to SNS rule action
- Permission policy for Bounce rule action
- Permission policy for Invoke Lambda function rule action

Permission policy for *Write to S3* rule action

The following policy is required for your IAM role to use the **Write to S3** rule action which delivers the received email to an S3 bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPutObject",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::MyDestinationBucketName/*"
            ]
        },
        {
            "Sid": "AllowListBucket",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::MyDestinationBucketName"
            ]
        }
```

```
    ]
}
```

If you're using AWS KMS customer managed key for an S3 bucket with server-side encryption enabled, then you'll need to add the IAM role policy action, `"kms:GenerateDataKey*"`. Using the preceding example, adding this action to your role policy would appear as follows:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowKMSKeyAccess",
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey*",
            "Resource": "arn:aws-eusc:kms:eusc-de-east-1:888888888888:key/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "kms:ResourceAliases": [
                        "alias/MyKeyAlias"
                    ]
                }
            }
        }
    ]
}
```

For more information about attaching policies to AWS KMS keys, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Permission policy for *Deliver to mailbox* rule action

The following policy is required for your IAM role to use the **Deliver to mailbox** rule action which delivers the received email to an Amazon WorkMail account.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["workmail:DeliverToMailbox"],
        "Resource": "arn:aws-eusc:workmail:eusc-de-
east-1:888888888888:organization/MyWorkMailOrganizationID>"
      }
    ]
  }
```

Permission policy for *Send to internet* rule action

The following policy is required for your IAM role to use the **Send to internet** rule action which sends the received email to an external domain.

> **Note**
>
> If your SES identity is using a default configuration set, you'll need to also add the configuration set resource as shown in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["ses:SendEmail", "ses:SendRawEmail"],
        "Resource":[
        "arn:aws-eusc:ses:eusc-de-east-1:888888888888:identity/example.com",
        "arn:aws-eusc:ses:eusc-de-east-1:888888888888:configuration-set/my-
configuration-set"
        ]
      }
    ]
  }
```

Permission policy for *Deliver to Q Business* rule action

The following policies are required to use the **Deliver to Q Business** rule action, which delivers the received email to an Amazon Q Business index.

IAM policy required for your role:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToQBusiness",
            "Effect": "Allow",
            "Action": [
                "qbusiness:BatchPutDocument"
            ],
            "Resource": [
                "arn:aws-eusc:qbusiness:eusc-de-
east-1:888888888888:application/ApplicationID/index/IndexID"
            ]
        }
    ]
}
```

Statement required within your KMS key policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToKMSKeyForQbusiness",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:DescribeKey"
```

```
            ],
            "Resource": [
                "arn:aws-eusc:kms:eusc-de-east-1:888888888888:key/*"
            ],
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "qbusiness.eusc-de-east-1.amazonaws.com",
                    "kms:CallerAccount": "888888888888"
                 },
                 "ForAnyValue:StringEquals": {
                    "kms:ResourceAliases": [
                        "alias/MyKeyAlias"
                    ]
                }
            }
        }
    ]
}
```

For more information about attaching policies to AWS KMS keys, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Permission policy for *Publish to SNS* rule action

The following policies are required to use the **Publish to SNS** rule action, which delivers the received email to an Amazon SNS topic.

IAM policy required for your role:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToSNSTopic",
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws-eusc:sns:eusc-de-east-1:888888888888:MySnsTopic"
```

```
            ]
        }
    ]
}
```

Statement required within your KMS key policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToKMSKeyForSNS",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:DescribeKey"
            ],
            "Resource": [
                "arn:aws-eusc:kms:eusc-de-east-1:888888888888:key/*"
            ],
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "qbusiness.eusc-de-east-1.amazonaws.com",
                    "kms:CallerAccount": "888888888888"
                 },
                 "ForAnyValue:StringEquals": {
                    "kms:ResourceAliases": [
                        "alias/MyKeyAlias"
                    ]
                }
            }
        }
    ]
}
```

For more information about attaching policies to AWS KMS keys, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Permission policy for *Bounce* rule action

The following policy is required for your IAM role to use the **Bounce** rule action which bounces the email by returning a bounce response to the sender.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSendBounce",
            "Effect": "Allow",
            "Action": [
                "ses:SendBounce"
            ],
            "Resource": [
                "arn:aws-eusc:ses:eusc-de-east-1:123456789012:identity/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ses:FromAddress": "sender@example.com"
                }
            }
        }
    ]
}
```

Permission policy for *Invoke Lambda function* rule action

The following policy is required for your IAM role to use the **Invoke Lambda function** rule action which invokes an AWS Lambda function to process the email.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInvokeLambdaFunction",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws-eusc:lambda:eusc-de-east-1:123456789012:function:MyFunction"
```

```
            ]
        }
    ]
}
```

Mail Manager logging

Mail Manager logging provides detailed visibility into your Mail Manager operations. The logging functionality tracks message flow from initial receipt at ingress endpoints through message processing based on your configured rule sets and rules.

Mail Manager offers logging for the following resources:

- Ingress endpoints
- Rule sets

Mail Manager delivers logs using the Amazon CloudWatch Logs service and the logs can be delivered to any of the following destinations: *CloudWatch Logs*, *Amazon S3*, or *Amazon Data Firehose*.

Setting up Mail Manager log delivery

A working log delivery consists of three elements:

- **DeliverySource** – A logical object that represents the resource that sends the logs—either an ingress endpoint or a rule set.
- **DeliveryDestination** – A logical object that represents the actual delivery destination (CloudWatch Logs, S3, or Firehose).
- **Delivery** – Connects a delivery source to a delivery destination.

This section will explain how to create these objects along with the necessary permissions required to use Mail Manager logging.

Prerequisites

Before setting up Mail Manager logging, ensure that:

1. You have created either an Ingress endpoint or a Rule set.

2. You have the necessary CloudWatch Logs and SES Mail Manager permissions to vend logs from your Mail Manager resources to their delivery destinations.

Required permissions

You'll need to setup the vended logs permissions as explained in the Logging that requires additional permissions [V2] section of the *Amazon CloudWatch Logs User Guide* and apply the permissions that correspond to your delivery destination:

- Logs sent to CloudWatch Logs
- Logs sent to Amazon S3
- Logs sent to Firehose

In addition, Mail Manager requires the following user permissions to configure log delivery:

- `ses:AllowVendedLogDeliveryForResource` – Required to allow Mail Manager to vend the logs on your behalf to CloudWatch Logs for your specific resources as shown in the example:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSesMailManagerLogDelivery",
            "Effect": "Allow",
            "Action": [
                "ses:AllowVendedLogDeliveryForResource"
            ],
            "Resource": [
            "arn:aws-eusc:ses:eusc-de-east-1:111122223333:mailmanager-ingress-
point/inp-xxxxx",
            "arn:aws-eusc:ses:eusc-de-east-1:111122223333:mailmanager-rule-set/
rs-xxxx"
            ]
        }
    ]
}
```

Enabling logging in the SES console

To enable logging for Mail Manager resources using the console:

1. Open the SES console at https://console.aws.amazon.com/ses/.
2. In the navigation pane under **Mail Manager**, choose either **Ingress endpoints** or **Rule sets** and select the specific resource you wish to enable for logging.
3. On the details page of the resource, expand **Add log delivery** and choose delivery to either **CloudWatch Logs**, **S3**, or **Firehose**.
4. In the **Add delivery to** dialogue box for the destination you chose, follow the prompts to configure log delivery options specific to the destination type.
5. (Optional) Expand **Additional settings** to customize fields of the record, output format, field delimiter, and other parameters specific to the destination type.

Enabling logging using the CloudWatch Logs API

To enable logging for Mail Manager resources using the CloudWatch Logs API, you will need to:

1. Create a DeliverySource with `PutDeliverySource`.
2. Create a DeliveryDestination with `PutDeliveryDestination`.
3. Create a Delivery by pairing exactly one delivery source and one delivery destination by using `CreateDelivery`.

You can view examples of IAM role and permissions policies with all the required permissions for your specific logging destination in the Logging that requires additional permissions [V2] section of the *Amazon CloudWatch Logs User Guide*, and follow the IAM role and permissions policy examples for your logging destination, including allowing updates to your specific logging destination resource, such as *CloudWatch Logs*, *S3*, or *Firehose*.

> **Note**
>
> When creating a DeliverySource, the `resourceArn` can be an Ingress endpoint ARN or a Rule set ARN. Depending on the DeliverySource, the `logType` can be as follows:
>
> - **Ingress endpoint ARN** – `APPLICATION_LOGS` or `TRAFFIC_POLICY_DEBUG_LOGS`
> - **Rule set ARN** – `APPLICATION_LOGS`

Interpreting the logs

The logs can be used to gain additional insight into the flow of your received messages as they are processed by Mail Manager.

The following examples detail the different fields of the logs for each resource and log type:

Log examples

- Ingress endpoint logs – APPLICATION_LOGS
- Ingress endpoint logs – TRAFFIC_POLICY_DEBUG_LOGS
- Rule set logs – APPLICATION_LOGS

Ingress endpoint logs – `APPLICATION_LOGS`

The logs are generated per message.

```
{
  "resource_arn": "arn:aws:ses:us-east-1:1234567890:mailmanager-ingress-point/inp-
xxxxx",
  "event_timestamp": 1728562395042,
  "ingress_point_type": "OPEN" | "AUTH" | "MTLS",
  "ingress_point_name": "MyIngressPoint",
  "message_id": "0000llcki1jmushh817gr586f963a5inhkvnh81",
  "message_size_bytes": 100000,
  "rule_set_id": "rs-xxxx",
  "sender_ip_address": "1.2.3.4",
  "smtp_mail_from": "someone@domain.com",
  "smtp_helo": "domain.com",
  "tls_protocol": "TLSv1.2",
  "tls_cipher_suite": "TLS_AES_256_GCM_SHA384",
  "recipients": ["me@mydomain.com", "you@mydomain.com", "they@mydomain.com"],
  "ingress_point_metadata": {
       // Only applies to AUTH Ingress endpoint
       "password_version": "",
       "secrets_manager_arn": "",
       // Only applies to MTLS Ingress endpoint
       "client_certificate_details": {
           "common_names": ["mail.example.com"],
           "serial_number": "0A:DE:EB:89:42:FB:1C:67",
           "subject_alternative_names": ["mail.example.com", "smtp.example.com"],
```

```
            "issuer": "CN=Example CA,O=Example Corp,C=US",
            "not_before": "2025-01-15T00:00:00Z",
            "not_after": "2026-01-15T23:59:59Z"
        },
        "trust_store_monitoring": {
            "ca_invalid_or_near_expiry": [
                {
                    "subject": "CN=Example CA,O=Example Corp,C=US",
                    "not_before": "2023-06-01T00:00:00Z",
                    "not_after": "2025-05-15T23:59:59Z"
                },
                ...
            ],
            "crl_invalid_or_near_expiry": [
                {
                    "issuer": "CN=Example CA,O=Example Corp,C=US",
                    "this_update": "2025-03-01T00:00:00Z",
                    "next_update": "2025-04-01T00:00:00Z"
                },
                ...
            ]
        }
    }
}
```

Note

Logs are created only for messages that are accepted by the ingress endpoint. An ingress endpoint that rejects all the incoming messages will not publish any application logs.

Note

The `trust_store_monitoring` lists (`ca_invalid_or_near_expiry` and `crl_invalid_or_near_expiry`) each return a maximum of 10 entries. “Near expiry” means the certificate or CRL expires within 90 days.

Example CloudWatch Logs Insights queries

Query messages from sender@domain.com:

```
fields @timestamp, @message, @logStream, @log
| filter smtp_mail_from like /sender@domain.com/
| sort @timestamp desc
| limit 10000
```

Query messages with size greater than 5000 bytes:

```
fields @timestamp, @message, @logStream, @log
| filter message_size_bytes > 5000
| sort @timestamp desc
| limit 10000
```

Ingress endpoint logs – TRAFFIC_POLICY_DEBUG_LOGS

The logs are generated per recipient.

```
{
    "resource_arn": "arn:aws:ses:us-east-1:1234567890:mailmanager-ingress-point/inp-
xxxxx",
    "event_timestamp": 1728562395042,
    "ingress_point_type": "OPEN" | "AUTH",
    "ingress_point_id": "inp-xxxx",
    "ingress_point_session_id": "xxxx",
    "traffic_policy_id": "tp-xxxx",
    "traffic_policy_evaluation": [
           // Array of policy evaluations
           {
                "action": "ALLOW" | "DENY",
                "conditions": [
                // Array of conditions
                {
                    "expression": {
                        "attribute": "RECIPIENT",
                        "operator": "CONTAINS",
                        "value": ["@domain.com", "@mydomain.com"]
                    },
                    "expressionResult": true | false
                }],
                "policyStatementMatched": true | false
            },
            // If no policy statement match then default action will be applied
            {
```

```
                "action": "ALLOW" | "DENY",
                "policyStatementMatched": true,
                "type": "DefaultAction",
                "scope": "Recipient"
            },
            // Only present if the email was too large according to the traffic policy
            {
                "action": "DENY",
                "allowedMessageSize": 41943040,
                "receivedMessageSize": 42495384,
                "scope": "Data"
            }
    ],
    "traffic_policy_verdict": "REJECT" | "ACCEPT",
    "sender_ip_address": "1.2.3.4",
    "smtp_mail_from": "someone@domain.com",
    "smtp_helo": "domain.com",
    "tls_protocol": "TLSv1.2",
    "recipient": "me@mydomain.com",
    "tls_cipher_suite": "TLS_AES_256_GCM_SHA384"
}
```

Note

- Logs are created for all messages that are evaluated by the traffic policy at the ingress endpoint regardless of whether they're accepted or rejected.
- All recipient traffic policy evaluations belonging to the same message (within the same SMTP conversation) share a common `ingress_point_session_id`. This ID serves as a correlation identifier since the `message_id` isn't available until after message acceptance.
- The `traffic_policy_evaluation` content varies based on your configuration and may terminate early once a verdict is determined.

Example CloudWatch Logs Insights queries

Query messages from sender@domain.com:

```
fields @timestamp, @message, @logStream, @log
| filter smtp_mail_from like /sender@domain.com/
```

```
| sort @timestamp desc
| limit 10000
```

Query messages belonging to a specific `ingress_point_session_id`:

```
fields @timestamp, @message, @logStream, @log
| filter ingress_point_session_id = 'xxx'
| sort @timestamp desc
| limit 10000
```

Query messages that were rejected:

```
fields @timestamp, @message, @logStream, @log
| filter traffic_policy_verdict = 'REJECT'
| sort @timestamp desc
| limit 10000
```

Rule set logs – `APPLICATION_LOGS`

The logs are generated per message per action. This means that a log record is generated each time a message is processed by an action in a rule in the rule set:

```
{
   "resource_arn": "arn:aws:ses:us-east-1:1234567890:mailmanager-rule-set/rs-xxxx",
   "event_timestamp": 1732298258254,
   "message_id": "0000llcki1jmushh817gr586f963a5inhkvnh81",
   "rule_set_name": "MyRuleSet",
   "rule_name": "MyRule",
   "rule_index": 1,
   "recipients_matched": ["recipient1@domain.com", "recipient2@domain.com"],
   "action_metadata": {
       "action_name": "WRITE_TO_S3" | "DROP" | "RELAY" | "DELIVER_TO_MAILBOX" | etc.,
       "action_index": 2,
       "action_status": "SUCCESS" | "FAILURE" | "IN_PROGRESS",
       "action_failure": "Access denied"
   }
}
```

- `recipients_matched` – The recipients that were matched by the conditions of the rule for which the action is being performed.
- `rule_index` – The rule's order within the rule set.

- `action_index` – The action's order within the rule.
- `action_status` – Indicates the outcome of performing the action on the given message.
- `action_failure` – Indicates the failure details of the action (only applies when an action fails). For instance, if the provided role does not have enough permissions to perform the action.

In addition, if the rule conditions did not match for a message, that is, the message is not processed by the rule, a single log is published to indicated that the message has been processed by the rule set, but did not have any actions performed on it:

```
{
   "resource_arn": "arn:aws:ses:us-east-1:1234567890:mailmanager-rule-set/rs-xxxx",
   "event_timestamp": 1732298258254,
   "message_id": "0000llcki1jmushh817gr586f963a5inhkvnh81",
   "rule_set_name": "MyRuleSet",
   "rule_name": "MyRule",
   "rule_index": 1,
   "recipients_matched": [],
}
```

Example CloudWatch Logs Insights queries

Query for a specific message-id (shows the message flow through the rule set):

```
fields @timestamp, @message, @logStream, @log
| filter message_id = 'message-id-123'
| sort @timestamp desc
| limit 10000
```

Query for failed WRITE_TO_S3 actions:

```
fields @timestamp, @message, @logStream, @log
| filter action_metadata.action_name = 'WRITE_TO_S3'
    and action_metadata.action_status = 'FAILURE'
| sort @timestamp desc
| limit 10000
```

Query for messages that did not get processed by the second rule of a rule set (the message did not meet the rule's conditions):

```
fields @timestamp, @message, @logStream, @log
```

```
| filter recipients_matched = '[]'
    and rule_index = 2
| sort @timestamp desc
| limit 10000
```

Managing lists and subscriptions in Amazon Simple Email Service

You can manage your own lists for mailing and subscriptions as well as for email suppression in Amazon SES. To help you maintain your sender reputation, SES offers account-level, tenant-level, and configuration set-level suppression that prevents you from sending to invalid recipients and harming your sender reputation. If you use multi-tenancy, tenant-level suppression lists give each tenant its own isolated suppression list, preventing cross-tenant contamination. As another measure against bounced emails and complaints, SES can automatically add unsubscribe links to all outgoing mail through subscription management.

Each of these types of lists is discussed in detail in the sections listed in this chapter's topics; however, an overview of suppression lists is presented here to understand how they differ as well as a key change with global suppression list management. It's suggested that you read this overview before working with any of the lists discussed in this chapter.

Overview of suppression lists and suppression override mechanism

The global suppression list removal feature is no longer customer facing and you no longer interact with it to manage suppression. The global suppression list operates and is managed in the background by SES. As a customer, you now have available to you an account-level suppression list and configuration set-level suppression overrides that offer you more customized control over how you handle email suppression for your own account.

The different types of suppression lists, their scope, and what advantages they offer is explained below.

- **Global suppression list** – Owned and managed by SES to protect the reputation of addresses in the SES shared IP pool.
- **Account-level suppression list** – Owned and managed by the customer to protect their account reputation - *overrides the global suppression list*.
  - **Configuration set-level suppression** – An override mechanism to provide conditional or fine-grained control of the account-level suppression list through the use of overrides specified in a configuration set.

The global suppression list was the only type of suppression list until account-level and configuration set-level suppression was introduced in the new Amazon SES console and API v2.

The global suppression list is owned and managed by SES to protect the reputation of SES. This is needed because all SES customers are sharing the same pool of IP addresses (unless they have dedicated IPs), it's important for SES to ensure that customers aren't sending spam or anything that would negatively impact the reputation of those IP addresses in the SES shared IP pool. While you no longer directly interact with the global suppression list, it still operates in the background and the general tenets of how the global suppression list works can also be applied to explain the overall principles of how the other types of suppression work. See Amazon SES global suppression list.

> **Note**
>
> The global suppression list removal request form is no longer in the Amazon SES console because the account-level suppression list has superseded it for all the advantages explained in this section.

The account-level suppression list was introduced so that customers can create and control their own suppression list and reputation, thus, the account-level suppression list applies to your account only. The account-level suppression list interface in the new console provides an easy way to manage addresses in your account-level suppression list, including bulk actions to add or remove addresses. If an address is on the global suppression list, but not on your account level suppression list *(which means you want to send to it)*, and you do send to it, Amazon SES will still attempt delivery, but if it bounces, the bounce will affect your own reputation, but no one else will get bounces because they can't send to that email address if they aren't using their own account level suppression list; therefore, the account-level suppression list overrides the global suppression list for your account only. See Using the Amazon SES account-level suppression list.

Configuration set-level suppression, while not a list per se, but a mechanism that enables you to configure suppression customizations and overrides to your account-level suppression list through the use of configuration sets specifically created for different email sending scenarios. For example, if your account-level suppression list is configured for both bounce and complaint addresses to be added, but you have a particular email demographic defined in a configuration set for which you're only interested in complaint addresses being added - you would achieve this by enabling this configuration set's suppression overrides so that email addresses are added to your account-level suppression list only for complaints (not bounces and complaints like is set in your account-level suppression list) from email sent with this configuration set. With configuration set-level suppression, there are different levels of overriding your account-level suppression, including

not using any suppression at all. See Using configuration set-level suppression to override your account-level suppression list.

Amazon SES global suppression list

Amazon SES maintains an internal *global suppression list* which operates and is managed in the background by SES. When any SES customer sends an email that results in a hard bounce, SES adds the email address that produced the bounce to a global suppression list. The global suppression list is *global* in the sense that it applies to all SES customers. In other words, if a different customer attempts to send an email to an address that's on the global suppression list, SES accepts the message, but doesn't send it, because the email address is suppressed.

The global suppression list email address removal request feature *is no longer customer facing and you no longer interact with it* to manage suppression. To replace this functionality, Amazon SES now offers a new way for you to manage your suppression by making available an **account-level suppression list** and **configuration set-level suppression overrides** that offer you more customized control over how you handle email suppression for your own account. For more information, see Using the Amazon SES account-level suppression list and Using configuration set-level suppression to override your account-level suppression list.

> **Important**
>
> The global suppression list email address removal request form is no longer in the Amazon SES console because the account-level suppression list has superseded it. To learn how to use the account-level suppression list, see Using the Amazon SES account-level suppression list.

Global suppression list considerations

Key factors regarding the global suppression list:

- The global suppression list operates and is managed in the background by SES - you cannot interact with it directly; however, you can override it by using your own account-level suppression list.
- The global suppression list is enabled by default for all SES accounts. You can't disable it.
- Because SES applies the global suppression list to all customers, you can't query the global suppression list or add addresses to it manually.

- When an email address produces a hard bounce, SES adds the address to the global suppression list for a short period of time. After that period of time elapses, SES removes the address from the list. If the address produces another hard bounce, SES adds it back to the global suppression list for a longer period of time, and removes it at the end of that period. The amount of time that an address remains on the global suppression list increases each time the address produces a hard bounce. An address can remain on the global suppression list for up to 14 days.
- If you attempt to send a message to an address that's on the global suppression list, SES accepts the message, but doesn't send it. SES generates a bounce notification with a `bounceType` value of `Permanent`, and a `bounceSubType` value of `Suppressed`. Receiving this type of bounce notification is the only way to know if an address is on the global suppression list. You can't query the global suppression list.
- SES counts the messages that you send to addresses on the global suppression list toward the bounce rate for your account and toward your daily sending quota.
- As with any email address that produces a hard bounce, you should remove addresses that cause a suppression list bounce from your mailing list unless you're certain that the address is valid.
- Suppression list bounces count towards your account's bounce rate. If your bounce rate gets too high, your account might be placed under review or your account's ability to send email could be paused.

> **Note**
>
> It's important to understand how the SES suppression lists are interrelated and their hierarchy, see Overview of suppression lists and suppression override mechanism.

Using the Amazon SES account-level suppression list

The Amazon SES account-level suppression list was introduced so that customers can create and control their own suppression list and manage their reputation, thus, your account-level suppression list applies to your account only. The account-level suppression list interface in the SES console provides an easy way to manage addresses in your account-level suppression list, including bulk actions to add or remove addresses.

Your SES *account-level suppression list* applies to your AWS account in the current AWS Region. You can add or remove, individually or in bulk, addresses from your account-level suppression list by using the SES API v2 or console.

Note

If you use multi-tenancy, you can also configure tenant-level suppression lists for per-tenant isolation. When tenant-level suppression is enabled for a tenant, SES uses the tenant's suppression list instead of the account-level suppression list. For more information, see Using tenant-level suppression lists in Amazon SES.

Note

To bulk add or remove addresses, you must have production access. To learn more about the sandbox, see Request production access (Moving out of the Amazon SES sandbox).

Important

SES account suppression lists are case-sensitive. When an email address is added to the suppression list, *User@Example.com*, it is stored exactly as received, preserving the original case. While email sending functionality treats addresses with different cases as identical, for example, *User@Example.com* is the same as *user@example.com*, API calls for suppression list management require an exact case match. Thus, when using APIs related to suppression list management, ensure you use the exact case of the email address as it appears in the suppression list.

Amazon SES Account-level suppression list considerations

You should consider the following factors when you use your account-level suppression list:

- If you started using Amazon SES after November 25, 2019, your account uses the account-level suppression list by default for both bounces and complaints. If you started using SES before this date, then you have to enable this feature by using the `PutAccountSuppressionAttributes` operation in the SES API.
- If you attempt to send a message to an address that's on your account-level suppression list that has a suppression reason that matches the same suppression reason chosen for your account-level suppression *settings*, SES accepts the message, but doesn't send it—however, if they don't match, then SES *will* send it. To help clarify this, the following examples are provided:

  - You've set your account-level suppression settings with the suppression reason of *Bounces only*, SES will not attempt delivery for addresses in your account-level suppression list with the suppression reason as *Bounce*. However, SES *will* attempt delivery for addresses in your account-level suppression list with the suppression reason of *Complaint* (because in this case, they do not match).
  - You've set your account-level suppression settings with the suppression reason of *Bounces and Complaints*, SES will not attempt delivery for addresses in your account-level suppression list with a suppression reason of either *Bounce* or *Complaint*.
- SES doesn't count the messages that you send to addresses on your account-level suppression list towards the *Reputation.BounceRate* or *Reputation.ComplaintRate* metrics in the AWS/SES namespace for your account. Such messages are counted under *Bounce* or *Complaint* metrics in the AWS/SES namespace.
- If an address is on the global suppression list, but not on your account level suppression list *(which means you want to send to it)*, and you do send to it, SES will still attempt delivery; however, if it bounces, it still counts toward the bounce rate for your account and toward your daily sending quota.
- SES counts the messages that you send to addresses on your account-level suppression list toward your daily sending quota.
- Email addresses on your account-level suppression list remain there until you remove them.
- If your account's ability to send email is paused, SES automatically deletes the addresses in your account-level suppression list after 90 days. If your account's ability to send email is restored before this 90-day period ends, then the addresses in the list aren't deleted.
- Gmail doesn't provide complaint data to SES. If a recipient uses the **Spam** button in the Gmail web client to report a message that they receive from you as spam, they aren't added to your account-level suppression list.
- You can enable your account-level suppression list if your account is in the SES sandbox. However, you can't use the PutSuppressedDestination or CreateImportJob operation until your account is removed from the sandbox. To learn more about the sandbox, see Request production access (Moving out of the Amazon SES sandbox).
- Only hard bounces are added to your account-level suppression list. For information about the differences between soft and hard bounces, see the section called “After Amazon SES sends an email”.
- When you use your account-level suppression list, SES adds addresses that result in hard bounces to the global suppression list as well.

> **Note**
>
> The procedures in the following sections assume that you've already installed the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

Enabling the Amazon SES account-level suppression list

You can use the PutAccountSuppressionAttributes operation in the Amazon SES API v2 to enable and set up your account-level suppression list. You can quickly and easily configure this setting by using the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To configure your account-level suppression list using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 put-account-suppression-attributes \
  --suppressed-reasons BOUNCE COMPLAINT
  ```

  Windows

  ```
  aws sesv2 put-account-suppression-attributes `
  --suppressed-reasons BOUNCE COMPLAINT
  ```

  To enable your account-level suppression list, you have to specify at least one reason for the `suppressed-reasons` parameter. You can specify either `BOUNCE` or `COMPLAINT`, or you can specify both, as shown in the preceding example.

To configure your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.

3. In the **Account-level settings** pane, choose **Edit**.
4. In **Suppression list**, check the **Enabled** box.
5. In **Suppression reasons**, select one of the reasons for which recipient email addresses should be automatically added to your account-level suppression list.
6. Choose **Save changes**.

Enabling the Amazon SES account-level suppression list for a configuration set

You can also configure your Amazon SES account-level suppression so that it only applies to specific configuration sets. When you do, addresses are only added to the suppression list if you specified the configuration set when you sent the email that caused the bounce or complaint event.

To configure your account-level suppression list for a configuration set using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 put-configuration-set-suppression-options \
  --configuration-set-name configSet \
  --suppressed-reasons BOUNCE COMPLAINT
  ```

  Windows

  ```
  aws sesv2 put-configuration-set-suppression-options `
  --configuration-set-name configSet `
  --suppressed-reasons BOUNCE COMPLAINT
  ```

  In the preceding example, replace *configSet* with the name of the configuration set that should use your account-level suppression list.

To configure your account-level suppression list for a configuration set using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. In **Configuration sets**, choose the name of the configuration set you want to configure with customized suppression.
4. In the **Suppression list options** pane, choose **Edit**.
5. The **Suppression list options** section provides a decision set to define customized suppression starting with the option to use this configuration set to override your account-level suppression. The configuration set-level suppression logic map will help you understand the effects of the override combinations. These multitiered selections of overrides can be combined to implement three different levels of suppression:

   a. **Use account-level suppression:** Do not override your account-level suppression and do not implement any configuration set-level suppression - basically, any email sent using this configuration set will just use your account-level suppression. To do this:

      - In **Suppression list settings**, uncheck the **Override account level settings** box.

   b. **Do not use any suppression:** Override your account-level suppression without enabling any configuration set-level suppression - this means any email sent using this configuration set will not use any of your account-level suppression; in other words, all suppression is cancelled. To do this:

      i. In **Suppression list settings**, check the **Override account level settings** box.

      ii. In **Suppression list**, uncheck the **Enabled** box.

   c. **Use configuration set-level suppression:** Override your account-level suppression with custom suppression list settings defined in this configuration set - this means any email sent using this configuration set will only use its own suppression settings and ignore any account-level suppression settings. To do this:

      i. In **Suppression list settings**, check the **Override account level settings** box.

      ii. In **Suppression list**, check **Enabled**.

      iii. In **Specify the reason(s)...**, select one of the suppression reasons for this configuration set to use.

6. Choose **Save changes**.

Adding individual email addresses to the Amazon SES account-level suppression list

You can add individual addresses to your Amazon SES account-level suppression list by using the PutSuppressedDestination operation in the SES API v2. There's no limit to the number of addresses that you can add to your account-level suppression list.

To add individual addresses to your account-level suppression list using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 put-suppressed-destination \
  --email-address recipient@example.com \
  --reason BOUNCE
  ```

  Windows

  ```
  aws sesv2 put-suppressed-destination `
  --email-address recipient@example.com `
  --reason BOUNCE
  ```

  In the preceding example, replace *recipient@example.com* with the email address that you want to add to your account-level suppression list, and *BOUNCE* with the reason that you're adding the address to the suppression list (acceptable values are `BOUNCE` and `COMPLAINT`).

To add individual addresses to your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Suppression list** pane, choose **Add email address**.

4. Type an email address in the **Email address** field followed by selecting a reason in **Suppression reason** - if you need to enter more addresses, choose **Enter another address** and repeat for each additional one.
5. When done entering addresses, review your entries for accuracy. If you decide any of your entries shouldn't be part of this submission, choose its **Remove** button.
6. Choose **Save changes** to add the entered email addresses to your account-level suppression list.

Adding email addresses in bulk to your Amazon SES account-level suppression list

You can add addresses in bulk by first uploading your contact list into an Amazon S3 object followed by using the CreateImportJob operation in the Amazon SES API v2.

> **Note**
>
> - There's no limit to the number of addresses that you can add to your account-level suppression list, but there is a bulk add limit of 100,000 addresses in an Amazon S3 object per API call.
> - You cannot run more than 20 concurrent import jobs.
> - If your data source is an S3 bucket, it must exist in the same region as you're importing into.

To add email addresses in bulk to your account-level suppression list, complete the following steps.

- Upload your address list into an Amazon S3 object in either CSV or JSON format.

  CSV format example for adding addresses:

  ```
  recipient1@example.com,BOUNCE

  recipient2@example.com,COMPLAINT
  ```

  Only newline-delimited JSON files are supported. In this format, each line is a complete JSON object that contains an individual address definition.

JSON format example for adding addresses:

```
{"emailAddress":"recipient1@example.com","reason":"BOUNCE"}

{"emailAddress":"recipient2@example.com","reason":"COMPLAINT"}
```

In the preceding examples, replace *recipient1@example.com* and *recipient2@example.com* with the email addresses that you want to add to your account-level suppression list. The acceptable reasons that you're adding the addresses to the suppression list are *BOUNCE* and *COMPLAINT*.

- Give SES permission to use your AWS KMS key.

  If the Amazon S3 object is encrypted with an AWS KMS key, you need to give Amazon SES permission to use the AWS KMS key. SES can only attain permission from a customer managed key, not a default KMS key. You need to give SES permission to use the customer managed key by adding a statement to the key's policy.

  Paste the following policy statement into the key policy to permit SES to use your customer managed key.

  ```
  {
     "Sid": "AllowSESToDecrypt",
     "Effect": "Allow",
     "Principal": {
         "Service":"ses.amazonaws.com"
     },
     "Action": [
         "kms:Decrypt",
     ],
     "Resource": "*"
  }
  ```

- Use the CreateImportJob operation in the SES API v2.

> **Note**
>
> The following example assumes that you've already installed the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

At the command line, enter the following command. Replace *s3bucket* with the name of an Amazon S3 bucket and *s3object* with the name of an Amazon S3 object.

```
aws sesv2 create-import-job --import-destination
 SuppressionListDestination={SuppressionListImportAction=PUT} --import-data-source
 S3Url=s3://s3bucket/s3object,DataFormat=CSV
```

To add email addresses in bulk to your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Suppression list** table, expand the **Bulk actions** button and select **Add email addresses in bulk**.
4. In **Bulk action specifications**, select either (a)**Choose file from S3 bucket** or (b)**Import from file** - procedures are given for each import method:

    a. **Choose file from S3 bucket** - *if your source file is already stored in an Amazon S3 bucket*:

        i. If you know the URI of the Amazon S3 bucket you want to use, enter it in the **Amazon S3 URI** field; otherwise, choose **Browse S3**:

            A. In **Buckets**, select the name of the S3 bucket.

            B. In **Objects**, select the name of the file then select **Choose** - you'll be returned to **Bulk action specifications**.

            C. (Optional) If you want to be taken to the Amazon S3 console to view details about your S3 object choose **View**.

        ii. In **File format**, select the format of the file you've chosen to import from you Amazon S3 bucket.

iii. Choose **Add email addresses** to kick off the import of addresses from your file - a table under the **Bulk actions** tab is displayed.

b. **Import from file** - *if you have a local source file to upload to a new or existing Amazon S3 bucket*:

i. In **Import source file**, select **Choose file**.

ii. Select the JSON or CSV file in the file browser and choose **Open** - you'll see the name, size, and date of your file displayed under the **Choose file** button.

iii. Expand **Amazon S3 bucket** and select the S3 bucket.

- To upload your file to a new bucket, choose **Create S3 bucket**, enter a name in the **Bucket name** field, and choose **Create bucket**.

iv. Choose **Add email addresses** to kick off the import of addresses from your file - a table under the **Bulk actions** tab is displayed.

5. Regardless of the import method you used, your job ID will be listed in **Bulk actions** along with import type, status, and date - to view job details, select the job ID.

6. Select the **Suppression list** tab and all the successfully imported email addresses are displayed with their suppression reason and date added - the following options are available:

a. Select an email address, or select its corresponding checkbox and choose **View report** to view its details. (If it's an address that was automatically added to your suppression list because of a bounce or complaint, information will be displayed about the feedback event that caused it to be added, including details about the email message that produced the triggering event.)

b. Select the corresponding checkbox of one or more email addresses you want to remove from your account suppression list and choose **Remove**.

Viewing a list of addresses that are on your Amazon SES account-level suppression list

You can view a list of all of the email addresses that are on your account-level suppression list for your account by using the ListSuppressedDestinations operation in the SES API v2.

To view a list of all of the email addresses that are on your account-level suppression list

- At the command line, enter the following command:

```
aws sesv2 list-suppressed-destinations
```

The preceding command returns all of the email addresses that are in your account-level suppression list for your account. The output resembles the following example:

```
{
    "SuppressedDestinationSummaries": [
        {
            "EmailAddress": "recipient2@example.com",
            "Reason": "COMPLAINT",
            "LastUpdateTime": "2020-04-10T21:03:05Z"
        },
        {
            "EmailAddress": "recipient0@example.com",
            "Reason": "COMPLAINT",
            "LastUpdateTime": "2020-04-10T21:04:26Z"
        },
        {
            "EmailAddress": "recipient1@example.com",
            "Reason": "BOUNCE",
            "LastUpdateTime": "2020-04-10T22:07:59Z"
        }
    ]
}
```

- **Note** – If your output includes a "NextToken" field with a string value, this indicates there are additional email addresses on the suppression list for your account. To view additional suppressed addresses, issue another request to `ListSuppressedDestinations`, and pass the returned string value in the `--next-token` parameter like so:

  ```
  aws sesv2 list-suppressed-destinations --next-token string
  ```

  In the preceding command, replace *`string`* with the returned NextToken value.

  For more information, see How to list over 1000 email addresses from account-level suppression list.

You can use the `StartDate` option to only show email addresses that were added to the list *after* a certain date.

To view a list of addresses that were added to your account-level suppression list after a specific date

- At the command line, enter the following command:

```
aws sesv2 list-suppressed-destinations --start-date 1604394130
```

  In the preceding command, replace *1604394130* with the Unix timestamp of the start date.

You can also use the `EndDate` option to only show email addresses that were added to the list *before* a certain date.

To view a list of addresses that were added to your account-level suppression list before a specific date

- At the command line, enter the following command:

```
aws sesv2 list-suppressed-destinations --end-date 1611126000
```

  In the preceding command, replace *1611126000* with the Unix timestamp of the end date.

On the Linux, macOS, or Unix command line, you can also use the built-in `grep` utility to search for specific addresses or domains.

To search your account-level suppression list for a specific address

- At the command line, enter the following command:

```
aws sesv2 list-suppressed-destinations | grep -A2 'example.com'
```

  In the preceding command, replace *example.com* with the string of text (such as the address or domain) that you want to search for.

To view a list of all of the email addresses that are on your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Suppression list** pane, all the email addresses on your account-level suppression list are displayed with their suppression reason and date added - the following options are available:

    a. Select an email address, or select its corresponding checkbox and choose **View report** to view its details. (If it's an address that was automatically added to your suppression list because of a bounce or complaint, information will be displayed about the feedback event that caused it to be added, including details about the email message that produced the triggering event.)

    b. You can customize the suppression list table by choosing the gear icon - a modal will be presented where you can customize page size, line wrap, and columns to view - after making your selections, choose **Confirm**. The suppression list table will reflect your viewing choices.

Removing individual email addresses from your Amazon SES account-level suppression list

If an address is on the suppression list for your account, but you know that the address shouldn't be on the list, you can remove it by using DeleteSuppressedDestination operation in the SES API v2.

To remove individual addresses from your account-level suppression list using the AWS CLI

- At the command line, enter the following command:

    Linux, macOS, or Unix

    ```
    aws sesv2 delete-suppressed-destination \
    --email-address recipient@example.com
    ```

Windows

```
aws sesv2 delete-suppressed-destination `
--email-address recipient@example.com
```

In the preceding example, replace *recipient@example.com* with the email address that you want to remove from your account-level suppression list.

To remove individual addresses from your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. Remove individual email addresses either by *(a)* table selection or *(b)* typed entry:

   a. *Select from table*: In the **Suppression list** table, select the corresponding checkbox of one or more email addresses and choose **Remove**.

   b. *Type in field*:

      i. In the **Suppression list** table, choose **Remove email address**.

      ii. Type an email address in the **Email address** field - if you need to enter more addresses, choose **Enter another address** and repeat for each additional one.

      iii. When done entering addresses, review your entries for accuracy. If you decide any of your entries shouldn't be part of this submission, choose its **Remove** button.

      iv. Choose **Save changes** to remove the entered email addresses from your account-level suppression list.

Removing email addresses in bulk from your Amazon SES account-level suppression list

You can remove addresses in bulk by first uploading your contact list into an Amazon S3 object followed by using the CreateImportJob operation in the SES API v2.

Note

- There's no limit to the number of addresses that you can remove from the account-level suppression list, but there is a bulk delete limit of 10,000 addresses in an Amazon S3 object per API call.
- If your data source is an S3 bucket, it must exist in the same region as you're importing into.

To remove email addresses in bulk from your account-level suppression list, complete the following steps.

- Upload your address list into an Amazon S3 object in either CSV or JSON format.

  CSV format example for removing addresses:

  ```
  recipient3@example.com
  ```

  Only newline-delimited JSON files are supported. In this format, each line is a complete JSON object that contains an individual address definition.

  JSON format example for adding addresses:

  ```
  {"emailAddress":"recipient3@example.com"}
  ```

  In the preceding examples, replace *recipient3@example.com* with the email addresses that you want to remove from your account-level suppression list.
- Give SES permission to read the Amazon S3 object.

  When applied to an Amazon S3 bucket, the following policy gives SES permission to read that bucket. For more information about attaching policies to Amazon S3 buckets, see Using Bucket Policies and User Policies in the *Amazon Simple Storage Service User Guide*.
- Give SES permission to use your AWS KMS key.

  If the Amazon S3 object is encrypted with an AWS KMS key, you need to give Amazon SES permission to use the AWS KMS key. SES can only attain permission from a customer managed key, not a default KMS key. You need to give SES permission to use the customer managed key by adding a statement to the key's policy.

Paste the following policy statement into the key policy to permit SES to use your customer managed key.

```
{
   "Sid": "AllowSESToDecrypt",
   "Effect": "Allow",
   "Principal": {
       "Service":"ses.amazonaws.com"
   },
   "Action": [
       "kms:Decrypt",
   ],
   "Resource": "*"
}
```

- Use the CreateImportJob operation in the SES API v2.

> **Note**
>
> The following example assumes that you've already installed the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

At the command line, enter the following command. Replace *s3bucket* with the name of the Amazon S3 bucket and *s3object* with the name of the Amazon S3 object.

```
aws sesv2 create-import-job --import-destination
 SuppressionListDestination={SuppressionListImportAction=DELETE} --import-data-source
 S3Url="s3://s3bucket/s3object",DataFormat=CSV
```

To remove email addresses in bulk from your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.

3. In the **Suppression list** table, expand the **Bulk actions** button and select **Remove email addresses in bulk**.
4. In **Bulk action specifications**, select either (a) **Choose file from S3 bucket** or (b) **Import from file** - procedures are given for each import method:

    a. **Choose file from S3 bucket** - *if your source file is already stored in an Amazon S3 bucket*:

        i. If you know the URI of the Amazon S3 bucket you want to use, enter it in the **Amazon S3 URI** field; otherwise, choose **Browse S3**:

            A. In **Buckets**, select the name of the S3 bucket.

            B. In **Objects**, select the name of the file then select **Choose** - you'll be returned to **Bulk action specifications**.

            C. (Optional) If you want to be taken to the Amazon S3 console to view details about your S3 object choose **View**.

        ii. In **File format**, select the format of the file you've chosen to import from your Amazon S3 bucket.

        iii. Choose **Remove email addresses** to kick off the import of addresses from your file - a table under the **Bulk actions** tab is displayed.

    b. **Import from file** - *if you have a local source file to upload to a new or existing Amazon S3 bucket*:

        i. In **Import source file**, select **Choose file**.

        ii. Select the JSON or CSV file in the file browser and choose **Open** - you'll see the name, size, and date of your file displayed under the **Choose file** button.

        iii. Expand **Amazon S3 bucket** and select the S3 bucket.

            - To upload your file to a new bucket, choose **Create S3 bucket**, enter a name in the **Bucket name** field, and choose **Create bucket**.

        iv. Choose **Remove email addresses** to kick off the import of addresses from your file - a table under the **Bulk actions** tab is displayed.

5. Regardless of the import method you used, your job ID will be listed in **Bulk actions** along with import type, status, and date - to view job details, select the job ID.
6. Select the **Suppression list** tab and all the successfully imported email addresses that were removed from your suppression list will no longer be displayed.

Viewing a list of import jobs for the account

You can view a list of all of the email addresses that are on your account-level suppression list for your account by using the ListImportJobs operation in the Amazon SES API v2.

To view a list of all of the import jobs for the account

- At the command line, enter the following command:

```
aws sesv2 list-import-jobs
```

The preceding command returns all of the import jobs for the account. The output resembles the following example:

```
{
    "ImportJobs": [
        {
            "CreatedTimestamp": 2020-07-31T06:06:55Z",
            "ImportDestination": {
                "SuppressionListDestination": {
                    "SuppressionListImportAction": "PUT"
                }
            },
            "JobStatus": "COMPLETED",
            "JobId": "755380d7-fbdb-4ed2-a9a3-06866220f5b5"
        },
        {
            "CreatedTimestamp": "2020-07-30T18:45:32Z",
            "ImportDestination": {
                "SuppressionListDestination": {
                    "SuppressionListImportAction": "DELETE"
                }
            },
            "JobStatus": "COMPLETED",
            "JobId": "076683bd-a7ee-4a40-9754-4ad1161ba8b6"
        },
        {
            "CreatedTimestamp": "2020-08-05T16:45:18Z",
            "ImportDestination": {
                "SuppressionListDestination": {
                    "SuppressionListImportAction": "PUT"
```

```
                }
            },
            "JobStatus": "COMPLETED",
            "JobId": "6e261869-bd30-4b33-b1f2-9e035a83a395"
        }
    ]
}
```

To view a list of all of the import jobs for the account using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Suppression list** pane, select the **Bulk actions** tab.
4. All the import jobs will be listed in the **Bulk actions** table along with import type, status, and date.
5. To view job details, select the job ID and the following panes are displayed:

    a. **Bulk action status**: shows the jobs overall status, the time and date it completed, how many records where imported, and the count of any records that failed to import successfully.

    b. **Bulk action details**: shows the job ID, whether it was used to add or remove addresses, whether the file format was JSON or CSV, the URI of the Amazon S3 bucket where the bulk file was stored, and the time and date the bulk action was created.

Getting information about an import job for the account

You can get information about an import job for the account by using the GetImportJob operation in the Amazon SES API v2.

To get information about an import job for the account

- At the command line, enter the following command:

```
aws sesv2 get-import-job --job-id JobId
```

The preceding command returns information about an import job for the account. The output resembles the following example:

```
{
    "ImportDataSource": {
        "S3Url": "s3://bucket/object",
        "DataFormat": "CSV"
    },
    "ProcessedRecordsCount": 2,
    "FailureInfo": {
        "FailedRecordsS3Url": "s3presignedurl"
    },
    "JobStatus": "COMPLETED",
    "JobId": "jobid",
    "CreatedTimestamp": "2020-08-12T17:05:15Z",
    "FailedRecordsCount": 1,
    "ImportDestination": {
        "SuppressionListDestination": {
            "SuppressionListImportAction": "PUT"
        }
    },
    "CompletedTimestamp": "2020-08-12T17:06:42Z"
}
```

To get information about an import job for the account using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Suppression list** pane, select the **Bulk actions** tab.
4. All the import jobs will be listed in the **Bulk actions** table along with import type, status, and date.
5. To view job details, select the job ID and the following panes are displayed:

    a. **Bulk action status**: shows the jobs overall status, the time and date it completed, how many records where imported, and the count of any records that failed to import successfully.

b. **Bulk action details**: shows the job ID, whether it was used to add or remove addresses, whether the file format was JSON or CSV, the URI of the Amazon S3 bucket where the bulk file was stored, and the time and date the bulk action was created.

Disabling the Amazon SES account-level suppression list

You can use the PutAccountSuppressionAttributes operation in the SES API v2 to effectively disable your account-level suppression list by removing the values from the `suppressed-reasons` attribute.

To disable your account-level suppression list using the AWS CLI

- At the command line, enter the following command:

```
aws sesv2 put-account-suppression-attributes --suppressed-reasons
```

To disable your account-level suppression list using the SES console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Suppression list**.
3. In the **Account-level settings** pane, choose **Edit**.
4. In **Suppression list**, uncheck the **Enabled** box.
5. Choose **Save changes**.

Using tenant-level suppression lists in Amazon SES

If you use Amazon Simple Email Service (SES) multi-tenancy to send email on behalf of multiple tenants, you can give each tenant its own suppression list. Tenant-level suppression isolates bounces and complaints so that they affect only the tenant that sent the email. Without this feature, all tenants share the account-level suppression list. When a recipient bounces or complains about one tenant's email, that address is suppressed for every tenant in the account. Tenant-level suppression prevents this cross-tenant contamination by maintaining a separate list of suppressed addresses for each tenant.

How tenant-level suppression works

Two settings control suppression behavior for each tenant: *suppression scope* and *suppressed reasons*. You configure both settings together by using the `PutTenantSuppressionAttributes` API operation, or by specifying `SuppressionAttributes` when you create a tenant with the `CreateTenant` API operation.

Suppression scope determines which suppression list SES uses for the tenant:

- `TENANT` – Use the tenant's own suppression list.
- `ACCOUNT` – Use the account-level suppression list. This is the default for both existing and new tenants.

Suppressed reasons determines which events cause SES to add addresses to the suppression list:

- `BOUNCE` – Add addresses that produce hard bounces.
- `COMPLAINT` – Add addresses that produce complaints.
- `BOUNCE` and `COMPLAINT` – Add addresses that produce either event.
- Empty – Don't check or record to any suppression list.

> **Note**
>
> You must provide `SuppressionScope` and `SuppressedReasons` together, or set both to null. Setting both to null clears the tenant's suppression settings and returns the tenant to the default `ACCOUNT` scope behavior.

Suppression behavior by configuration

The following table describes the send-time and recording behavior for each combination of suppression scope and suppressed reasons.

Scope	Reasons	Send-time behavior	Recording behavior
`TENANT`	`BOUNCE`, `COMPLAINT`	Checks tenant list for bounces and complaints	Records both to tenant list

Scope	Reasons	Send-time behavior	Recording behavior
TENANT	BOUNCE	Checks tenant list for bounces only	Records bounces to tenant list only
TENANT	COMPLAINT	Checks tenant list for complaints only	Records complaints to tenant list only
TENANT	Empty	No suppression check	No recording
ACCOUNT	BOUNCE, COMPLAINT	Checks account list for bounces and complaints	Records both to account list
ACCOUNT	BOUNCE	Checks account list for bounces only	Records bounces to account list only
ACCOUNT	COMPLAINT	Checks account list for complaints only	Records complaints to account list only
ACCOUNT	Empty	No suppression check	No recording

Suppression precedence and send-time behavior

SES resolves suppression settings in the following precedence order, from highest to lowest:

1. **Configuration set** – If the configuration set specifies suppression settings, SES uses those settings.
2. **Tenant** – If no configuration set override exists, SES uses the tenant's suppression settings.
3. **Account** – If neither the configuration set nor the tenant specifies settings, SES uses the account-level defaults.

> **Important**
>
> The suppression scope determines which suppression list SES checks at send time. When the scope is TENANT, SES checks only the tenant's suppression list and skips the account-level suppression list. An address that is on the account-level suppression list but not on the tenant's suppression list is not suppressed. When the scope is ACCOUNT, SES

checks only the account-level suppression list. The scope also controls where bounces and complaints are recorded.

Configuration set overrides

The `SuppressionOptions` for a configuration set now includes a `SuppressionScope` field. You can use a configuration set to override the scope only, the reasons only, or both independently. This gives you fine-grained control over suppression behavior for specific sending workflows. You don't need to change the tenant's default settings.

Automatic suppression recording

When the suppression scope is `TENANT`, SES automatically records suppression entries as follows:

- **Hard bounces** – SES adds the address to the tenant's suppression list and the global suppression list. SES does *not* add the address to the account-level suppression list.
- **Complaints** – SES adds the address to the tenant's suppression list only.
- **Not-spam feedback** – When a recipient marks a previously reported message as not spam, SES automatically removes `COMPLAINT`-reason entries from the tenant's suppression list.

Identifying tenant suppression bounces

When SES suppresses a message because the address is on a tenant's suppression list, you can identify the event by using the following indicators:

- **Bounce notification** – The bounce type is `Permanent` with a subtype of `OnTenantSuppressionList`.
- **Virtual Deliverability Manager (VDM)** – The bounce reason is `ON_TENANT_SUPPRESSION_LIST`.
- **Diagnostic code** – `"Amazon SES did not send the message to this address because it is on the suppression list for your tenant."`
- **Event tag** – SES adds a `ses:tenant-name` tag to bounce and complaint notifications so that you can identify which tenant the event belongs to.

Managing tenant suppression list entries

You manage tenant suppression list entries by using the same API operations that you use for account-level entries. To target a tenant's suppression list, include the `TenantName` parameter in your request:

- `PutSuppressedDestination` – Add an address to a tenant's suppression list.
- `GetSuppressedDestination` – Retrieve details about a suppressed address for a tenant.
- `DeleteSuppressedDestination` – Remove an address from a tenant's suppression list.
- `ListSuppressedDestinations` – List all suppressed addresses for a tenant.

If you omit the `TenantName` parameter, these operations target the account-level suppression list. This maintains backward compatibility with existing integrations.

Considerations and limitations

Keep the following in mind when you use tenant-level suppression lists:

- **Multi-tenancy required** – You must have SES multi-tenancy configured with at least one tenant to use this feature.
- **One list per tenant** – Each tenant has exactly one suppression list. There is a 1:1 mapping between tenants and suppression lists.
- **Region-specific** – Tenant suppression lists are specific to the AWS Region in which you configure them.
- **Addresses persist** – Suppressed addresses remain on the tenant's list until you explicitly remove them or until SES automatically removes them (for example, when a recipient reports a not-spam event for a `COMPLAINT`-reason entry).
- **Case-sensitive storage** – Email addresses on the suppression list are stored with case preserved. API calls for suppression list management require an exact case match.
- **Sending quota impact** – Messages suppressed by a tenant suppression list count against your daily sending quota.
- **Gmail complaint data** – Gmail does not send complaint data to SES. Addresses hosted by Gmail are not automatically added to the suppression list for complaints.
- **Sandbox restrictions** – You can't call `PutSuppressedDestination` until your account has production access.

- **Clearing settings** – To clear a tenant's suppression settings, call `PutTenantSuppressionAttributes` with null values for both scope and reasons.
- **Deleting a tenant** – When you delete a tenant, SES also deletes all suppression list entries for that tenant.

Related resources

The following resources can help you learn more about suppression lists in SES.

- Using the Amazon SES account-level suppression list
- Amazon SES global suppression list

Configuring tenant-level suppression

Configure tenant-level suppression settings to control how Amazon Simple Email Service (Amazon SES) handles bounces and complaints for individual tenants. By default, tenants use your account-level suppression list. When you enable tenant-level suppression, SES maintains a separate suppression list for each tenant. This gives you granular control over email deliverability.

This topic provides procedures for enabling, modifying, disabling, and viewing tenant-level suppression settings by using the AWS CLI.

> **Important**
>
> Disabling tenant-level suppression does not delete existing entries from the tenant's suppression list. If you re-enable tenant-level suppression later, those entries are still active.

Prerequisites

Before you configure tenant-level suppression, make sure that you meet the following requirements:

- Multi-tenancy is enabled in your SES account.
- At least one tenant exists in your account, or you plan to create one as part of this procedure.
- AWS CLI version 2 is installed and configured with appropriate permissions. For more information, see the AWS Command Line Interface User Guide.

- You are familiar with account-level suppression concepts. For more information, see Using the Amazon SES account-level suppression list.

Enabling tenant-level suppression for an existing tenant

Use the `PutTenantSuppressionAttributes` API operation to enable tenant-level suppression for an existing tenant. This operation is idempotent, so you can call it multiple times with the same parameters without side effects.

To enable tenant-level suppression

1. At the command line, run the following `put-tenant-suppression-attributes` command to enable tenant-level suppression for both bounces and complaints:

   Linux, macOS, or Unix

   ```
   aws sesv2 put-tenant-suppression-attributes \
   --tenant-name MyTenant \
   --suppression-scope TENANT \
   --suppressed-reasons BOUNCE COMPLAINT
   ```

   Windows

   ```
   aws sesv2 put-tenant-suppression-attributes `
   --tenant-name MyTenant `
   --suppression-scope TENANT `
   --suppressed-reasons BOUNCE COMPLAINT
   ```

   Replace *MyTenant* with the name of your tenant.

2. Verify that the command returns no output. A successful request returns an HTTP 200 response with no body.
3. Confirm the configuration by running the following command:

   ```
   aws sesv2 get-tenant --tenant-name MyTenant
   ```

The tenant now uses its own suppression list for both bounces and complaints.

You can also configure suppression for a single reason type. The following examples show how to enable suppression for bounces only or complaints only.

The following command suppresses bounces only:

Linux, macOS, or Unix

```
aws sesv2 put-tenant-suppression-attributes \
--tenant-name MyTenant \
--suppression-scope TENANT \
--suppressed-reasons BOUNCE
```

Windows

```
aws sesv2 put-tenant-suppression-attributes `
--tenant-name MyTenant `
--suppression-scope TENANT `
--suppressed-reasons BOUNCE
```

The following command suppresses complaints only:

Linux, macOS, or Unix

```
aws sesv2 put-tenant-suppression-attributes \
--tenant-name MyTenant \
--suppression-scope TENANT \
--suppressed-reasons COMPLAINT
```

Windows

```
aws sesv2 put-tenant-suppression-attributes `
--tenant-name MyTenant `
--suppression-scope TENANT `
--suppressed-reasons COMPLAINT
```

Note

You must provide `--suppression-scope` and `--suppressed-reasons` together. Providing one without the other returns an `InvalidParameterValueException`.

Configuring suppression when creating a new tenant

Use the `CreateTenant` API operation with the `SuppressionAttributes` parameter to create a tenant with tenant-level suppression enabled from the start.

To create a tenant with tenant-level suppression

1. At the command line, run the following `create-tenant` command:

   Linux, macOS, or Unix

   ```
   aws sesv2 create-tenant \
   --tenant-name MyNewTenant \
   --suppression-attributes '{"SuppressionScope":"TENANT","SuppressedReasons":
   ["BOUNCE","COMPLAINT"]}'
   ```

   Windows

   ```
   aws sesv2 create-tenant `
   --tenant-name MyNewTenant `
   --suppression-attributes '{"SuppressionScope":"TENANT","SuppressedReasons":
   ["BOUNCE","COMPLAINT"]}'
   ```

   Replace *MyNewTenant* with the name for your new tenant.

2. Verify that the response contains the tenant ID:

   ```
   {
       "TenantId": "tn-abc123def456"
   }
   ```

The tenant is created with tenant-level suppression enabled for both bounces and complaints.

> **Note**
>
> If you omit `--suppression-attributes`, the tenant defaults to `ACCOUNT` scope and uses the account-level suppression list.

Overriding tenant suppression with a configuration set

You can override a tenant's suppression settings by adding `SuppressionScope` to a configuration set's `Suppression0ptions`. When you send email using this configuration set, the configuration set's suppression settings take precedence over the tenant's settings. You can override scope only, reasons only, or both independently.

To create a configuration set with tenant scope override

1. At the command line, run the following `create-configuration-set` command:

   Linux, macOS, or Unix

   ```
   aws sesv2 create-configuration-set \
   --configuration-set-name my-config-set \
   --suppression-options '{"SuppressionScope":"TENANT","SuppressedReasons":
   ["BOUNCE"]}'
   ```

   Windows

   ```
   aws sesv2 create-configuration-set `
   --configuration-set-name my-config-set `
   --suppression-options '{"SuppressionScope":"TENANT","SuppressedReasons":
   ["BOUNCE"]}'
   ```

   Replace *my-config-set* with the name for your configuration set.

2. Verify that the command returns no output. A successful request returns an HTTP 200 response.

To update an existing configuration set with suppression options

1. At the command line, run the following `put-configuration-set-suppression-options` command:

   Linux, macOS, or Unix

   ```
   aws sesv2 put-configuration-set-suppression-options \
   --configuration-set-name my-config-set \
   --suppression-scope TENANT \
   ```

```
--suppressed-reasons BOUNCE
```

Windows

```
aws sesv2 put-configuration-set-suppression-options `
--configuration-set-name my-config-set `
--suppression-scope TENANT `
--suppressed-reasons BOUNCE
```

Replace *my-config-set* with the name of your configuration set.

2. Verify that the command returns no output. A successful request returns an HTTP 200 response.

Disabling tenant-level suppression

To return a tenant to account-level suppression, you can either set the scope to ACCOUNT or clear all suppression settings.

To set the tenant to account-level suppression

1. At the command line, run the following command to explicitly set the scope to ACCOUNT:

   Linux, macOS, or Unix

```
aws sesv2 put-tenant-suppression-attributes \
--tenant-name MyTenant \
--suppression-scope ACCOUNT \
--suppressed-reasons BOUNCE COMPLAINT
```

   Windows

```
aws sesv2 put-tenant-suppression-attributes `
--tenant-name MyTenant `
--suppression-scope ACCOUNT `
--suppressed-reasons BOUNCE COMPLAINT
```

   Replace *MyTenant* with the name of your tenant.

2. Verify that the command returns no output.

To clear all suppression settings

1. At the command line, run the following command without specifying scope or reasons:

```
aws sesv2 put-tenant-suppression-attributes --tenant-name MyTenant
```

   Replace *MyTenant* with the name of your tenant.
2. Verify that the command returns no output.

> **Note**
>
> When you omit both `--suppression-scope` and `--suppressed-reasons`, SES clears the tenant's suppression settings. The tenant falls back to account-level suppression behavior.

> **Important**
>
> Disabling tenant-level suppression does not delete existing entries from the tenant's suppression list. If you re-enable tenant-level suppression later, those entries are still active.

Viewing current suppression configuration

Use the `GetTenant` API operation to check a tenant's current suppression settings.

To view a tenant's suppression configuration

1. At the command line, run the following `get-tenant` command:

```
aws sesv2 get-tenant --tenant-name MyTenant
```

   Replace *MyTenant* with the name of your tenant.
2. Review the response. The following example shows a tenant with tenant-level suppression enabled:

```
{
    "Tenant": {
        "TenantName": "MyTenant",
        "TenantId": "tn-abc123def456",
        "TenantArn": "arn:aws:ses:us-east-1:123456789012:tenant/MyTenant/tn-
abc123def456",
        "SendingStatus": "ENABLED",
        "CreatedTimestamp": 1705312800,
        "UpdateTimestamp": 1705762800,
        "SuppressionAttributes": {
            "SuppressionScope": "TENANT",
            "SuppressedReasons": ["BOUNCE", "COMPLAINT"]
        }
    }
}
```

The following table describes the key fields in the `SuppressionAttributes` object.

Field	Description
`SuppressionScope`	The suppression scope for the tenant. A value of `TENANT` means the tenant uses its own suppression list. A value of `ACCOUNT` means the tenant uses the account-level suppression list.
`SuppressedReasons`	The suppression reasons that are active for the tenant. Possible values are `BOUNCE`, `COMPLAINT` , or both.

If `SuppressionAttributes` is absent or null in the response, the tenant uses account-level suppression.

Troubleshooting

The following are common errors that you might encounter when configuring tenant-level suppression.

`InvalidParameterValueException: SuppressedReasons cannot be null when SuppressionScope is provided`

- **Cause** – You specified `--suppression-scope` without `--suppressed-reasons`.
- **Fix** – Provide both parameters together, or omit both to clear settings.

`InvalidParameterValueException: SuppressionScope is required when SuppressedReasons are provided`

- **Cause** – You specified `--suppressed-reasons` without `--suppression-scope`.
- **Fix** – Provide both parameters together.

`NotFoundException`

- **Cause** – The tenant name that you specified doesn't exist.
- **Fix** – Verify the tenant name by running the following command:

```
aws sesv2 list-tenants
```

Related resources

- Using tenant-level suppression lists in Amazon SES
- Managing tenant suppression list entries
- Using the Amazon SES account-level suppression list
- Tenants

Managing tenant suppression list entries

You manage entries on a tenant's suppression list by using the same Amazon Simple Email Service (SES) API operations that you use for the account-level suppression list. To target a specific tenant's list, include the `--tenant-name` parameter in each request. When you omit this parameter, the operations target the account-level list. Use the AWS CLI to add, check, remove, and list suppressed addresses for a specific tenant.

> **Important**
>
> Email addresses on the suppression list are stored with case preserved. While email sending functionality treats addresses with different cases as identical, API calls for suppression list

management require an exact case match. Make sure that you use the exact case of the email address as it appears in the suppression list when you add, check, or remove entries.

Prerequisites

Before you manage tenant suppression list entries, make sure that you meet the following requirements:

- Tenant-level suppression is enabled for the tenant. For more information, see Configuring tenant-level suppression.
- AWS CLI version 2 is installed and configured with appropriate permissions. For more information, see the AWS Command Line Interface User Guide.
- The tenant exists in your SES account.

Adding an address to a tenant's suppression list

Use the `PutSuppressedDestination` API operation with the `--tenant-name` parameter to add an email address to a tenant's suppression list.

To add an address to a tenant's suppression list using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 put-suppressed-destination \
  --email-address recipient@example.com \
  --reason BOUNCE \
  --tenant-name MyTenant
  ```

  Windows

  ```
  aws sesv2 put-suppressed-destination `
  --email-address recipient@example.com `
  --reason BOUNCE `
  --tenant-name MyTenant
  ```

In the preceding example, replace *recipient@example.com* with the email address to add, *BOUNCE* with the suppression reason (acceptable values are BOUNCE and COMPLAINT), and *MyTenant* with the name of your tenant.

The command produces no output on success (HTTP 200). The address is now on the tenant's suppression list.

> **Note**
>
> The `--reason` parameter accepts BOUNCE or COMPLAINT. You can add the same address with different reasons.

Checking if an address is on a tenant's suppression list

Use the `GetSuppressedDestination` API operation with the `--tenant-name` parameter to check whether an email address is on a tenant's suppression list.

To check if an address is on a tenant's suppression list using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 get-suppressed-destination \
  --email-address recipient@example.com \
  --tenant-name MyTenant
  ```

  Windows

  ```
  aws sesv2 get-suppressed-destination `
  --email-address recipient@example.com `
  --tenant-name MyTenant
  ```

  In the preceding example, replace *recipient@example.com* with the email address to check and *MyTenant* with the name of your tenant.

If the address is on the tenant's suppression list, the command returns output that resembles the following example:

```
{
    "SuppressedDestination": {
        "EmailAddress": "recipient@example.com",
        "Reason": "BOUNCE",
        "LastUpdateTime": "2026-03-15T10:30:00.000Z",
        "Attributes": {
            "MessageId": "0100018e1234abcd-12345678-abcd-1234-
abcd-123456789012-000000",
            "FeedbackId": "0100018e1234abce-12345678-abcd-1234-
abcd-123456789012-000000"
        }
    }
}
```

The response includes the following fields:

- `EmailAddress` – The suppressed email address.
- `Reason` – Why the address was suppressed (`BOUNCE` or `COMPLAINT`).
- `LastUpdateTime` – When the entry was last updated.
- `Attributes` – Contains `MessageId` and `FeedbackId` from the event that caused suppression. These fields are present for automatically suppressed addresses and absent for manually added entries.

> **Note**
>
> If the address is not on the tenant's suppression list, the API returns a `NotFoundException`.

Removing an address from a tenant's suppression list

Use the `DeleteSuppressedDestination` API operation with the `--tenant-name` parameter to remove an email address from a tenant's suppression list.

To remove an address from a tenant's suppression list using the AWS CLI

- At the command line, enter the following command:

  Linux, macOS, or Unix

  ```
  aws sesv2 delete-suppressed-destination \
  --email-address recipient@example.com \
  --tenant-name MyTenant
  ```

  Windows

  ```
  aws sesv2 delete-suppressed-destination `
  --email-address recipient@example.com `
  --tenant-name MyTenant
  ```

  In the preceding example, replace *recipient@example.com* with the email address to remove and *MyTenant* with the name of your tenant.

The command produces no output on success (HTTP 200). The address is removed from the tenant's suppression list.

> **Important**
>
> Removing an address from a tenant's suppression list does not remove it from the global suppression list or the account-level suppression list. Each list is managed independently.

Listing addresses on a tenant's suppression list

Use the `ListSuppressedDestinations` API operation with the `--tenant-name` parameter to retrieve all suppressed addresses for a specific tenant.

To list all addresses on a tenant's suppression list using the AWS CLI

- At the command line, enter the following command:

Linux, macOS, or Unix

```
aws sesv2 list-suppressed-destinations \
--tenant-name MyTenant
```

Windows

```
aws sesv2 list-suppressed-destinations `
--tenant-name MyTenant
```

In the preceding example, replace *MyTenant* with the name of your tenant.

The following example output shows two suppressed addresses:

```
{
    "SuppressedDestinationSummaries": [
        {
            "EmailAddress": "bounce@example.com",
            "Reason": "BOUNCE",
            "LastUpdateTime": "2026-03-10T08:15:00.000Z"
        },
        {
            "EmailAddress": "complaint@example.com",
            "Reason": "COMPLAINT",
            "LastUpdateTime": "2026-03-12T14:30:00.000Z"
        }
    ]
}
```

Filtering the results

You can filter the results by reason, date range, or both.

The following command filters by reason:

Linux, macOS, or Unix

```
aws sesv2 list-suppressed-destinations \
--tenant-name MyTenant \
```

```
--reasons BOUNCE COMPLAINT
```

Windows

```
aws sesv2 list-suppressed-destinations `
--tenant-name MyTenant `
--reasons BOUNCE COMPLAINT
```

The following command filters by date range:

Linux, macOS, or Unix

```
aws sesv2 list-suppressed-destinations \
--tenant-name MyTenant \
--start-date 2026-01-01T00:00:00Z \
--end-date 2026-03-31T23:59:59Z
```

Windows

```
aws sesv2 list-suppressed-destinations `
--tenant-name MyTenant `
--start-date 2026-01-01T00:00:00Z `
--end-date 2026-03-31T23:59:59Z
```

If the response includes a `NextToken` value, use it in a subsequent request to retrieve more results. Use the `--page-size` parameter to control the number of results per page.

> **Note**
>
> The list operation returns summary information only. To get full details including `Attributes`, use `GetSuppressedDestination` for individual addresses.

Automatic suppression entries

SES automatically manages suppression entries based on the tenant's suppression configuration. You don't need to take any action for automatic entries. They appear in the tenant's suppression list alongside manually added entries.

SES automatically adds entries in the following cases:

- A hard bounce occurs, if bounce suppression is enabled for the tenant.
- A complaint is received, if complaint suppression is enabled for the tenant.

SES automatically removes entries in the following case:

- A not-spam report is received for a complaint-suppressed address.

Using configuration set-level suppression to override your account-level suppression list

While the account-level suppression list is set for your entire account, you can customize it separately for different configuration sets by overriding it with configuration set-level suppression. This finer granularity allows you to use customized suppression settings for different email sending groups that you've assigned to their own configuration sets. For example, let's say your account-level suppression list is configured for both bounce and complaint addresses to be added, but you have a particular email demographic defined in a configuration set for which you're only interested in complaint addresses being added - you would achieve this by enabling this configuration set's suppression overrides so that email addresses are added to your account-level suppression list just for complaints (not bounces and complaints like is set in your account-level suppression list) from email sent with this configuration set.

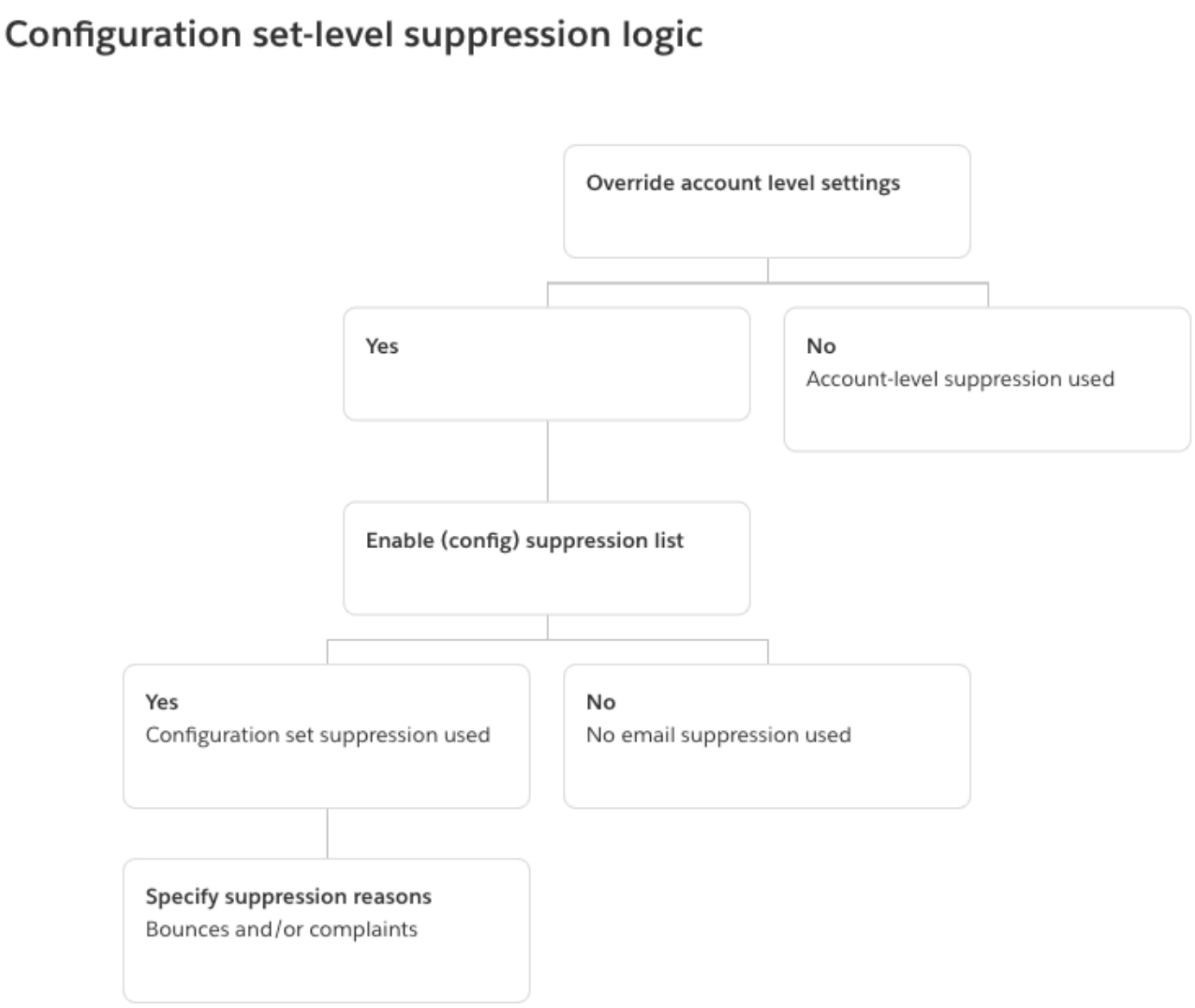
If you use multi-tenancy, configuration sets can also override tenant-level suppression settings. The `SuppressionOptions` for a configuration set includes a `SuppressionScope` field that determines which suppression list SES uses. You can set `SuppressionScope` to `TENANT` to use the tenant's suppression list, or `ACCOUNT` to use the account-level suppression list. You can override the scope, the suppressed reasons, or both independently. For more information about tenant-level suppression, see Using tenant-level suppression lists in Amazon SES.

> **Important**
>
> If you set `SuppressionScope` to `TENANT` on a configuration set, all `SendEmail` and `SendBulkEmail` requests that use this configuration set must include a tenant name. Requests without a tenant name are rejected.

With configuration set-level suppression, there are different levels of overriding your account-level suppression, including not using any suppression at all. To help understand these various levels of suppression that can be set in the following console procedures, the following relationship map models the decision set of choices you can make for the enabling or disabling of various levels of overrides, that depending on their combination, can be used to implement three different levels of suppression:

- **No overrides (default)** – the configuration set uses your account-level suppression list settings.
- **Override account level settings** – this will negate any account-level suppression list settings; email sent with this configuration set will not use any suppression settings at all.
- **Override account level settings with configuration set-level suppression enabled** – email sent with this configuration set will only use the suppression conditions you enabled for it (bounces, complaints, or bounces and complaints) - regardless of what your account-level suppression list settings are, it will override them.

> **Note**
>
> For any suppression condition *not* specified at the configuration set-level, suppression behavior will fall back to the global suppression list since account-level settings have been overridden.



Keep in mind that configuration set-level suppression is not an actual suppression *list*, rather, it's simply a mechanism to override your account-level suppression list with custom suppression settings defined in a configuration set - this means any email sent using the configuration set will only use its own suppression settings and ignore any account-level suppression settings. In other words, configuration set-level suppression is interacting with your account-level suppression list by simply changing (overriding) the suppression reasons that determine what email addresses get added to your account-level suppression list.

Enabling configuration set-level suppression

To enable configuration set-level suppression using the Amazon SES new console:

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, under **Configuration**, choose **Configuration sets**.
3. In **Configuration sets**, choose the name of the configuration set you want to configure with customized suppression.
4. In the **Suppression list options** pane, choose **Edit**.
5. The **Suppression list options** section provides a decision set to define customized suppression starting with the option to use this configuration set to override your account-level suppression. The configuration set-level suppression logic map will help you understand the effects of the override combinations. These multitiered selections of overrides can be combined to implement three different levels of suppression:

   a. **Use account-level suppression:** Do not override your account-level suppression and do not implement any configuration set-level suppression - basically, any email sent using this configuration set will just use your account-level suppression. To do this:

      - In **Suppression list settings**, uncheck the **Override account level settings** box.

   b. **Do not use any suppression:** Override your account-level suppression without enabling any configuration set-level suppression - this means any email sent using this configuration set will not use any of your account-level suppression; in other words, all suppression is cancelled. To do this:

      i. In **Suppression list settings**, check the **Override account level settings** box.

      ii. In **Suppression list**, uncheck the **Enabled** box.

   c. **Use configuration set-level suppression:** Override your account-level suppression list with custom suppression settings defined in this configuration set - this means any email sent using this configuration set will only use its own suppression settings and ignore any account-level suppression settings. To do this:

      i. In **Suppression list settings**, check the **Override account level settings** box.

      ii. In **Suppression list**, check **Enabled**.

iii. In **Specify the reason(s)...**, select one of the suppression reasons for this configuration set to use.

6. Choose **Save changes**.

Using list management

Amazon SES offers list management capabilities, which means customers can manage their own mailing lists, known as contact lists. A *contact list* is a list that allows you to store all of your contacts that have subscribed to a particular topic or topics. A *contact* is an end-user who is receiving your emails. A *topic* is an interest group, theme, or label within a list. Lists can have multiple topics.

By using the `ListContacts` operation in the Amazon SES API v2, you can retrieve a list of all your contacts who have subscribed to a particular topic, to whom you can send emails using the `SendEmail` operation.

For information about subscription management, see Using subscription management.

List management overview

You should consider the following factors when you use list management:

- You can specify list topics while creating the list.
- Only one contact list is allowed per AWS account.
- A list can have a maximum of 20 topics.
- You can update an existing contact list, including adding new topics to the list, adding or deleting contacts from a list, and updating contact preferences for a list or topic.
- You can update topic metadata, such as the topic display name or description.
- You can get a list of contacts in a contact list, contacts subscribed to a topic, contacts unsubscribed from a topic, and contacts unsubscribed from all topics in the list.
- You can import your existing contact lists to SES using the `CreateImportJob` API.
- SES will issue a bounce event for a message that is sent to an unsubscribed contact on your contact list. For more information, see Using subscription management.
- Each contact can have associated attributes which you can use to store information about that contact.

Configuring list management

You can use the following operations to configure list management capabilities. For the full list of contact list and contact operations, see the Amazon SES API v2 Reference.

Create a contact list

You can use the `CreateContactList` operation in the SES API v2 to create a contact list. You can quickly and easily configure this setting by using the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To create a contact list by using the AWS CLI

- At the command line, enter the following command:

```
aws sesv2 create-contact-list --cli-input-json file://CONTACT-LIST-JSON
```

In the preceding command, replace *CONTACT-LIST-JSON* with the path to your JSON file for your `CreateContactList` request.

An example `CreateContactList` input JSON file for the request is as follows:

```
{
    "ContactListName": "ExampleContactListName",
    "Description": "Creating a contact list example",
    "Topics": [
     {
         "TopicName": "Sports",
         "DisplayName": "Sports Newsletter",
         "Description": "Sign up for our free newsletter to receive updates on all
 sports.",
         "DefaultSubscriptionStatus": "OPT_OUT"
     },
     {
         "TopicName": "Cycling",
         "DisplayName": "Cycling newsletter",
         "Description": "Never miss a cycling update by subscribing to our
 newsletter.",
         "DefaultSubscriptionStatus": "OPT_IN"
     },
     {
         "TopicName": "NewProducts",
```

```
        "DisplayName": "New products",
        "Description": "Hear about new products by subscribing to this mailing
 list.",
        "DefaultSubscriptionStatus": "OPT_IN"
    },
    {
        "TopicName": "DailyUpdates",
        "DisplayName": "Daily updates",
        "Description": "Start your day with sport updates, Monday through
 Friday.",
        "DefaultSubscriptionStatus": "OPT_OUT"
    }
   ]
}
```

Create a contact

You can use the `CreateContact` operation in the SES API v2 to create a contact. You can quickly and easily configure this setting by using the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To create a contact by using the AWS CLI

- At the command line, enter the following command:

```
aws sesv2 create-contact --cli-input-json file://CONTACT-JSON
```

In the preceding command, replace *CONTACT-JSON* with the path to your JSON file for your `CreateContact` request.

An example `CreateContact` input JSON file for the request is as follows:

```
{
    "ContactListName": "ExampleContactListName",
    "EmailAddress": "example@amazon.com",
    "UnsubscribeAll": false,
    "TopicPreferences": [
        {
            "TopicName": "Sports",
            "SubscriptionStatus": "OPT_IN"
        }
```

```
    ],
    "AttributesData": "{\"Name\": \"John\", \"Location\": \"Seattle\"}"
}
```

In the example above, an `UnsubscribeAll` value of `false` shows that the contact has not unsubscribed from all topics, where a value of `true` would mean the contact has unsubscribed from all topics.

`TopicPreferences` includes information about the contact's subscription status to topics. In the preceding example, the contact has opted in to the "*Sports*" topic and will receive all emails to the "*Sports*" topic.

The `AttributesData` is a JSON field where you can put any metadata about our contact. It must be a valid JSON object.

Bulk importing contacts to your contact list

You can manually add addresses in bulk by first uploading your contacts into an Amazon S3 object followed by using the `CreateImportJob` operation in the SES API v2 or by using the SES console. For more information see Adding email addresses in bulk to your account-level suppression list.

You should create a contact list before importing your contacts.

> **Note**
>
> You can add up to 1 million contacts to a contact list per ImportJob.

To add contacts in bulk to your contact list, complete the following steps.

- Upload your contacts into an Amazon S3 object in either CSV or JSON format.

  CSV format

  The first line of the file that is uploaded to Amazon S3 should be a header line.

  The `topicPreferences` object needs to be flattened for the CSV format. Every topic in the `topicPreferences` will have a separate header field.

  CSV format example for adding contacts in bulk to a contact list:

```
emailAddress,unsubscribeAll,attributesData,topicPreferences.Sports,topicPreferences.Cycling
example1@amazon.com,false,{"Name": "John"},OPT_IN,OPT_OUT
example2@amazon.com,true,,OPT_OUT,OPT_OUT
```

JSON format

Only newline-delimited JSON files are supported. In this format, each line is a complete JSON object that contains one contact's information.

JSON format example for adding contacts in bulk to a contact list:

```
{
     "emailAddress": "example1@amazon.com",
     "unsubscribeAll": false,
     "attributesData": "{\"Name\":\"John\"}",
     "topicPreferences": [
      {
          "topicName": "Sports",
          "subscriptionStatus": "OPT_IN"
      },
      {
          "topicName": "Cycling",
          "subscriptionStatus": "OPT_OUT"
      }
     ]
}
{
     "emailAddress": "example2@amazon.com",
     "unsubscribeAll": true,
     "topicPreferences": [
      {
          "topicName": "Sports",
          "subscriptionStatus": "OPT_OUT"
      },
      {
          "topicName": "Cycling",
          "subscriptionStatus": "OPT_OUT"
      }
     ]
}
```

In the preceding examples, replace *example1@amazon.com* and *example2@amazon.com* with the email addresses you want to add to the contact list. Replace the `attributesData` values with the values specific to the contact. Additionally, replace *Sports* and *Cycling* with the `topicName` that applies to your contact. The acceptable `topicPreferences` are *OPT_IN* and *OPT_OUT*.

The following attributes are supported when uploading your contacts into an Amazon S3 object in either CSV or JSON format:

Attribute	Description
`emailAddress`	The contact's email address. This is a mandatory field.
`unsubscribeAll`	A boolean value status noting if the contact is unsubscribed from all contact list topics.
`topicPreferences`	The contact's preferences for being opted-in to or opted-out of topics.
`attributesData`	The attribute data attached to a contact.

- Give SES permission to read the Amazon S3 object.

  When applied to an Amazon S3 bucket, the following policy gives SES permission to read that bucket. For more information about attaching policies to Amazon S3 buckets, see Using Bucket Policies and User Policies in the *Amazon Simple Storage Service User Guide*.

- Give SES permission to use your AWS KMS key.

  If the Amazon S3 object is encrypted with an AWS KMS key, you need to give Amazon SES permission to use the KMS key. SES can only attain permission from a customer managed key, not a default KMS key. You must give SES permission to use the customer managed key by adding a statement to the key's policy.

  Paste the following policy statement into the key policy to permit SES to use your customer managed key.

```
{
   "Sid": "AllowSESToDecrypt",
   "Effect": "Allow",
   "Principal": {
       "Service":"ses.amazonaws.com"
   },
   "Action": [
       "kms:Decrypt",
   ],
   "Resource": "*"
}
```

- Use the `CreateImportJob` operation in the SES API v2.

> **Note**
>
> The following example assumes that you've already installed the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

At the command line, enter the following command. Replace *s3bucket* withe the name of the Amazon S3 bucket and *s3object* with the name of the Amazon S3 object name.

```
aws sesv2 create-import-job --import-destination
 ContactListDestination={ContactListName=ExampleContactListName,ContactListImportAction=PUT}
 --import-data-source S3Url="s3://s3bucket/s3object",DataFormat=CSV
```

List management walkthrough with examples

The following walkthrough provides examples of how you can use list management to list your contacts, utilize `ListManagementOptions` to specify a contact list and topic name in your email, and how to insert unsubscribe links.

1. **List contacts by using the AWS CLI** – You can use the `ListContacts` operation to retrieve a list of all your contacts who have subscribed to a particular topic, in conjunction with the `SendEmail` operation, which allows you to send them emails.

   At the command line, enter the following command:

```
aws sesv2 list-contacts --cli-input-json file://LIST-CONTACTS-JSON
```

In the preceding command, replace *LIST-CONTACTS-JSON* with the path to your JSON file for your `ListContacts` request.

An example `ListContacts` input JSON file for the request is as follows:

```
{
    "ContactListName": "ExampleContactListName",
    "Filter": {
        "FilteredStatus": "OPT_IN",
        "TopicFilter": {
            "TopicName": "Cycling",
            "UseDefaultIfPreferenceUnavailable": true
        }
    },
    "PageSize": 50
}
```

The `FilteredStatus` shows the subscription status for which you want to filter, which is either `OPT_IN` or `OPT_OUT`.

The `TopicFilter` is an optional filter which specifies which topic you want results for, and in the example above, that is "*Cycling*."

`UseDefaultIfPreferenceUnavailable` can have a value of `true` or `false`. If `true`, the topic default preference will be used if the contact doesn't have any explicit preference for a topic. If `false`, only contacts with an explicitly set preference are considered for filtering.

2. **Send mail with `ListManagementOptions` enabled** – After listing the contacts in your list using the above `ListContacts` operation, you can use the `SendEmail` operation to send emails to each of your contacts by utilizing the `ListManagementOptions` header to specify your contact list and topic name.

   To use `ListManagementOptions` with the `SendEmail` operation, include the `contactListName` and `topicName` to which the email belongs (the `topicName` is optional):

```
ListManagementOptions:
    String contactListName
```

```
    String topicName
```

If you include `ListManagementOptions` in your `SendEmail` request to a recipient email address that is not on your contact list, then a contact will be created on your list automatically.

SES will issue a bounce event for a message that is sent to an unsubscribed contact on your contact list, which means you won't need to update your `SendEmail` requests to avoid sending to contacts who have unsubscribed.

3. **Indicate the location for your unsubscribe links** – When utilizing `ListManagementOptions` you have the option to enable SES to add unsubscribe footer links in your email using the `{{amazonSESUnsubscribeUrl}}` placeholder to specify where SES needs to insert the unsubscribe URL. Placeholder replacement is supported only for HTML and TEXT content types. You can include the placeholder two times maximum. If used more than two times, only the first two occurrences are replaced. For more information, see Using subscription management.

   Alternatively, if you're using the SMTP interface to send email, you can use the `X-SES-LIST-MANAGEMENT-OPTIONS` header to specify a list and topic name.

   To specify a list and topic name while sending email using the SMTP interface, add the following email header to your message:

   ```
   X-SES-LIST-MANAGEMENT-OPTIONS: {contactListName}; topic={topicName}
   ```

Using subscription management

Amazon SES provides a subscription management capability, in which Amazon SES automatically enables the unsubscribe links in every outgoing email when you specify the `contactListName` and `topicName` within `ListManagementOptions` in the `SendEmail` operation request.

If a contact unsubscribes from a particular topic or list, Amazon SES does not allow email sending to the contact for that topic or list in the future.

Note

- Amazon SES subscription management supports *Bulk Sender Requirements* as enforced by many email service providers, see *Section 2* in An Overview of Bulk Sender Changes for more information.
- Subscription management is available for those using Easy DKIM in Amazon SES, but it's not possible for Amazon SES to add the unsubscribe links to your email for senders who are signing emails themselves before calling Amazon SES.

For information about list management and how to use it, including retrieving a list of all your contacts who have subscribed to a particular topic, see Using list management.

Subscription management overview

You should consider the following factors when you use subscription management:

- Subscription management will be fully managed by Amazon SES. This means that Amazon SES receives unsubscribe emails and requests from the unsubscribe webpage and then updates the contact's preference in your list. You can receive unsubscribe notifications using configuration set notifications. For more information about configuration sets, see Using configuration sets in Amazon SES.
- You need to specify the contact list while sending the email. Subscription management via the `List-Unsubscribe` header and `ListManagementOptions` footer links will be handled accordingly.
- Amazon SES adds support for the `List-Unsubscribe` header standards, which will enable email clients and inbox providers to display an unsubscribe link at the top of the email *if they support it* - not all email service providers support these headers.
- `List-Unsubscribe` headers follow the following behavior:
  - If a contact clicks the unsubscribe link in an email which has both the contact list and topic specified, then the contact will be unsubscribed only from that specific topic.
  - If the topic is not specified, then the contact will be unsubscribed from all the topics in the list.
- Contacts will be taken to an unsubscribe landing page when they click an unsubscribe link in the email footer.

- The unsubscribe landing page will give contacts an option to update their preferences, meaning `OPT_IN` or `OPT_OUT`, for all the topics in a particular list. The landing page also gives an option to unsubscribe from all topics in the list.
- If using `ListManagementOptions`, you must include the `{{amazonSESUnsubscribeUrl}}` placeholder in your emails to indicate where Amazon SES needs to insert the unsubscribe URL. You can include the placeholder two times maximum. If used more than two times, only the first two occurrences are replaced.
- The `List-Unsubscribe` header and `ListManagementOptions` footer links are added only if the email is being sent to a single recipient.
- For transactional emails where you don't want contacts to be able to unsubscribe, you can omit the `ListManagementOptions` field with your `SendEmail` request.

Unsubscribe header considerations

Subscription management through an unsubscribe link is enabled when the email contains the following headers:

`List-Unsubscribe`

`List-Unsubscribe-Post`

When you use Amazon SES's subscription management, `ListManagementOptions`, Amazon SES will override these headers if they are present in the email.

Recipients who unsubscribe by clicking the link produced by these headers will have a different experience depending on their email client or inbox provider because some providers do not recognize the `List-Unsubscribe` and `List-Unsubscribe-Post` headers; email sent to recipients using such providers will not see the Unsubscribe link.

Recipients whose email client recognizes these headers will see the Unsubscribe link and will be able to unsubscribe via the link but will not have the option of choosing which topics they unsubscribe from, and will simply be unsubscribed from the topic to which the email was sent.

For more information about the `List-Unsubscribe` header, see RFC 2369, and for the `List-Unsubscribe-Post` header, see RFC 8058.

> **Note**
>
> Amazon SES supports *one-click unsubscribe* in accordance with *Bulk Sender Requirements* as enforced by many email service providers, see Using one-click unsubscribe with Amazon SES for more information.

Adding an unsubscribe footer link

You will need to use the `{{amazonSESUnsubscribeUrl}}` placeholder in templated and non-templated emails to specify where Amazon SES needs to insert the unsubscribe URL.

Placeholder replacement is supported only for HTML and TEXT content types.

You can include the placeholder two times maximum. If used more than two times, only the first two occurrences are replaced.

> **Note**
>
> The `{{amazonSESUnsubscribeUrl}}` placeholder can only be used if `ListManagementOptions` is specified as a header while using the `SendEmail` operation or X-SES-LIST-MANAGEMENT-OPTIONS is specified as a header while using the SMTP interface. (Not to be confused with the `List-Unsubscribe` or `List-Unsubscribe-Post` headers which are not dependent on `ListManagementOptions` and can be used by themselves.)

Pricing plans

With Amazon Simple Email Service (SES) pricing plans, you get simpler billing, packaged features, and volume discounts. You choose a single plan that combines the capabilities you need at a lower overall cost. When your sending needs grow, you can upgrade your plan at any time.

Amazon SES offers three plans—Essentials, Pro, and Enterprise. Each plan includes everything in the plans before it, plus additional features. For more information about pricing and plan comparisons, see Amazon SES Pricing.

With a pricing plan, you know which feature charges your plan covers and which ones you pay for as optional add-ons. When you choose a plan, no features turn on automatically. You still enable or disable each feature individually, regardless of which plan you choose. Turn on and start using the features that your plan includes to get the most value from it.

Choose and manage your plan

You select and manage your pricing plan on the Amazon SES console **Pricing plan** page. Plans apply for each account and each AWS Region, so you manage your plan separately in each Region where you use Amazon SES.

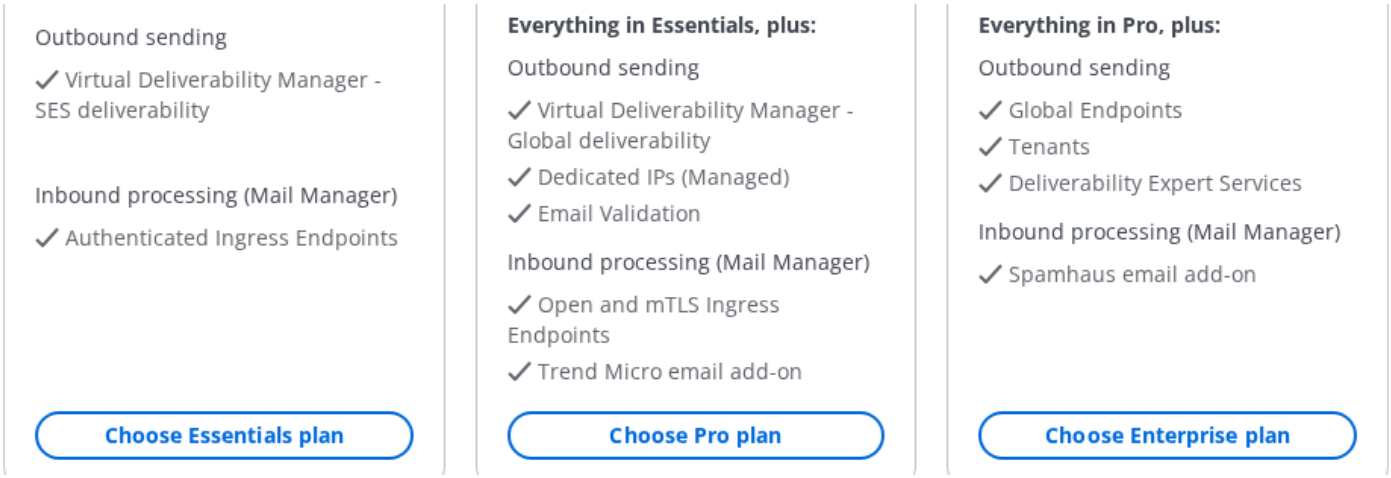
The **Pricing plan** page shows your current plan name, the AWS Region, and your next plan if you have a change scheduled.

Choosing a plan

If you do not have a plan, you are on à la carte pricing (pay-per-use pricing for individual features). To choose a plan, use one of the following methods:

- **Pricing plan page** – Open the **Pricing plan** page in the Amazon SES console, review the available plans (Essentials, Pro, and Enterprise), and choose the plan that fits your needs.
- **Get set up wizard** – New customers can choose a plan during the Amazon SES **Get set up** wizard in the **Select pricing plan** step.

The following screenshot shows the plan options and **Choose plan** buttons on the **Pricing plan** page.

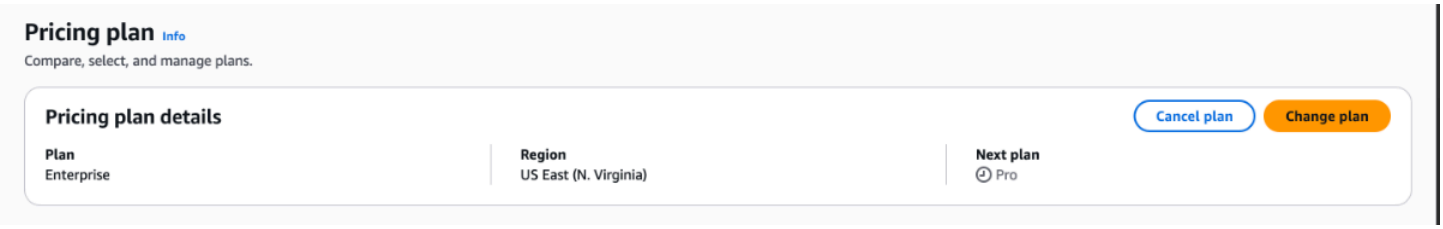


Changing or canceling your plan

You can change or cancel your plan from the **Pricing plan** page in the Amazon SES console. The page displays your **Plan** and **Region**, and shows a **Next plan** field when you have a scheduled plan change, indicating which plan your account moves to at the next billing cycle.

- **Change your plan** – Choose **Change plan** to switch to a different plan (Essentials, Pro, or Enterprise).
- **Cancel your plan** – Choose **Cancel plan** to return to à la carte pricing.

The following screenshot shows the **Pricing plan details** card.



When plan changes take effect

Upgrades take effect immediately when you submit the request. If you did not explicitly choose a plan and were defaulted to the Essentials plan, your first downgrade or cancellation to à la carte pricing also takes effect immediately. All other downgrades or cancellations take effect at the start of your next billing cycle.

Monitoring your Amazon SES sending activity

Amazon SES provides methods to monitor your sending activity using events, metrics, and statistics. An event is something that happens related to your sending activity that you've specified to be tracked as a metric. A metric represents a time-ordered set of data points representing the values of a monitored event type producing statistics. Statistics are metric data aggregations for a specified period of time including up to the present.

These monitoring methods assist you in keeping track of important measures, such as your account's bounce, complaint and reject rates. Excessively high bounce and complaint rates may jeopardize your ability to send emails using SES. These methods can also be used to measure the rates at which your customers engage with the emails you send by helping you to identify your overall open and click through rates utilizing event publishing and custom domains associated with configuration sets - see Configuring custom domains to handle open and click tracking.

The first step in setting up monitoring is to identify the types of email events related to your sending activity that you want to measure and monitor using SES. You can choose the following event types to monitor in SES:

- **Send** – The send request was successful and Amazon SES will attempt to deliver the message to the recipient's mail server. (If account-level or global suppression is being used, SES will still count it as a send, but delivery is suppressed.)
- **RenderingFailure** – The email wasn't sent because of a template rendering issue. This event type can occur when template data is missing, or when there is a mismatch between template parameters and data. (This event type only occurs when you send email using the `SendTemplatedEmail` or `SendBulkTemplatedEmail` API operations.)
- **Reject** – Amazon SES accepted the email, but determined that it contained a virus and didn't attempt to deliver it to the recipient's mail server.
- **Delivery** – Amazon SES successfully delivered the email to the recipient's mail server.
- **Bounce** – A *hard bounce* that the recipient's mail server permanently rejected the email. (*Soft bounces* are only included when SES is no longer retrying to deliver the email. Generally these soft bounces indicate a delivery failure, although in some cases a soft bounce can be returned even when the mail reaches the recipient inbox successfully. This typically occurs when the recipient sends an out-of-office automatic reply. Learn more about soft bounces in this AWS re:Post article.)

- **Complaint** – The email was successfully delivered to the recipient's mail server, but the recipient marked it as spam.
- **DeliveryDelay** – The email couldn't be delivered to the recipient's mail server because a temporary issue occurred. Delivery delays can occur, for example, when the recipient's inbox is full, or when the receiving email server experiences a transient issue.
- **Subscription** – The email was successfully delivered, but the recipient updated the subscription preferences by clicking `List-Unsubscribe` in the email header or the `Unsubscribe` link in the footer.
- **Open** – The recipient received the message and opened it in their email client.
- **Click** – The recipient clicked one or more links in the email.

You can monitor email sending events in several ways. The method you choose depends on the type of event you want to monitor, the granularity and level of detail you want to monitor it with, and the location where you want SES to publish the data. You're required to use either feedback notifications or event publishing to track bounce and complaint events. You can also choose to use multiple monitoring methods. The characteristics of each method are listed in the following table.

Monitoring Method	Events You Can Monitor	How to Access the Data	Level of Detail	Granularity
SES console	Account health, emails sent, quota used, successful send requests, rejects, bounces & complaints *(recent history to current reputation)*	Account dashboard page in the SES console	Count and percentage	Across entire AWS account
SES console	Account health, emails sent, bounces & complaint	Reputation metrics page in the SES console	Calculated rates only	Across entire AWS account

Monitoring Method	Events You Can Monitor	How to Access the Data	Level of Detail	Granularity
	s *(current reputation)*			
Virtual Deliverability Manager	Accounts statistics, ISP, sending identities, configuration sets, sent, delivered, complaints, transient & permanent bounces, opens & clicks, deliverability and reputation.	the section called “Dashboard” in the SES console the section called “Advisor” in the SES console	Count and percentage	Across entire AWS account
SES API	Deliveries, bounces, complaints, and rejects	`GetSendStatistics` API operation	Count only	Across entire AWS account

Monitoring Method	Events You Can Monitor	How to Access the Data	Level of Detail	Granularity
Amazon CloudWatch console	Sends, deliveries, opens, clicks, bounces, bounce rate, complaints, complaint rate, rejects, rendering failures, and blacklisted IPs.	CloudWatch console **Note** Some metrics don't appear in CloudWatch until the associated event occurs. For example, bounce metrics don't appear in CloudWatch until at least one email that you send bounces, or until you generate a	Count only	Across entire AWS account

Monitoring Method	Events You Can Monitor	How to Access the Data	Level of Detail	Granularity
		simulated bounce event by using the mailbox simulator.		
Feedback notifications	Deliveries, bounces, and complaints	Amazon SNS notification (deliveries, bounces, and complaints) or email (bounces and complaints only). See Setting up event notifications.	Details on each event	Across entire AWS account

Monitoring Method	Events You Can Monitor	How to Access the Data	Level of Detail	Granularity
Event publishing	Sends, deliveries, opens, clicks, bounces, complaints, rejects, and rendering failures.	Amazon CloudWatch or Amazon Data Firehose, or by Amazon SNS notification—see Monitor email sending using event publishing. *(Additional charges apply, see Price per metric for CloudWatch.)*	Details on each event	Fine-grained (based on user-definable email characteristics)
Event publishing utilizing custom domains associated with configuration sets - more info	Open and click tracking.	Amazon CloudWatch or Amazon Data Firehose, or by Amazon SNS notification. *(Additional charges apply, see Price per metric for CloudWatch.)*	Details on each event.	Fine-grained (based on user-definable email characteristics)

Note

The metrics measured by email sending events may not align perfectly with your sending quotas. This discrepancy can be caused by email bounces and rejections, or by using the

SES inbox simulator. To find out how close you are to your sending quotas, see Monitoring your sending quotas.

For information on how to use each monitoring method, see the following topics:

- Monitoring your sending statistics using the Amazon SES console
- Monitoring your usage statistics using the Amazon SES API
- Monitor email sending using Amazon SES event publishing

Monitoring your sending statistics using the Amazon SES console

From the Amazon SES console's **Account dashboard**, **Reputation metrics**, and **SMTP settings** pages, you can monitor all your email sending, usage, statistics, SMTP settings, overall account health, and reputation metrics. The following sections describe the metrics and statistics provided on each of these console pages.

It should be noted that while both the the section called “Account dashboard” and the section called “Reputation metrics” console pages contain bounce and complaint metrics, there is a subtle difference between these two sets of bounce and complaint rates as explained below:

- **Account dashboard page** – based on the date range selected, you can view what the bounce and complaint rates were in the past showing the metric progression of change leading up to the present time.
- **Reputation metrics page** – bounce and complaint rates based on the latest data point received from calculating your overall historic average at a high level (this shouldn’t be confused with your regular bounce/complaint rate, which corresponds to precise bounce/complaint events as they occur in real-time as shown on the **Account dashboard** page).

As a simple example to compare either the bounce or complaint rates between the **Reputation metrics** page and the **Account dashboard** page, let’s say the rate was 2% yesterday and is 1% now, on the **Reputation metrics** page, you'll only see the current rate of 1%, but on the **Account dashboard** page, the graphs will plot the charted progression showing a rate of 2% for yesterday and 1% for today.

Account dashboard

You can monitor the number of emails sent from your account, as well as the percentage of your sending quota that's been used, directly from the SES console's **Account dashboard** page in the *Daily email usage* pane. Delivery and rejection rates for your account can be monitored in the *Sending Statistics* pane, as well as other key factors related to your email sending in the following panes:

- **Sending limits** – contains the following quotas applicable to sending mail through SES:
  - *Daily sending quota* - maximum number of emails that you can send in a 24-hour period.
  - *Maximum send rate* - maximum number of emails that can be send from your account each second.
- **Account health** – the status of your SES account:
  - `Healthy` - there are no reputation-related issues that currently impact your account.
  - `Under review` - potential issues have been identified with your SES account - your account is under review while you work on correcting the issues.
  - `Paused` - your account's ability to send email is currently paused because of an issue with the email sent from your account. When the issue's been corrected, you can request that your account's ability to send email is resumed.
- **Daily email usage** – to check your daily usage to ensure you aren't approaching your sending limits:
  - *Emails sent* - total number of emails sent in a 24-hour period.
  - *Remaining sends* - total number of remaining emails available to be sent in a 24-hour period.
  - *Sending quota used* - percentage of your daily sending quota used.
- **Sending statistics** – comprised of graphs that show the progression of four essential metrics in a time-ordered set of data points representing the values of a monitored event type producing statistics for the selected date range using an aggregation period of *1 hour*. You can select a data range with start values from `Last 1 day` to `Last 14 days` to filter the charts below:
  - *Sends* - sum of successful email send requests for the date range selected.
  - *Rejects* - average rate of rejected send requests by SES based on `Rejects/Sends * 100` for the date range selected.
  - *Bounces* - average rate derived from your overall historic sender reputation metrics showing the progression for the date range selected.

- *Complaints* - average rate derived from your overall historic sender reputation metrics showing the progression for the date range selected.

Each of these charts contain a **View in CloudWatch** button that will open the respective metric in the Amazon CloudWatch console allowing detailed data to be viewed, customized metric math performed, and the creation of alarms in CloudWatch.

Reputation metrics

In addition to bounce and complaint rates, the **Reputation metrics** page also provides other high-level visibility into key factors affecting your reputation consisting of the following panes:

- **Summary** – provides an overview of your reputation health.
  - *Status* - overall reputation health based on historic bounce and complaint rates:
    - `Healthy` - both metrics are within normal levels.
    - `Under review` - one or both metrics have automatically caused your account to be placed under review.
    - `At risk` - one or both metrics have reached unhealthy levels and your account's ability to send email may be at risk.
  - *Emails sent (last 24 hours)* — the total number of emails sent in the last 24-hour period.
  - *Remaining sends* — total number of remaining emails available to be sent in a 24-hour period.
  - *Sending quota used* — percentage of your daily sending quota used.
- **Account-level tab contents:**
  - Bounce rate
    - *Status* - indicates the health of your bounce rate using the same values as described for the Summary pane.
    - *Historic bounce rate* - percentage of emails from your account that resulted in a hard bounce calculated from your overall historic average based on a representative volume that represents your typical sending practices.
  - Complaint rate
    - *Status* - Indicates the health of your complaint rate using the same values as described for the Summary pane.

    - *Historic bounce rate* - percentage of emails sent from your account that resulted in recipients reporting them as spam calculated from your overall historic average based on a representative volume that represents your typical sending practices.
- **Configuration set tab contents:**
    - Reputation by configuration set
        - *Configuration set* - lets you type or select a configuration set that have reputation metrics enabled so you can see summary, bounce, and complaint data based on the emails sent using the selected configuration set. The resulting panes that appear after selecting a configuration set are the same as described above for the Reputation metrics page except they are based only on email sent with the selected configuration set as apposed to your overall account-level sending metrics.

SMTP settings

This page lists the SMTP settings that are required for using the Amazon SES SMTP interface either through the SES API or programmatically, and provides links for creating and managing your SMTP credentials:

- **SMTP settings** – if you want to use an SMTP-enabled programming language, email server, or application to connect to the Amazon SES SMTP interface, the following information is provided:
    - SMTP endpoint
    - STARTTLS Port
    - Transport Layer Security (TLS)
    - TLS Wrapper Port
    - Authentication links provided for SMTP and IAM credential creation and management

Using the console to monitor send and reputation metrics

The following procedures will get you started in exploring your send and reputation metrics either using the **Account dashboard** page for metrics based on recent history (up to 14 days), or use the **Reputation metrics** page for metrics based on your overall history to the present time.

To view emails sent and sending quota used

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane, choose **Account dashboard**. Your usage statistics are shown in the **Daily email usage** section.

To view count of sends, rates of rejects, bounces, and complaints

1. In the navigation pane, choose **Account dashboard**.
2. In the **Sending statistics** section, use the **Date range** dropdown to select a start value for a date range to filter the four charts directly below the **Sending statistics** section.
3. Based on the date range selected, you can view what the counts and rates were in the past showing the metric progression of change leading up to the present time.
4. In any of the charts, choose the **View in CloudWatch** button to open the respective metric in the **Amazon CloudWatch** console where you can view detailed data, perform customized metric math, and create monitoring alarms in CloudWatch.

To view overall historic bounce and complaint rates

1. In the navigation pane, choose **Reputation metrics**.
2. In the **Bounce rate** pane you can view the percentage of emails sent from your account that resulted in a hard bounce, and in the **Complaint rate** pane you can view the percentage of emails sent from your account that resulted in recipients reporting them as spam; both metrics calculated from a representative volume of email based on your typical sending practices.
3. In either of the panes, choose the **View in CloudWatch** button to open the respective metric in the **Amazon CloudWatch** console where you can view detailed data, perform customized metric math, and create monitoring alarms in CloudWatch.

To view reputation metrics by configuration sets

1. In the navigation pane, choose **Reputation metrics**.
2. On the Reputation metrics page, select the **Configuration set** tab.
3. In the **Reputation by configuration set** pane, click inside the **Configuration set** field and either start typing for, or select, a configuration set that has reputation metrics enabled.

4. After selecting the configuration set, it will load the Summary, Bounce, and Complaint panes showing metrics based only on email sent with the selected configuration set.

Monitoring your usage statistics using the Amazon SES API

The Amazon SES API provides the `GetSendStatistics` operation, which returns information about your service usage. We recommend that you check your sending statistics regularly, so that you can make adjustments if needed.

When you call the `GetSendStatistics` operation, you receive a list of data points representing the last two weeks of your sending activity. Each data point in this list represents 15 minutes of activity and contains the following information for that period:

- The number of hard bounces
- The number of complaints
- The number of delivery attempts (corresponds to the number of emails you have sent)
- The number of rejected send attempts
- A timestamp for the analysis period

For a complete description of the `GetSendStatistics` operation, see the Amazon Simple Email Service API Reference.

In this section, you will find the following topics:

- the section called “Calling the `GetSendStatistics` API operation using the AWS CLI”
- the section called “Calling the `GetSendStatistics` operation programmatically”

Calling the `GetSendStatistics` API operation using the AWS CLI

The easiest way to call the `GetSendStatistics` API operation is to use the AWS Command Line Interface (AWS CLI).

To call the `GetSendStatistics` API operation using the AWS CLI

1. If you have not already done so, install the AWS CLI. For more information, see "Installing the AWS Command Line Interface" in the *AWS Command Line Interface User Guide*.

2. If you have not already done so, configure the AWS CLI to use your AWS credentials. For more information, see "Configuring the AWS CLI" in the *AWS Command Line Interface User Guide*.
3. At the command line, run the following command:

```
aws ses get-send-statistics
```

   If the AWS CLI is properly configured, you see a list of sending statistics in JSON format. Each JSON object includes aggregated sending statistics for a 15-minute period.

Calling the `GetSendStatistics` operation programmatically

You can also call the `GetSendStatistics` operation using the AWS SDKs. This section includes code examples for the AWS SDKs for Go, PHP, Python, and Ruby. Choose one of the following links to view code examples for that language:

- Code example for the AWS SDK for Go
- Code example for the AWS SDK for PHP
- Code example for the AWS SDK for Python (Boto)
- Code example for the AWS SDK for Ruby

> **Note**
>
> These code examples assume that you have created an AWS shared credentials file that contains your AWS access key ID, your AWS secret access key, and your preferred AWS Region. For more information, see Shared credentials and config files.

Calling `GetSendStatistics` using the AWS SDK for Go

```
package main

import (
    "fmt"

    //go get github.com/aws/aws-sdk-go/...
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
```

```
    "github.com/aws/aws-sdk-go/service/ses"
    "github.com/aws/aws-sdk-go/aws/awserr"
)

const (
    // Replace us-west-2 with the AWS Region you're using for Amazon SES.
    AwsRegion = "us-west-2"
)

func main() {

    // Create a new session and specify an AWS Region.
    sess, err := session.NewSession(&aws.Config{
        Region:aws.String(AwsRegion)},
    )

    // Create an SES client in the session.
    svc := ses.New(sess)
    input := &ses.GetSendStatisticsInput{}

    result, err := svc.GetSendStatistics(input)

    // Display error messages if they occur.
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            default:
                fmt.Println(aerr.Error())
            }
        } else {
            // Print the error, cast err to awserr.Error to get the Code and
            // Message from an error.
            fmt.Println(err.Error())
        }
        return
    }

    fmt.Println(result)
}
```

Calling `GetSendStatistics` using the AWS SDK for PHP

```
<?php
```

```
// Replace path_to_sdk_inclusion with the path to the SDK as described in
// http://docs.aws.amazon.com/aws-sdk-php/v3/guide/getting-started/basic-usage.html
define('REQUIRED_FILE','path_to_sdk_inclusion');

// Replace us-west-2 with the AWS Region you're using for Amazon SES.
define('REGION','us-west-2');

require REQUIRED_FILE;

use Aws\Ses\SesClient;

$client = SesClient::factory(array(
    'version'=> 'latest',
    'region' => REGION
));

try {
     $result = $client->getSendStatistics([]);
  echo($result);
} catch (Exception $e) {
     echo($e->getMessage()."\n");
}

?>
```

Calling `GetSendStatistics` using the AWS SDK for Python (Boto)

```
import boto3 #pip install boto3
import json
from botocore.exceptions import ClientError

client = boto3.client('ses')

try:
    response = client.get_send_statistics(
)
except ClientError as e:
    print(e.response['Error']['Message'])
else:
    print(json.dumps(response, indent=4, sort_keys=True, default=str))
```

Calling `GetSendStatistics` using the AWS SDK for Ruby

```
require 'aws-sdk' # gem install aws-sdk
require 'json'

# Replace us-west-2 with the AWS Region you're using for Amazon SES.
awsregion = "us-west-2"

# Create a new SES resource and specify a region
ses = Aws::SES::Client.new(region: awsregion)

begin

  resp = ses.get_send_statistics({
  })
  puts JSON.pretty_generate(resp.to_h)

# If something goes wrong, display an error message.
rescue Aws::SES::Errors::ServiceError => error
  puts error

end
```

Monitor email sending using Amazon SES event publishing

To enable you to track your email sending at a granular level, you can set up Amazon SES to publish *email sending events* to Amazon CloudWatch, Amazon Data Firehose, Amazon Pinpoint, Amazon Simple Notification Service, or Amazon EventBridge based on characteristics that you define.

You can track several types of email sending events, including sends, deliveries, opens, clicks, bounces, complaints, rejections, rendering failures, and delivery delays. This information can be useful for operational and analytical purposes. For example, you can publish your email sending data to CloudWatch and create dashboards that track the performance of your email campaigns, or you can use Amazon SNS to send you notifications when certain events occur.

How event publishing works with configuration sets and message tags

To use event publishing, you first set up one or more *configuration sets*. A configuration set specifies where to publish your events and which events to publish. Then, each time you send

an email, you provide the name of the configuration set and one or more *message tags*, in the form of name/value pairs, to categorize the email. For example, if you advertise books, you could name a message tag *genre*, and assign a value of *sci-fi* or *western*, when you send an email for the associated campaign.

Depending on which email sending interface you use, you either provide the message tag as a parameter to the `EmailTags` field of the `SendEmail` API operation or add the message tag to the SES-specific email header `X-SES-MESSAGE-TAGS`. For more information about configuration sets, see Using configuration sets in Amazon SES.

In addition to the message tags that you specify, SES also adds *auto-tags* to the messages you send. You do not need to perform any additional steps to use auto-tags.

The following table lists the auto-tags that are automatically applied to messages you send using SES.

SES Auto-Tags

Auto-tag name	Description
`ses:caller-identity`	The IAM identity of the SES user who sent the email.
`ses:configuration-set`	The name of the Configuration Set associated with the email.
`ses:from-domain`	The domain of the "From" address.
`ses:outgoing-ip`	The IP address that SES used to send the email.
`ses:source-ip`	The IP address that the caller used to send the email.
`ses:source-tls-version`	The TLS protocol version the caller used to send the email.
`ses:outgoing-tls-version`	The TLS protocol version that SES used to send the email.

Fine-grained feedback for email campaigns

The `ses:feedback-id-<a or b>` tag is an optional message tag that you can think of as a hybrid or semi-automatic tag—while it's similar to the auto-tags discussed in the previous section, the difference is that you must manually add it and use the `ses:` prefix key. You can use up to two of these tags defined as `ses:feedback-id-a` and `ses:feedback-id-b`.

When you specify these tags, SES automatically appends them to the standard `Feedback-ID` header which is used in providing delivery statistics, such as complaint and spam rates, as part of a feedback loop (FBL), see Feedback loops. The `Feedback-ID` header is comprised of the identifier, *SESInternalID*, used by SES for collecting complaint information, and the static tag, *AmazonSES*, identifying SES as the sending platform such as:

`FeedBackId:feedback-id-a:feedback-id-b:((SESInternalID):(AmazonSES))`

These optional feedback ID tags are offered as a way for you to generate fine-grained feedback, such as for messages you send as part of an email campaign. You can use `ses:feedback-id-<a or b>` by specifying it as a message tag in the `EmailTags` field of the `SendEmail` operation request as shown in the following example:

```
{
  "FromEmailAddress": "noreply@example.com",
  "Destination": {
    "ToAddresses": [
      "customer@example.net"
    ]
  },
  "Content": {
    "Simple": {
      "Subject": {
        "Data": "Hello and welcome"
      },
      "Body": {
        "Text": {
          "Data": "Lorem ipsum dolor sit amet."
        },
        "Html": {
          "Data": "Lorem ipsum dolor sit amet."
        }
      }
    }
```

```
  },
  "EmailTags": [
    {
      "Name": "ses:feedback-id-a",
      "Value": "new-members-campaign"
    },
    {
      "Name": "ses:feedback-id-b",
      "Value": "football-campaign"
    }
  ],
  "ConfigurationSetName": "football-club"
}
```

If sending in raw format, you would add `ses:feedback-id-<a or b>` as a message tag to the SES-specific header `X-SES-MESSAGE-TAGS`.

The `ses:feedback-id-<a or b>` message tag can also be tracked in Amazon CloudWatch by specifying it as a CloudWatch value source just like any other message tag, see the section called "Adding CloudWatch Event Destination Details" *(Additional charges apply, see Price per metric for CloudWatch.)*

How to use event publishing

The following sections contain the information you need to set up and use SES event publishing.

- Setting up event publishing
- Working with event data

Event publishing terminology

The following list defines terms related to SES event publishing.

Email sending event

Information associated with the outcome of an email you submit to SES. Sending events include the following:

- **Send** – The send request was successful and Amazon SES will attempt to deliver the message to the recipient's mail server. (If account-level or global suppression is being used, SES will still count it as a send, but delivery is suppressed.)

- **RenderingFailure** – The email wasn't sent because of a template rendering issue. This event type can occur when template data is missing, or when there is a mismatch between template parameters and data. (This event type only occurs when you send email using the `SendTemplatedEmail` or `SendBulkTemplatedEmail` API operations.)
- **Reject** – Amazon SES accepted the email, but determined that it contained a virus and didn't attempt to deliver it to the recipient's mail server.
- **Delivery** – Amazon SES successfully delivered the email to the recipient's mail server.
- **Bounce** – A *hard bounce* that the recipient's mail server permanently rejected the email. (*Soft bounces* are only included when SES is no longer retrying to deliver the email. Generally these soft bounces indicate a delivery failure, although in some cases a soft bounce can be returned even when the mail reaches the recipient inbox successfully. This typically occurs when the recipient sends an out-of-office automatic reply. Learn more about soft bounces in this AWS re:Post article.)
- **Complaint** – The email was successfully delivered to the recipient's mail server, but the recipient marked it as spam.
- **DeliveryDelay** – The email couldn't be delivered to the recipient's mail server because a temporary issue occurred. Delivery delays can occur, for example, when the recipient's inbox is full, or when the receiving email server experiences a transient issue.
- **Subscription** – The email was successfully delivered, but the recipient updated the subscription preferences by clicking `List-Unsubscribe` in the email header or the `Unsubscribe` link in the footer.
- **Open** – The recipient received the message and opened it in their email client.
- **Click** – The recipient clicked one or more links in the email.

Configuration set

A set of rules that defines the destination that SES publishes email sending events to, and the types of email sending events that you want to publish. When you send an email that you want to use with event publishing, you specify the configuration set to associate with the email.

Event destination

An AWS service that you publish SES email sending events to. Each event destination that you set up belongs to one, and only one, configuration set.

Message tag

A name/value pair that you use to categorize an email for the purpose of event publishing. Examples are *campaign/book* and *campaign/clothing*. When you send an email, you either specify the message tag as a parameter to the API call or as an SES-specific email header.

Auto-tag

Message tags that are automatically included in event publishing reports. There is an auto-tag for the configuration set name, the domain of the "From" address, the caller's outgoing IP address, the SES outgoing IP address, and the IAM identity of the caller.

Setting up Amazon SES event publishing

This section describes what you need to do to configure Amazon SES to publish your email sending events to the following AWS services:

- Amazon CloudWatch
- Amazon Data Firehose
- Amazon Pinpoint
- Amazon Simple Notification Service (Amazon SNS)

The following steps required for setting up event publishing are covered in the topics below:

1. You must create a *configuration set* using the Amazon SES console or API.
2. Add one or more *event destinations* (CloudWatch, Firehose, Pinpoint, or SNS) to the configuration set, and configure parameters unique to the event destination.
3. When you send an email, you specify which configuration set to use that contains your event destination.

Topics in this section

- Step 1: Create a configuration set
- Step 2: Add an event destination
- Step 3: Specify your configuration set when you send email

Step 1: Create a configuration set

You must first have a configuration set to set up event publishing. If you do not yet have a configuration set, or would like to create a new one, please see Creating configuration sets in SES

You can also create configuration sets using the CreateConfigurationSet operation in the Amazon SES API V2 or the Amazon SES CLI v2, see Create a configuration set (AWS CLI).

Step 2: Add an event destination

Event destinations are places that you publish Amazon SES events to. Each event destination that you set up belongs to one, and only one, configuration set. When you set up an event destination with Amazon SES, you choose the AWS service destination, and you specify parameters associated with that destination.

When you set up an event destination, you can choose to send events to one of the following AWS services:

- Amazon CloudWatch
- Amazon Data Firehose
- Amazon EventBridge
- Amazon Pinpoint
- Amazon Simple Notification Service (Amazon SNS)

The event destination that you choose depends on the level of detail you want about the events, and the way you want to receive the event information. If you simply want a running total of each type of event (for example, so that you can set an alarm when the total gets too high), you can use CloudWatch.

If you want detailed event records that you can output to another service such as Amazon OpenSearch Service or Amazon Redshift for analysis, you can use Firehose.

If you want to receive notifications when certain events occur, you can use Amazon SNS.

This section contains the following topics

- Set up a CloudWatch event destination for event publishing
- Set up a Data Firehose event destination for Amazon SES event publishing

- Set up an Amazon EventBridge destination for event publishing
- Set up an Amazon Pinpoint event destination for event publishing
- Set up an Amazon SNS event destination for event publishing

Set up a CloudWatch event destination for event publishing

With Amazon CloudWatch metrics, you can use event destinations to publish Amazon SES email sending events to CloudWatch. Because a CloudWatch event destination can only be set up in a configuration set, you must first create a configuration set and then add the event destination to the configuration set.

When you add a CloudWatch event destination to a configuration set, you must choose one or more CloudWatch *dimensions* that correspond to the message tags you use when you send your emails. Like message tags, a CloudWatch dimension is a name/value pair that helps you uniquely identify a metric.

For example, you might have a message tag and a dimension called `campaign` that you use to identify your email campaign. When you publish your email sending events to CloudWatch, choosing your message tags and dimensions is important because these choices affect your CloudWatch billing and determine how you can filter your email sending event data in CloudWatch.

This section provides information to help you choose your dimensions, and then shows how to add a CloudWatch event destination to a configuration set.

Topics in this section

- Adding a CloudWatch Event Destination
- Choosing CloudWatch Dimensions

Adding a CloudWatch Event Destination

The procedure in this section shows how to add CloudWatch event destination details to a configuration set and assumes you have completed steps 1 through 6 in Creating an event destination.

You can also use the UpdateConfigurationSetEventDestination operation in the Amazon SES API V2 to create and modify event destinations.

To add CloudWatch event destination details to a configuration set using the console

1. These are the detailed instructions for selecting CloudWatch as your event destination type in Step 7 and assumes you have completed all the previous steps in Creating an event destination. After selecting the CloudWatch **Destination type**, entering a destination **Name**, and enabling **Event publishing**, the **Amazon CloudWatch dimensions** pane is displayed—its fields are addressed in the following steps. *(Additional charges apply, see Price per metric for CloudWatch.)*
2. For **Value Source**, specify how Amazon SES will obtain the data that it passes to CloudWatch. The following value sources are available:

   - **Message Tag** – Amazon SES retrieves the dimension name and value from a tag that you specify by using the `X-SES-MESSAGE-TAGS` header or the `EmailTags` API parameter. For more information about using message tags, see the section called "Step 3: Specify your configuration set when sending".

     > **Note**
     >
     > Message tags can include the numbers 0–9, the letters A–Z (both uppercase and lowercase), hyphens (-), and underscores (_).

     You can also use the **Message Tag** value source to create dimensions based on Amazon SES auto-tags. To use an auto-tag, type the complete name of the auto-tag as the **Dimension Name**. For example, to create a dimension based on the configuration set auto-tag, use `ses:configuration-set` for the **Dimension Name**, and the name of the configuration set for the **Default Value**. For a complete list of auto-tags, see How event publishing works with configuration sets and message tags.
   - **Email Header** – Amazon SES retrieves the dimension name and value from a header in the email.

     > **Note**
     >
     > You can't use any of the following email headers as the **Dimension Name**: `Received`, `To`, `From`, `DKIM-Signature`, `CC`, `message-id`, or `Return-Path`.

- **Link Tag** – Amazon SES retrieves the dimension name and value from a tag that you specified in a link. For more information about adding tags to links, see Can I tag links with unique identifiers?.

3. For **Dimension Name**, type the name of the dimension that you want to pass to CloudWatch.

> **Note**
>
> Dimension names can contain only ASCII letters (a-z, A-Z), numbers (0-9), underscores (_), and dashes (-). Spaces, accented characters, non-Latin characters, and other special characters are not allowed.

4. For **Default Value**, type the value of the dimension.

> **Note**
>
> Dimension values can contain only ASCII letters (a-z, A-Z), numbers (0-9), underscores (_), dashes (-), at signs (@), and periods (.). Spaces, accented characters, non-Latin characters, and other special characters are not allowed.

5. If you want to add more dimensions, choose **Add Dimension**. Otherwise, choose **Next**.
6. On the review screen, if you're satisfied with how you defined your event destination, choose **Add destination**.

Choosing CloudWatch Dimensions

When you choose names and values to use as CloudWatch dimensions, consider the following factors:

- **Price per metric** – You can view basic Amazon SES metrics in CloudWatch for free. However, when you collect metrics using event publishing, you incur CloudWatch Detailed Monitoring costs. Each unique combination of event type, dimension name, and dimension value creates a different metric in CloudWatch. When you use CloudWatch, Detailed Monitoring, you are charged for each metric. For this reason, you might want to avoid choosing dimensions that can take many different values. For example, unless you are very interested in tracking your email sending events by "From" domain, you might not want to define a dimension for the Amazon SES auto-tag `ses:from-domain` because it can take many different values. For more information, see CloudWatch Pricing.

- **Metric filtering** – If a metric has multiple dimensions, you cannot access the metric in CloudWatch based on each dimension separately. For that reason, think carefully before you add more than one dimension to a single CloudWatch event destination. For example, if you want metrics by `campaign` and by a combination of `campaign` and `genre`, you need to add two event destinations: one with only `campaign` as a dimension, and one with both `campaign` and `genre` as dimensions.
- **Dimension value source** – As an alternative to specifying your dimension values using Amazon SES-specific headers or a parameter to the API, you can also choose for Amazon SES to take the dimension values from your own MIME message headers. You might use this option if you are already using custom headers and you do not want to change your emails or your calls to the email sending API to collect metrics based on your header values. If you use your own MIME message headers for Amazon SES event publishing, the header names and values that you use for Amazon SES event publishing may only include the letters A through Z, the numbers 0 through 9, underscores (_), at signs (@), hyphens (-), and periods (.). If you specify a name or value that contains other characters, the email sending call will still succeed, but the event metrics will not be sent to Amazon CloudWatch.

For more information about CloudWatch concepts, see Amazon CloudWatch Concepts in the *Amazon CloudWatch User Guide*.

Set up a Data Firehose event destination for Amazon SES event publishing

An Amazon Data Firehose event destination represents an entity that publishes specific Amazon SES email sending events to Firehose. Because a Firehose event destination can only be set up in a configuration set, you first have to create a configuration set. Next, you add the event destination to the configuration set.

The procedure in this section shows how to add Firehose event destination details to a configuration set and assumes you have completed steps 1 through 6 in Creating an event destination.

You can also use the UpdateConfigurationSetEventDestination operation in the Amazon SES API V2 destination to create and update event destinations.

To add Firehose event destination details to a configuration set using the console

1. These are the detailed instructions for selecting Firehose as your event destination type in Step 7 and assumes you have completed all the previous steps in Creating an event destination.

After selecting the Firehose **Destination type**, entering a destination **Name**, and enabling **Event publishing**, the **Amazon Data Firehose delivery stream** pane is displayed—its fields are addressed in the following steps.

2. For **Delivery stream**, choose an existing Firehose delivery stream, or choose **Create new stream** to create a new one using the Firehose console.

   For information about creating a stream using the Firehose console, see Creating an Amazon Kinesis Firehose Delivery Stream in the *Amazon Data Firehose Developer Guide*.

3. For **Identity and Access Management (IAM) Role**, choose an IAM role for which Amazon SES has permission to publish to Firehose on your behalf. You can choose an existing role, have Amazon SES create a role for you, or create your own role.

   If you choose an existing role or create your own role, you must manually modify the role's policies to give the role permission to access the Firehose delivery stream, and to give Amazon SES permission to assume the role. For example policies, see Giving Amazon SES Permission to Publish to Your Firehose Delivery Stream.

4. Choose **Next**.

5. On the review screen, if you're satisfied with how you defined your event destination, choose **Add destination**.

For information about how to use the `UpdateConfigurationSetEventDestination` API to add a Firehose event destination, see the Amazon Simple Email Service API Reference.

Giving Amazon SES Permission to Publish to Your Firehose Delivery Stream

To enable Amazon SES to publish records to your Firehose delivery stream, you must use an AWS Identity and Access Management (IAM) role and attach or modify the role's permissions policy and trust policy. The permissions policy enables the role to publish records to your Firehose delivery stream, and the trust policy enables Amazon SES to assume the role.

This section provides examples of both policies. For information about attaching policies to IAM roles, see Modifying a Role in the *IAM User Guide*.

Permissions Policy

The following permissions policy enables the role to publish data records to your Firehose delivery stream.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "firehose:PutRecordBatch"
            ],
            "Resource": [
                "arn:aws-eusc:firehose:eusc-de-
east-1:111122223333:deliverystream/delivery-stream-name"
            ]
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *delivery-region* with the AWS Region where you created the Firehose delivery stream.
- Replace *111122223333* with your AWS account ID.
- Replace *delivery-stream-name* with the name of the Firehose delivery stream.

Trust Policy

The following trust policy enables Amazon SES to assume the role.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "ses.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws-eusc:ses:delivery-
region:111122223333:configuration-set/configuration-set-name"
        }
      }
    }
  ]
}
```

Make the following changes to the preceding policy example:

- Replace *delivery-region* with the AWS Region where you created the Firehose delivery stream.
- Replace *111122223333* with your AWS account ID.
- Replace *configuration-set-name* with the name of your configuration set associated with the Firehose delivery stream.

Set up an Amazon EventBridge destination for event publishing

An Amazon EventBridge event destination notifies you about the email sending events you specify in a configuration set. SES generates and sends email sending events that you define when creating an event destination to the EventBridge default event bus. An event bus is a router that receives events and can deliver them to multiple destinations. You can learn more about integrating email sending events with Amazon EventBridge in *Monitoring using EventBridge*. Because an EventBridge event destination can only be set up in a configuration set, you have to create a configuration set before you add the event destination to the configuration set.

The procedure in this section shows how to add EventBridge event destination details to a configuration set and assumes you have completed steps 1 through 6 in Creating an event destination.

You can also use the UpdateConfigurationSetEventDestination operation in the Amazon SES API V2 to create and modify event destinations.

To add EventBridge event destination details to a configuration set using the console

1. These are the detailed instructions for selecting EventBridge as your event destination type in Step 7 and assumes you have completed all the previous steps in Creating an event destination. After selecting the *Amazon EventBridge* **Destination type**, entering a destination **Name**, and enabling **Event publishing**, an **Amazon EventBridge event bus** informational pane is displayed.
2. Choose **Next**.
3. On the review screen, if you're satisfied with how you defined your event destination, choose **Add destination**. This will open the event destination's summary page where a success banner will confirm if your event destination was created or modified successfully.

Set up an Amazon Pinpoint event destination for event publishing

An Amazon Pinpoint event destination notifies you about the email sending events you specify in a configuration set. Because an Amazon Pinpoint event destination can only be set up in a configuration set, you have to create a configuration set before you add the event destination to the configuration set.

The procedure in this section shows how to add Amazon Pinpoint event destination details to a configuration set and assumes you have completed steps 1 through 6 in Creating an event destination.

You can also use the UpdateConfigurationSetEventDestination operation in the Amazon SES API V2 to create and modify event destinations.

There are additional charges for the types of channels you have configured in your Amazon Pinpoint projects. For more information, see Amazon Pinpoint Pricing.

To add Amazon Pinpoint event destination details to a configuration set using the console

1. These are the detailed instructions for selecting Amazon Pinpoint as your event destination type in Step 7 and assumes you have completed all the previous steps in Creating an event destination.

   > **Note**
   >
   > Amazon Pinpoint does not support event types **Delivery delays** or **Subscriptions**.

After selecting the Amazon Pinpoint **Destination type**, entering a destination **Name**, and enabling **Event publishing**, the **Amazon Pinpoint project details** pane is displayed—its fields are addressed in the following steps.

2. For **Project**, choose an existing Amazon Pinpoint project, or choose **Create a new project in Amazon Pinpoint** to create a new one.

   For information about creating a project, see Create a project in the *Amazon Pinpoint User Guide*.

3. Choose **Next**.
4. On the review screen, if you're satisfied with how you defined your event destination, choose **Add destination**. This will open the event destination's summary page where a success banner will confirm if your event destination was created or modified successfully.

Set up an Amazon SNS event destination for event publishing

An Amazon SNS event destination notifies you about the email sending events you specify in a configuration set. Because an Amazon SNS event destination can only be set up in a configuration set, you have to create a configuration set before you add the event destination to the configuration set.

The procedure in this section shows how to add Amazon SNS event destination details to a configuration set and assumes you have completed steps 1 through 6 in Creating an event destination.

You can also use the UpdateConfigurationSetEventDestination operation in the Amazon SES API V2 to create and modify event destinations.

> **Note**
>
> Feedback notifications for bounces, complaints, and deliveries can also be set up through Amazon SNS for any of your verified sending identities. For more information, see. the section called “Configuring Amazon SNS notifications”.

There are additional charges for sending messages to the endpoints that are subscribed to your Amazon SNS topics. For more information, see Amazon SNS Pricing.

To add Amazon SNS event destination details to a configuration set using the console

1. These are the detailed instructions for selecting Amazon SNS as your event destination type in Step 7 and assumes you have completed all the previous steps in Creating an event destination. After selecting the Amazon SNS **Destination type**, entering a destination **Name**, and enabling **Event publishing**, the **Amazon Simple Notification Service (SNS) topic** pane is displayed—its fields are addressed in the following steps.
2. For **SNS topic**, choose an existing Amazon SNS topic, or choose **Create SNS topic** to create a new one.

   For information about creating a topic, see Create a Topic in the *Amazon Simple Notification Service Developer Guide*.

   > **Important**
   >
   > When you create your topic using Amazon SNS, for **Type**, only choose **Standard**. (SES does not support FIFO type topics.)

3. Choose **Next**.
4. On the review screen, if you're satisfied with how you defined your event destination, choose **Add destination**. This will open the event destination's summary page where a success banner will confirm if your event destination was created or modified successfully.
5. Whether you created a new SNS topic or selected an existing one, you will now need to give access to SES to publish notifications to the topic. On the event destination's summary page from the previous step, choose **Amazon SNS** from the **Destination type** column - this will take you to the **Topics** list in the Amazon Simple Notification Service console - *perform the following steps from the Amazon SNS console:*

   a. Select the name of the SNS topic you created or modified in the previous step.

   b. On the topic's detail screen, choose **Edit**.

   c. To give SES permission to publish notifications to the topic, on the **Edit topic** screen in the SNS console, expand **Access policy** and in the **JSON editor**, add the following permission policy:

      JSON

```
{
```

```
    "Version":"2012-10-17",
    "Id": "notification-policy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": "sns:Publish",
            "Resource": "arn:aws-eusc:sns:eusc-de-
east-1:111122223333:topic_name",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceAccount": "111122223333",
                    "AWS:SourceArn": "arn:aws-
eusc:ses:topic_region:111122223333:configuration-set/configuration-set-
name"
                }
            }
        }
    ]
}
```

Make the following changes to the preceding policy example:

- Replace *topic_region* with the AWS Region where you created the SNS topic.
- Replace *111122223333* with your AWS account ID.
- Replace *topic_name* with the name of your SNS topic.
- Replace *configuration-set-name* with the name of your configuration set associated with the SNS event destination.

d. Choose **Save changes**.

Step 3: Specify your configuration set when you send email

After you create a configuration set and add an event destination, the last step to event publishing is to send your emails.

To publish events associated with an email, you must provide the name of the configuration set to associate with the email. Optionally, you can provide message tags to categorize the email.

You provide this information to Amazon SES as either parameters to the email sending API, Amazon SES-specific email headers, or custom headers in your MIME message. The method you choose depends on which email sending interface you use, as shown in the following table.

Email Sending Interface	Ways to Publish Events
`SendEmail`	API parameters
`SendTemplatedEmail`	API parameters
`SendBulkTemplatedEmail`	API parameters
`SendCustomVerificationEmail`	API parameters
`SendRawEmail`	API parameters, Amazon SES-specific email headers, or custom MIME headers **Important** If you specify message tags using both headers and API parameters, Amazon SES uses only the message tags provided by the API parameter s. Amazon SES does not join message tags specified by API parameters and headers.
SMTP interface	Amazon SES-specific email headers

The following sections describe how to specify the configuration set and message tags using headers and using API parameters.

- Using Amazon SES API Parameters
- Using Amazon SES-Specific Email Headers
- Using Custom Email Headers

> **Note**
>
> You can optionally include message tags in the headers of your emails. Message tags can include the numbers 0–9, the letters A–Z (both uppercase and lowercase), hyphens (-), and underscores (_).

Using Amazon SES API Parameters

To use SendEmail, SendTemplatedEmail, SendBulkTemplatedEmail, SendCustomVerificationEmail, or SendRawEmail with event publishing, you specify the configuration set and the message tags by passing data structures called ConfigurationSet and MessageTag to the API call.

For more information about using the Amazon SES API, see the Amazon Simple Email Service API Reference.

Using Amazon SES-Specific Email Headers

When you use `SendRawEmail` or the SMTP interface, you can specify the configuration set and the message tags by adding Amazon SES-specific headers to the email. Amazon SES removes the headers before sending the email. The following table shows the names of the headers to use.

Event Publishing Information	Header
Configuration set	`X-SES-CONFIGURATION-SET`
Message tags	`X-SES-MESSAGE-TAGS`

The following example shows how the headers might look in a raw email that you submit to Amazon SES.

```
X-SES-MESSAGE-TAGS: tagName1=tagValue1, tagName2=tagValue2
X-SES-CONFIGURATION-SET: myConfigurationSet
From: sender@example.com
To: recipient@example.com
Subject: Subject
Content-Type: multipart/alternative;
 boundary="----=_boundary"
```

```
------=_boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

body
------=_boundary
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

body
------=_boundary--
```

Using Custom Email Headers

Although you must specify the configuration set name using the Amazon SES-specific header `X-SES-CONFIGURATION-SET`, you can specify the message tags by using your own MIME headers.

> **Note**
>
> Header names and values that you use for Amazon SES event publishing must be in ASCII. If you specify a non-ASCII header name or value for Amazon SES event publishing, the email sending call will still succeed, but the event metrics will not be emitted to Amazon CloudWatch.

Working with Amazon SES event data

After you set up event publishing and specify a configuration set for sending emails, you can retrieve your email sending events from the event destination that you specified when you set up the configuration set associated with the email.

This section describes how to retrieve your email sending events from Amazon CloudWatch and Amazon Data Firehose, and how to interpret event data provided by Amazon SNS.

- Retrieving Amazon SES event data from CloudWatch
- Retrieving Amazon SES event data from Firehose
- Interpreting Amazon SES event data from Amazon SNS

Retrieving Amazon SES event data from CloudWatch

Amazon SES can publish metrics for your email sending events to Amazon CloudWatch. When you publish event data to CloudWatch, it provides these metrics as an ordered set of time-series data. You can use these metrics to monitor the performance of your email sending. For example, you can monitor the complaint metric and set a CloudWatch alarm to trigger when the metric exceeds a certain value.

There are two levels of granularity at which Amazon SES can publish these events to CloudWatch:

- **Across your AWS account** – These coarse metrics, which correspond to the metrics you monitor using the Amazon SES console and the `GetSendStatistics` API, are totals across your entire AWS account. Amazon SES publishes these metrics to CloudWatch automatically.
- **Fine-grained** – These metrics are categorized by email characteristics that you define using *message tags*. To publish these metrics to CloudWatch, you have to set up event publishing with a CloudWatch event destination and specify a configuration set when you send an email. You can also specify message tags or use auto-tags that Amazon SES automatically provides.

This section describes the available metrics and how to view the metrics in CloudWatch.

Available Metrics

You can publish following Amazon SES email sending metrics to CloudWatch:

- **Send** – The send request was successful and Amazon SES will attempt to deliver the message to the recipient's mail server. (If account-level or global suppression is being used, SES will still count it as a send, but delivery is suppressed.)
- **RenderingFailure** – The email wasn't sent because of a template rendering issue. This event type can occur when template data is missing, or when there is a mismatch between template parameters and data. (This event type only occurs when you send email using the `SendTemplatedEmail` or `SendBulkTemplatedEmail` API operations.)
- **Reject** – Amazon SES accepted the email, but determined that it contained a virus and didn't attempt to deliver it to the recipient's mail server.
- **Delivery** – Amazon SES successfully delivered the email to the recipient's mail server.
- **Bounce** – A *hard bounce* that the recipient's mail server permanently rejected the email. (*Soft bounces* are only included when SES is no longer retrying to deliver the email. Generally these soft bounces indicate a delivery failure, although in some cases a soft bounce can be returned

even when the mail reaches the recipient inbox successfully. This typically occurs when the recipient sends an out-of-office automatic reply. Learn more about soft bounces in this AWS re:Post article.)

- **Complaint** – The email was successfully delivered to the recipient's mail server, but the recipient marked it as spam.
- **DeliveryDelay** – The email couldn't be delivered to the recipient's mail server because a temporary issue occurred. Delivery delays can occur, for example, when the recipient's inbox is full, or when the receiving email server experiences a transient issue.
- **Subscription** – The email was successfully delivered, but the recipient updated the subscription preferences by clicking `List-Unsubscribe` in the email header or the `Unsubscribe` link in the footer.
- **Open** – The recipient received the message and opened it in their email client.
- **Click** – The recipient clicked one or more links in the email.

Available Dimensions

CloudWatch uses the dimension names that you specify when you add a CloudWatch event destination to a configuration set in Amazon SES. For more information, see Set up a CloudWatch event destination for event publishing.

Viewing Amazon SES Metrics in the CloudWatch Console

The following procedure describes how to view your Amazon SES event publishing metrics using the CloudWatch console.

To view metrics using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. If necessary, change the region. From the navigation bar, select the region where your AWS resources reside. For more information, see Regions and Endpoints.
3. In the navigation pane, choose **All Metrics**.
4. In the **Metrics** pane, select **SES**.
5. Select the metric you want to view. To view fine-grained event publishing metrics, choose the combination of dimensions that you specified when you set up your CloudWatch

event destination. To learn more about viewing metrics with CloudWatch, see Use Amazon CloudWatch metrics.

To view metrics using the AWS CLI

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/SES"
```

Retrieving Amazon SES event data from Firehose

Amazon SES publishes email sending events to Firehose as JSON records. Firehose then publishes the records to the AWS service destination that you chose when you set up the delivery stream in Firehose. For information about setting up Firehose delivery streams, see Creating an Firehose Delivery Stream in the *Amazon Data Firehose Developer Guide*.

Topics in this section:

- Contents of event data that Amazon SES publishes to Firehose
- Examples of event data that Amazon SES publishes to Firehose

Contents of event data that Amazon SES publishes to Firehose

Amazon SES publishes email sending event records to Amazon Data Firehose in JSON format. When publishing events to Firehose, Amazon SES follows each JSON record with a newline character.

You can find example records for all of these notification types in Examples of event data that Amazon SES publishes to Firehose.

Topics in this section

- Top-level JSON object
- Mail object
- Bounce object
- Complaint object
- Delivery object
- Send object

- Reject object
- Open object
- Click object
- Rendering Failure object
- DeliveryDelay object
- Subscription object

Top-level JSON object

The top-level JSON object in an email sending event record contains the following fields.

Field Name	Description
`eventType`	A string that describes the type of event. Possible values: `Bounce`, `Complaint`, `Delivery`, `Send`, `Reject`, `Open`, `Click`, `Rendering Failure`, `DeliveryDelay`, or `Subscription`. If you did not set up event publishing this field is named `notificationType`.
`mail`	A JSON object that contains information about the email that produced the event.
`bounce`	This field is only present if `eventType` is `Bounce`. It contains information about the bounce.
`complaint`	This field is only present if `eventType` is `Complaint`. It contains information about the complaint.
`delivery`	This field is only present if `eventType` is `Delivery`. It contains information about the delivery.

Field Name	Description
send	This field is only present if eventType is Send.
reject	This field is only present if eventType is Reject. It contains information about the rejection.
open	This field is only present if eventType is Open. It contains information about the open event.
click	This field is only present if eventType is Click. It contains information about the click event.
failure	This field is only present if eventType is Rendering Failure . It contains informati on about the rendering failure event.
deliveryDelay	This field is only present if eventType is DeliveryDelay . It contains information about the delayed delivery of an email.
subscription	This field is only present if eventType is Subscription . It contains information about the subscription preferences.

Mail object

Each email sending event record contains information about the original email in the mail object. The JSON object that contains information about a mail object has the following fields.

Field Name	Description
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the message was sent.
`messageId`	A unique ID that Amazon SES assigned to the message. Amazon SES returned this value to you when you sent the message. **Note** This message ID was assigned by Amazon SES. You can find the message ID of the original email in the `headers` and `commonHeaders` fields of the `mail` object.
`source`	The email address that the message was sent from (the envelope MAIL FROM address).
`sourceArn`	The Amazon Resource Name (ARN) of the identity that was used to send the email. In the case of sending authorization, the `sourceArn` is the ARN of the identity that the identity owner authorized the delegate sender to use to send the email. For more information about sending authorization, see Email authentication methods.
`sendingAccountId`	The AWS account ID of the account that was used to send the email. In the case of sending authorization, the `sendingAccountId` is the delegate sender's account ID.
`destination`	A list of email addresses that were recipients of the original mail.

Field Name	Description
`headersTruncated`	A string that specifies whether the headers are truncated in the notification, which occurs if the headers are larger than 10 KB. Possible values are `true` and `false`.
`headers`	A list of the email's original headers. Each header in the list has a `name` field and a `value` field. **Note** Any message ID within the `headers` field is from the original message that you passed to Amazon SES. The message ID that Amazon SES subsequently assigned to the message is in the `messageId` field of the `mail` object.
`commonHeaders`	A mapping of the email's original, commonly used headers. **Note** Any message ID within the `commonHeaders` field is the message ID that Amazon SES subsequently assigned to the message in the `messageId` field of the `mail` object.
`tags`	A list of tags associated with the email.

Bounce object

The JSON object that contains information about a `Bounce` event will always have the following fields.

Field Name	Description
`bounceType`	The type of bounce, as determined by Amazon SES.
`bounceSubType`	The subtype of the bounce, as determined by Amazon SES.
`bouncedRecipients`	A list that contains information about the recipients of the original mail that bounced.
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the ISP sent the bounce notification.
`feedbackId`	A unique ID for the bounce.
`reportingMTA`	The value of the `Reporting-MTA` field from the DSN. This is the value of the Message Transfer Authority (MTA) that attempted to perform the delivery, relay, or gateway operation described in the DSN. **Note** This field only appears if a delivery status notification (DSN) was attached to the bounce.

Bounced recipients

A bounce event may pertain to a single recipient or to multiple recipients. The `bouncedRecipients` field holds a list of objects—one object per recipient to whom the bounce event pertains—and will always contain the following field.

Field Name	Description
`emailAddress`	The email address of the recipient. If a DSN is available, this is the value of the `Final-Recipient` field from the DSN.

Optionally, if a DSN is attached to the bounce, the following fields may also be present.

Field Name	Description
`action`	The value of the `Action` field from the DSN. This indicates the action performed by the reporting MTA as a result of its attempt to deliver the message to this recipient.
`status`	The value of the `Status` field from the DSN. This is the per-recipient transport-independent status code that indicates the delivery status of the message.
`diagnosticCode`	The status code issued by the reporting MTA. This is the value of the `Diagnostic-Code` field from the DSN. This field may be absent in the DSN (and therefore also absent in the JSON).

Bounce types

Each bounce event will be of one of the types shown in the following table.

The event publishing system only publishes hard bounces and soft bounces that will no longer be retried by Amazon SES. When you receive bounces marked `Permanent`, you should remove the corresponding email addresses from your mailing list; you will not be able to send to them in the future. `Transient` bounces are sent to you when a message has soft bounced several times, and Amazon SES has stopped trying to re-deliver it. You may be able to successfully resend to an address that initially resulted in a `Transient` bounce in the future.

bounceType	bounceSubType	Description
`Undetermined`	`Undetermined`	Amazon SES was unable to determine a specific bounce reason.
`Permanent`	`General`	Amazon SES received a general hard bounce. If you receive this type of bounce, you should remove the recipient's email address from your mailing list.
`Permanent`	`NoEmail`	Amazon SES received a permanent hard bounce because the target email address does not exist. If you receive this type of bounce, you should remove the recipient's email address from your mailing list.
`Permanent`	`Suppressed`	Amazon SES has suppressed sending to this address because it has a recent history of bouncing as an invalid address. To override the global suppression list, see Using the Amazon SES account-level suppression list.
`Permanent`	`OnAccountSuppressionList`	Amazon SES has suppressed sending to this address because it is on the account-level suppression list. This does not count toward your bounce rate metric.
`Permanent`	`EmailValidationSuppressed`	Amazon SES has suppressed sending to this address because the address did not meet the threshold for your email validation settings.

bounceType	bounceSubType	Description
		This does not count toward your bounce rate metric.
Permanent	OnTenantSuppressionList	Amazon SES has suppressed sending to this address because it is on the tenant-level suppression list. This does not count toward your bounce rate metric.
Transient	General	Amazon SES received a general bounce. You may be able to successfully send to this recipient in the future.
Transient	MailboxFull	Amazon SES received a mailbox full bounce. You may be able to successfully send to this recipient in the future.
Transient	MessageTooLarge	Amazon SES received a message too large bounce. You may be able to successfully send to this recipient if you reduce the size of the message.
Transient	CustomTimeoutExceeded	Amazon SES was not able to successfully deliver the email within the time specified by the email sender. *(The bounce message will specify the reason for any possible delivery attempt failures within the defined TTL.)*
Transient	ContentRejected	Amazon SES received a content rejected bounce. You may be able to successfully send to this recipient if you change the content of the message.
Transient	AttachmentRejected	Amazon SES received an attachment rejected bounce. You may be able to successfully send to this recipient if you remove or change the attachment.

Complaint object

The JSON object that contains information about a `Complaint` event has the following fields.

Field Name	Description
`complainedRecipients`	A list that contains information about recipients that may have submitted the complaint.
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the ISP sent the complaint notification.
`feedbackId`	A unique ID for the complaint.
`complaintSubType`	The subtype of the complaint, as determined by Amazon SES.

Further, if a feedback report is attached to the complaint, the following fields may be present.

Field Name	Description
`userAgent`	The value of the `User-Agent` field from the feedback report. This indicates the name and version of the system that generated the report.
`complaintFeedbackType`	The value of the `Feedback-Type` field from the feedback report received from the ISP. This contains the type of feedback.
`arrivalDate`	The value of the `Arrival-Date` or `Received-Date` field from the feedback report in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*). This field may be absent in the report (and therefore also absent in the JSON).

Complained recipients

The `complainedRecipients` field contains a list of recipients that may have submitted the complaint.

> **Important**
>
> Since most ISPs redact the email address of the recipient who submitted the complaint from their complaint notification, this list contains information about recipients who might have sent the complaint, based on the recipients of the original message and the ISP from which we received the complaint. Amazon SES performs a lookup against the original message to determine this recipient list.

JSON objects in this list contain the following field.

Field Name	Description
`emailAddress`	The email address of the recipient.

Complaint types

You may see the following complaint types in the `complaintFeedbackType` field as assigned by the reporting ISP, according to the Internet Assigned Numbers Authority website:

Field Name	Description
`abuse`	Indicates unsolicited email or some other kind of email abuse.
`auth-failure`	Email authentication failure report.
`fraud`	Indicates some kind of fraud or phishing activity.
`not-spam`	Indicates that the entity providing the report does not consider the message to be spam.

Field Name	Description
	This may be used to correct a message that was incorrectly tagged or categorized as spam.
`other`	Indicates any other feedback that does not fit into other registered types.
`virus`	Reports that a virus is found in the originating message.

Delivery object

The JSON object that contains information about a `Delivery` event will always have the following fields.

Field Name	Description
`timestamp`	The date and time when Amazon SES delivered the email to the recipient's mail server, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).
`processingTimeMillis`	The time in milliseconds between when Amazon SES accepted the request from the sender to when Amazon SES passed the message to the recipient's mail server.
`recipients`	A list of intended recipients that the delivery event applies to.
`smtpResponse`	The SMTP response message of the remote ISP that accepted the email from Amazon SES. This message will vary by email, by receiving mail server, and by receiving ISP.
`reportingMTA`	The host name of the Amazon SES mail server that sent the mail.

Field Name	Description
`remoteMtaIp`	The IP address of the MTA to which Amazon SES delivered the email.

Send object

The JSON object that contains information about a `send` event is always empty.

Reject object

The JSON object that contains information about a `Reject` event will always have the following fields.

Field Name	Description
`reason`	The reason the email was rejected. The only possible value is `Bad content`, which means that Amazon SES detected that the email contained a virus. When a message is rejected, Amazon SES stops processing it, and doesn't attempt to deliver it to the recipient's mail server.

Open object

The JSON object that contains information about a `Open` event will always contain the following fields.

Field Name	Description
`ipAddress`	The recipient's IP address.
`timestamp`	The date and time when the open event occurred in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).

Field Name	Description
`userAgent`	The user agent of the device or email client that the recipient used to open the email.
`isBotEvent`	The likelihood that the event was generated by an automated system (bot) rather than a human. Possible values: `Likely`, `Unlikely`.

Click object

The JSON object that contains information about a `Click` event will always contain the following fields.

Field Name	Description
`ipAddress`	The recipient's IP address.
`timestamp`	The date and time when the click event occurred in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).
`userAgent`	The user agent of the client that the recipient used to click a link in the email.
`link`	The URL of the link that the recipient clicked.
`linkTags`	A list of tags that were added to the link using the `ses:tags` attribute. For more information about adding tags to links in your emails, see Q5. Can I tag links with unique identifiers? in the Amazon SES email sending metrics FAQs.
`isBotEvent`	The likelihood that the event was generated by an automated system (bot) rather than a human. Possible values: `Likely`, `Unlikely`.

Rendering Failure object

The JSON object that contains information about a `Rendering Failure` event has the following fields.

Field Name	Description
`templateName`	The name of the template used to send the email.
`errorMessage`	A message that provides more information about the rendering failure.

DeliveryDelay object

The JSON object that contains information about a `DeliveryDelay` event has the following fields.

Field Name	Description
`delayType`	The type of delay. Possible values are: • **InternalFailure** – An internal Amazon SES issue caused the message to be delayed. • **General** – A generic failure occurred during the SMTP conversation. • **MailboxFull** – The recipient's mailbox is full and is unable to receive additional messages. • **SpamDetected** – The recipient's mail server has detected a large amount of unsolicited email from your account. • **RecipientServerError** – A temporary issue with the recipient's email server is preventin g the delivery of the message.

Field Name	Description
	• **IPFailure** – The IP address that's sending the message is being blocked or throttled by the recipient's email provider. • **TransientCommunicationFailure** – There was a temporary communication failure during the SMTP conversation with the recipient's email provider. • **BYOIPHostNameLookupUnavailable** – Amazon SES was unable to look up the DNS hostname for your IP addresses. This type of delay only occurs when you use Bring Your Own IP. • **Undetermined** – Amazon SES wasn't able to determine the reason for the delivery delay. • **SendingDeferral** – Amazon SES has deemed it appropriate to internally defer the message.
`delayedRecipients`	An object that contains information about the recipient of the email.
`expirationTime`	The date and time when Amazon SES will stop trying to deliver the message. This value is shown in ISO 8601 format.
`reportingMTA`	The IP address of the Message Transfer Agent (MTA) that reported the delay.
`timestamp`	The date and time when the delay occurred, shown in ISO 8601 format.

Delayed recipients

The `delayedRecipients` object contains the following values.

Field Name	Description
`emailAddress`	The email address that resulted in the delivery of the message being delayed.
`status`	The SMTP status code associated with the delivery delay.
`diagnosticCode`	The diagnostic code provided by the receiving Message Transfer Agent (MTA).

Subscription object

The JSON object that contains information about a `Subscription` event has the following fields.

Field Name	Description
`contactList`	The name of the list the contact is on.
`timestamp`	The date and time, in ISO8601 format *(YYYY-MM-DDThh:mm:ss.sZ)*, when the ISP sent the subscription notification.
`source`	The email address that the message was sent from (the envelope MAIL FROM address).
`newTopicPreferences`	A JSON data-structure (map) which specifies the subscription status of all the topics in the contact list indicating the status after a change (contact subscribed or unsubscribed).
`oldTopicPreferences`	A JSON data-structure (map) which specifies the subscription status of all the topics in the contact list indicating the status before the change (contact subscribed or unsubscribed).

New/old topic preferences

The `newTopicPreferences` and `oldTopicPreferences` objects contain the following values.

Field Name	Description
`unsubscribeAll`	Specifies if the contact unsubscribed from all the topics in the contact list.
`topicSubscriptionStatus`	Specifies the subscription status of the topic in the `topicName` field indicating whether it is currently subscribed to receive notifications from SES for the specified event type. Possible values are **OptIn** (subscribed) or **OptOut** (unsubscribed) in the `subscriptionStatus` field.
`topicDefaultSubscriptionStatus`	Specifies the default subscription status of the topic in the `topicName` field determining whether new topics added to the event destination will be subscribed or unsubscribed by default. Possible values are **OptIn** (subscribed by default) or **OptOut** (unsubscribed by default) in the `subscriptionStatus` field.

Examples of event data that Amazon SES publishes to Firehose

This section provides examples of the types of email sending event record that Amazon SES publishes to Firehose.

Topics in this section:

- Bounce record
- Complaint record
- Delivery record
- Send record
- Reject record

- Open record
- Click record
- Rendering Failure record
- DeliveryDelay record
- Subscription record

> **Note**
>
> In the following examples where a `tag` field is utilized, it is using event publishing through a configuration set for which SES supports the publishing of tags for all event types. If using feedback notifications directly on the identity, SES does not publish tags. Read about adding tags when creating a configuration set or modifying a configuration set.

Bounce record

The following is an example of a `Bounce` event record that Amazon SES publishes to Firehose.

```
{
  "eventType":"Bounce",
  "bounce":{
    "bounceType":"Permanent",
    "bounceSubType":"General",
    "bouncedRecipients":[
      {
        "emailAddress":"recipient@example.com",
        "action":"failed",
        "status":"5.1.1",
        "diagnosticCode":"smtp; 550 5.1.1 user unknown"
      }
    ],
    "timestamp":"2017-08-05T00:41:02.669Z",
    "feedbackId":"01000157c44f053b-61b59c11-9236-11e6-8f96-7be8aexample-000000",
    "reportingMTA":"dsn; mta.example.com"
  },
  "mail":{
    "timestamp":"2017-08-05T00:40:02.012Z",
    "source":"Sender Name <sender@example.com>",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
```

```
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "headers":[
      {
        "name":"From",
        "value":"Sender Name <sender@example.com>"
      },
      {
        "name":"To",
        "value":"recipient@example.com"
      },
      {
        "name":"Subject",
        "value":"Message sent from Amazon SES"
      },
      {
        "name":"MIME-Version",
        "value":"1.0"
      },
      {
        "name":"Content-Type",
        "value":"multipart/alternative; boundary=\"----
=_Part_7307378_1629847660.1516840721503\""
      }
    ],
    "commonHeaders":{
      "from":[
        "Sender Name <sender@example.com>"
      ],
      "to":[
        "recipient@example.com"
      ],
      "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject":"Message sent from Amazon SES"
    },
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ],
      "ses:source-ip":[
        "192.0.2.0"
```

```
      ],
      "ses:from-domain":[
        "example.com"
      ],
      "ses:caller-identity":[
        "ses_user"
      ]
    }
  }
}
```

Complaint record

The following is an example of a `Complaint` event record that Amazon SES publishes to Firehose.

```
{
  "eventType":"Complaint",
  "complaint": {
    "complainedRecipients":[
      {
        "emailAddress":"recipient@example.com"
      }
    ],
    "timestamp":"2017-08-05T00:41:02.669Z",
    "feedbackId":"01000157c44f053b-61b59c11-9236-11e6-8f96-7be8aexample-000000",
    "userAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/60.0.3112.90 Safari/537.36",
    "complaintFeedbackType":"abuse",
    "arrivalDate":"2017-08-05T00:41:02.669Z"
  },
  "mail":{
    "timestamp":"2017-08-05T00:40:01.123Z",
    "source":"Sender Name <sender@example.com>",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "headers":[
      {
        "name":"From",
        "value":"Sender Name <sender@example.com>"
```

```
      },
      {
        "name":"To",
        "value":"recipient@example.com"
      },
      {
        "name":"Subject",
        "value":"Message sent from Amazon SES"
      },
      {
        "name":"MIME-Version","value":"1.0"
      },
      {
        "name":"Content-Type",
        "value":"multipart/alternative; boundary=\"----
=_Part_7298998_679725522.1516840859643\""
      }
    ],
    "commonHeaders":{
      "from":[
        "Sender Name <sender@example.com>"
      ],
      "to":[
        "recipient@example.com"
      ],
      "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject":"Message sent from Amazon SES"
    },
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ],
      "ses:source-ip":[
        "192.0.2.0"
      ],
      "ses:from-domain":[
        "example.com"
      ],
      "ses:caller-identity":[
        "ses_user"
      ]
    }
  }
```

```
}
```

Delivery record

The following is an example of a `Delivery` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Delivery",
  "mail": {
    "timestamp": "2016-10-19T23:20:52.240Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "recipient@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "text/html; charset=UTF-8"
      },
      {
        "name": "Content-Transfer-Encoding",
        "value": "7bit"
      }
    ],
```

```
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "ses:outgoing-ip": [
        "192.0.2.0"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "delivery": {
    "timestamp": "2016-10-19T23:21:04.133Z",
    "processingTimeMillis": 11893,
    "recipients": [
      "recipient@example.com"
    ],
    "smtpResponse": "250 2.6.0 Message received",
    "remoteMtaIp": "123.456.789.012",
    "reportingMTA": "mta.example.com"
  }
```

```
}
```

Send record

The following is an example of a Send event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Send",
  "mail": {
    "timestamp": "2016-10-14T05:02:16.645Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "recipient@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/mixed;  boundary=\"----=_Part_0_716996660.1476421336341\""
      },
      {
        "name": "X-SES-MESSAGE-TAGS",
        "value": "myCustomTag1=myCustomTagValue1, myCustomTag2=myCustomTagValue2"
      }
    ],
```

```
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "send": {}
}
```

Reject record

The following is an example of a `Reject` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Reject",
  "mail": {
    "timestamp": "2016-10-14T17:38:15.211Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
```

```
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "sender@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/mixed; boundary=\"qMm9M+Fa2AknHoGS\""
      },
      {
        "name": "X-SES-MESSAGE-TAGS",
        "value": "myCustomTag1=myCustomTagValue1, myCustomTag2=myCustomTagValue2"
      }
    ],
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
```

```
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "reject": {
    "reason": "Bad content"
  }
}
```

Open record

The following is an example of an Open event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Open",
  "mail": {
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES",
      "to": [
        "recipient@example.com"
      ]
    },
    "destination": [
      "recipient@example.com"
    ],
```

```
    "headers": [
      {
        "name": "X-SES-CONFIGURATION-SET",
        "value": "ConfigSet"
      },
      {
        "name":"X-SES-MESSAGE-TAGS",
        "value":"myCustomTag1=myCustomValue1, myCustomTag2=myCustomValue2"
      },
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/alternative; boundary=\"XBoundary\""
      }
    ],
    "headersTruncated": false,
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "sendingAccountId": "123456789012",
    "source": "sender@example.com",
    "tags": {
      "myCustomTag1":[
        "myCustomValue1"
      ],
      "myCustomTag2":[
        "myCustomValue2"
      ],
      "ses:caller-identity": [
        "IAM_user_or_role_name"
      ],
```

```
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ]
    },
    "timestamp": "2017-08-09T21:59:49.927Z"
  },
  "open": {
    "ipAddress": "192.0.2.1",
    "timestamp": "2017-08-09T22:00:19.652Z",
    "userAgent": "Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac OS X)
 AppleWebKit/603.3.8 (KHTML, like Gecko) Mobile/14G60",
    "isBotEvent": "Unlikely"
  }
}
```

Click record

The following is an example of a `Click` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Click",
  "click": {
    "ipAddress": "192.0.2.1",
    "link": "http://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-email-
smtp.html",
    "linkTags": {
      "samplekey0": [
        "samplevalue0"
      ],
      "samplekey1": [
        "samplevalue1"
      ]
    },
    "timestamp": "2017-08-09T23:51:25.570Z",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/60.0.3112.90 Safari/537.36",
    "isBotEvent": "Likely"
  },
```

```
  "mail": {
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES",
      "to": [
        "recipient@example.com"
      ]
    },
    "destination": [
      "recipient@example.com"
    ],
    "headers": [
      {
        "name": "X-SES-CONFIGURATION-SET",
        "value": "ConfigSet"
      },
      {
        "name":"X-SES-MESSAGE-TAGS",
        "value":"myCustomTag1=myCustomValue1, myCustomTag2=myCustomValue2"
      },
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/alternative; boundary=\"XBoundary\""
      },
      {
```

```
        "name": "Message-ID",
        "value": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000"
      }
    ],
    "headersTruncated": false,
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "sendingAccountId": "123456789012",
    "source": "sender@example.com",
    "tags": {
      "myCustomTag1":[
        "myCustomValue1"
      ],
      "myCustomTag2":[
        "myCustomValue2"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ]
    },
    "timestamp": "2017-08-09T23:50:05.795Z"
  }
}
```

Rendering Failure record

The following is an example of a `Rendering Failure` event record that Amazon SES publishes to Firehose.

```
{
  "eventType":"Rendering Failure",
  "mail":{
    "timestamp":"2018-01-22T18:43:06.197Z",
    "source":"sender@example.com",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
```

```
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ]
    }
  },
  "failure":{
    "errorMessage":"Attribute 'attributeName' is not present in the rendering data.",
    "templateName":"MyTemplate"
  }
}
```

DeliveryDelay record

The following is an example of a `DeliveryDelay` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "DeliveryDelay",
  "mail":{
    "timestamp":"2020-06-16T00:15:40.641Z",
    "source":"sender@example.com",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ]
    }
  },
  "deliveryDelay": {
    "timestamp": "2020-06-16T00:25:40.095Z",
```

```
    "delayType": "TransientCommunicationFailure",
    "expirationTime": "2020-06-16T00:25:40.914Z",
    "delayedRecipients": [{
      "emailAddress": "recipient@example.com",
      "status": "4.4.1",
      "diagnosticCode": "smtp; 421 4.4.1 Unable to connect to remote host"
    }]
  }
}
```

Subscription record

The following is an example of a `Subscription` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Subscription",
  "mail": {
    "timestamp": "2022-01-12T01:00:14.340Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLEe4bccb684-777bc8de-afa7-4970-92b0-f515137b1497-000000",
    "destination": ["recipient@example.com"],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
```

```
        "name": "Content-Type",
        "value": "text/html; charset=UTF-8"
      },
      {
        "name": "Content-Transfer-Encoding",
        "value": "7bit"
      }
    ],
    "commonHeaders": {
      "from": ["sender@example.com"],
      "to": ["recipient@example.com"],
      "messageId": "EXAMPLEe4bccb684-777bc8de-afa7-4970-92b0-f515137b1497-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:operation": ["SendEmail"],
      "ses:configuration-set": ["ConfigSet"],
      "ses:source-ip": ["192.0.2.0"],
      "ses:from-domain": ["example.com"],
      "ses:caller-identity": ["ses_user"],
      "myCustomTag1": ["myCustomValue1"],
      "myCustomTag2": ["myCustomValue2"]
    }
  },
  "subscription": {
    "contactList": "ContactListName",
    "timestamp": "2022-01-12T01:00:17.910Z",
    "source": "UnsubscribeHeader",
    "newTopicPreferences": {
      "unsubscribeAll": true,
      "topicSubscriptionStatus": [
        {
          "topicName": "ExampleTopicName",
          "subscriptionStatus": "OptOut"
        }
      ]
    },
    "oldTopicPreferences": {
      "unsubscribeAll": false,
      "topicSubscriptionStatus": [
        {
          "topicName": "ExampleTopicName",
          "subscriptionStatus": "OptOut"
        }
```

```
      ]
    }
  }
}
```

Interpreting Amazon SES event data from Amazon SNS

Amazon SES publishes email sending events to Amazon Simple Notification Service (Amazon SNS) as JSON records. Amazon SNS then delivers notifications to the endpoints that are subscribed to the Amazon SNS topic associated with the event destination. For information about setting up topics and subscriptions in Amazon SNS, see Getting Started in the *Amazon Simple Notification Service Developer Guide*.

For a description of the record contents and for example records, see the following sections.

- Event record contents
- Event record examples

Contents of event data that Amazon SES publishes to Amazon SNS

Amazon SES publishes email sending event records to Amazon Simple Notification Service in JSON format.

You can find example records for all of these notification types in Examples of event data that Amazon SES publishes to Amazon SNS.

Topics in this section:

- Top-level JSON object
- Mail object
- Bounce object
- Complaint object
- Delivery object
- Send object
- Reject object
- Open object
- Click object

- Rendering Failure object
- DeliveryDelay object
- Subscription object

Top-level JSON object

The top-level JSON object in an email sending event record contains the following fields. The event type determines which other objects are present.

Field Name	Description
`eventType`	A string that describes the type of event. Possible values: `Bounce`, `Complaint`, `Delivery`, `Send`, `Reject`, `Open`, `Click`, `Rendering Failure`, `DeliveryDelay`, or `Subscription`. If you did not set up event publishing this field is named `notificationType`.
`mail`	A JSON object that contains information about the email that produced the event.
`bounce`	This field is only present if `eventType` is `Bounce`. It contains information about the bounce.
`complaint`	This field is only present if `eventType` is `Complaint`. It contains information about the complaint.
`delivery`	This field is only present if `eventType` is `Delivery`. It contains information about the delivery.
`send`	This field is only present if `eventType` is `Send`.

Field Name	Description
`reject`	This field is only present if `eventType` is `Reject`. It contains information about the rejection.
`open`	This field is only present if `eventType` is `Open`. It contains information about the open event.
`click`	This field is only present if `eventType` is `Click`. It contains information about the click event.
`failure`	This field is only present if `eventType` is `Rendering Failure`. It contains information about the rendering failure event.
`deliveryDelay`	This field is only present if `eventType` is `DeliveryDelay`. It contains information about the delayed delivery of an email.
`subscription`	This field is only present if `eventType` is `Subscription`. It contains information about the subscription preferences.

Mail object

Each email sending event record contains information about the original email in the `mail` object. The JSON object that contains information about a `mail` object has the following fields.

Field Name	Description
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the message was sent.

Field Name	Description
`messageId`	A unique ID that Amazon SES assigned to the message. Amazon SES returned this value to you when you sent the message. **Note** This message ID was assigned by Amazon SES. You can find the message ID of the original email in the `headers` and `commonHeaders` fields of the `mail` object.
`source`	The email address that the message was sent from (the envelope MAIL FROM address).
`sourceArn`	The Amazon Resource Name (ARN) of the identity that was used to send the email. In the case of sending authorization, the `sourceArn` is the ARN of the identity that the identity owner authorized the delegate sender to use to send the email. For more information about sending authorization, see Email authentication methods.
`sendingAccountId`	The AWS account ID of the account that was used to send the email. In the case of sending authorization, the `sendingAccountId` is the delegate sender's account ID.
`destination`	A list of email addresses that were recipients of the original mail.

Field Name	Description
`headersTruncated`	A string that specifies whether the headers are truncated in the notification, which occurs if the headers are larger than 10 KB. Possible values are `true` and `false`.
`headers`	A list of the email's original headers. Each header in the list has a `name` field and a `value` field. **Note** Any message ID within the `headers` field is from the original message that you passed to Amazon SES. The message ID that Amazon SES subsequently assigned to the message is in the `messageId` field of the `mail` object.
`commonHeaders`	A mapping of the email's original, commonly used headers. **Note** Any message ID within the `commonHeaders` field is the message ID that Amazon SES subsequently assigned to the message in the `messageId` field of the `mail` object.
`tags`	A list of tags associated with the email.

Bounce object

The JSON object that contains information about a `Bounce` event has the following fields.

Field Name	Description
`bounceType`	The type of bounce, as determined by Amazon SES.
`bounceSubType`	The subtype of the bounce, as determined by Amazon SES.
`bouncedRecipients`	A list that contains information about the recipients of the original mail that bounced.
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the ISP sent the bounce notification.
`feedbackId`	A unique ID for the bounce.
`reportingMTA`	The value of the `Reporting-MTA` field from the DSN. This is the value of the Message Transfer Authority (MTA) that attempted to perform the delivery, relay, or gateway operation described in the DSN. **Note** This field only appears if a delivery status notification (DSN) was attached to the bounce.

Bounced recipients

A bounce event may pertain to a single recipient or to multiple recipients. The `bouncedRecipients` field holds a list of objects—one object per recipient whose email address produced a bounce—and contains the following field.

Field Name	Description
`emailAddress`	The email address of the recipient. If a DSN is available, this is the value of the `Final-Recipient` field from the DSN.

Optionally, if a DSN is attached to the bounce, the following fields may also be present.

Field Name	Description
`action`	The value of the `Action` field from the DSN. This indicates the action performed by the reporting MTA as a result of its attempt to deliver the message to this recipient.
`status`	The value of the `Status` field from the DSN. This is the per-recipient transport-independent status code that indicates the delivery status of the message.
`diagnosticCode`	The status code issued by the reporting MTA. This is the value of the `Diagnostic-Code` field from the DSN. This field may be absent in the DSN (and therefore also absent in the JSON).

Bounce types

Each bounce event is of one of the types shown in the following table.

The event publishing system only publishes hard bounces and soft bounces that are no longer retried by Amazon SES. When you receive bounces marked `Permanent`, you should remove the corresponding email addresses from your mailing list; you will not be able to send to them in the future. `Transient` bounces are sent to you when a message has soft bounced several times, and Amazon SES has stopped trying to re-deliver it. You may be able to successfully resend to an address that initially resulted in a `Transient` bounce in the future.

bounceType	bounceSubType	Description
Undetermined	Undetermined	Amazon SES was unable to determine a specific bounce reason.
Permanent	General	Amazon SES received a general hard bounce. If you receive this type of bounce, you should remove the recipient's email address from your mailing list.
Permanent	NoEmail	Amazon SES received a permanent hard bounce because the target email address does not exist. If you receive this type of bounce, you should remove the recipient's email address from your mailing list.
Permanent	Suppressed	Amazon SES has suppressed sending to this address because it has a recent history of bouncing as an invalid address. To override the global suppression list, see Using the Amazon SES account-level suppression list.
Permanent	OnAccountSuppressionList	Amazon SES has suppressed sending to this address because it is on the account-level suppression list. This does not count toward your bounce rate metric.
Permanent	EmailValidationSuppressed	Amazon SES has suppressed sending to this address because the address did not meet the threshold for your email validation settings. This does not count toward your bounce rate metric.
Permanent	OnTenantSuppressionList	Amazon SES has suppressed sending to this address because it is on the tenant-level suppression list. This does not count toward your bounce rate metric.

bounceType	bounceSubType	Description
`Transient`	`General`	Amazon SES received a general bounce. You may be able to successfully send to this recipient in the future.
`Transient`	`MailboxFull`	Amazon SES received a mailbox full bounce. You may be able to successfully send to this recipient in the future.
`Transient`	`MessageTooLarge`	Amazon SES received a message too large bounce. You may be able to successfully send to this recipient if you reduce the size of the message.
`Transient`	`CustomTimeoutExceeded`	Amazon SES was not able to successfully deliver the email within the time specified by the email sender. *(The bounce message will specify the reason for any possible delivery attempt failures within the defined TTL.)*
`Transient`	`ContentRejected`	Amazon SES received a content rejected bounce. You may be able to successfully send to this recipient if you change the content of the message.
`Transient`	`AttachmentRejected`	Amazon SES received an attachment rejected bounce. You may be able to successfully send to this recipient if you remove or change the attachment.

Complaint object

The JSON object that contains information about a `Complaint` event has the following fields.

Field Name	Description
`complainedRecipients`	A list that contains information about recipients that may have submitted the complaint.
`timestamp`	The date and time, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*), when the ISP sent the complaint notification.
`feedbackId`	A unique ID for the complaint.
`complaintSubType`	The subtype of the complaint, as determined by Amazon SES.

Further, if a feedback report is attached to the complaint, the following fields may be present.

Field Name	Description
`userAgent`	The value of the `User-Agent` field from the feedback report. This indicates the name and version of the system that generated the report.
`complaintFeedbackType`	The value of the `Feedback-Type` field from the feedback report received from the ISP. This contains the type of feedback.
`arrivalDate`	The value of the `Arrival-Date` or `Received-Date` field from the feedback report in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*). This field may be absent in the report (and therefore also absent in the JSON).

Complained recipients

The `complainedRecipients` field contains a list of recipients that may have submitted the complaint.

> **Important**
>
> Most ISPs redact the email addresses of recipients who submit complaints. For this reason, the `complainedRecipients` field includes a list of everyone who was sent the email whose address is on the domain that issued the complaint notification.

JSON objects in this list contain the following field.

Field Name	Description
`emailAddress`	The email address of the recipient.

Complaint types

You may see the following complaint types in the `complaintFeedbackType` field as assigned by the reporting ISP, according to the Internet Assigned Numbers Authority website:

Field Name	Description
`abuse`	Indicates unsolicited email or some other kind of email abuse.
`auth-failure`	Email authentication failure report.
`fraud`	Indicates some kind of fraud or phishing activity.
`not-spam`	Indicates that the entity providing the report does not consider the message to be spam. This may be used to correct a message that was incorrectly tagged or categorized as spam.

Field Name	Description
`other`	Indicates any other feedback that does not fit into other registered types.
`virus`	Reports that a virus is found in the originating message.

Complaint subtypes

The value of the `complaintSubType` field can either be null or `OnAccountSuppressionList`. If the value is `OnAccountSuppressionList`, Amazon SES accepted the message, but didn't attempt to send it because it was on the account-level suppression list. The value can also be `OnTenantSuppressionList` if the address was on the tenant-level suppression list.

Delivery object

The JSON object that contains information about a `Delivery` event has the following fields.

Field Name	Description
`timestamp`	The date and time when Amazon SES delivered the email to the recipient's mail server, in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).
`processingTimeMillis`	The time in milliseconds between when Amazon SES accepted the request from the sender to when Amazon SES passed the message to the recipient's mail server.
`recipients`	A list of intended recipients that the delivery event applies to.
`smtpResponse`	The SMTP response message of the remote ISP that accepted the email from Amazon SES. This message will vary by email, by receiving mail server, and by receiving ISP.

Field Name	Description
`reportingMTA`	The host name of the Amazon SES mail server that sent the mail.
`remoteMtaIp`	The IP address of the MTA to which Amazon SES delivered the email.

Send object

The JSON object that contains information about a `send` event is always empty.

Reject object

The JSON object that contains information about a `Reject` event has the following fields.

Field Name	Description
`reason`	The reason the email was rejected. The only possible value is `Bad content`, which means that Amazon SES detected that the email contained a virus. When a message is rejected, Amazon SES stops processing it, and doesn't attempt to deliver it to the recipient's mail server.

Open object

The JSON object that contains information about a `Open` event has the following fields.

Field Name	Description
`ipAddress`	The recipient's IP address.
`timestamp`	The date and time when the open event occurred in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).

Field Name	Description
`userAgent`	The user agent of the device or email client that the recipient used to open the email.
`isBotEvent`	The likelihood that the event was generated by an automated system (bot) rather than a human. Possible values: `Likely`, `Unlikely`.

Click object

The JSON object that contains information about a `Click` event has the following fields.

Field Name	Description
`ipAddress`	The recipient's IP address.
`timestamp`	The date and time when the click event occurred in ISO8601 format (*YYYY-MM-DDThh:mm:ss.sZ*).
`userAgent`	The user agent of the client that the recipient used to click a link in the email.
`link`	The URL of the link that the recipient clicked.
`linkTags`	A list of tags that were added to the link using the `ses:tags` attribute. For more information about adding tags to links in your emails, see Q5. Can I tag links with unique identifiers? in the Amazon SES email sending metrics FAQs.
`isBotEvent`	The likelihood that the event was generated by an automated system (bot) rather than a human. Possible values: `Likely`, `Unlikely`.

Rendering Failure object

The JSON object that contains information about a `Rendering Failure` event has the following fields.

Field Name	Description
`templateName`	The name of the template used to send the email.
`errorMessage`	A message that provides more information about the rendering failure.

DeliveryDelay object

The JSON object that contains information about a `DeliveryDelay` event has the following fields.

Field Name	Description
`delayType`	The type of delay. Possible values are: • **InternalFailure** – An internal Amazon SES issue caused the message to be delayed. • **General** – A generic failure occurred during the SMTP conversation. • **MailboxFull** – The recipient's mailbox is full and is unable to receive additional messages. • **SpamDetected** – The recipient's mail server has detected a large amount of unsolicited email from your account. • **RecipientServerError** – A temporary issue with the recipient's email server is preventin g the delivery of the message.

Field Name	Description
	• **IPFailure** – The IP address that's sending the message is being blocked or throttled by the recipient's email provider. • **TransientCommunicationFailure** – There was a temporary communication failure during the SMTP conversation with the recipient's email provider. • **BYOIPHostNameLookupUnavailable** – Amazon SES was unable to look up the DNS hostname for your IP addresses. This type of delay only occurs when you use Bring Your Own IP. • **Undetermined** – Amazon SES wasn't able to determine the reason for the delivery delay. • **SendingDeferral** – Amazon SES has deemed it appropriate to internally defer the message.
`delayedRecipients`	An object that contains information about the recipient of the email.
`expirationTime`	The date and time when Amazon SES will stop trying to deliver the message. This value is shown in ISO 8601 format.
`reportingMTA`	The IP address of the Message Transfer Agent (MTA) that reported the delay.
`timestamp`	The date and time when the delay occurred, shown in ISO 8601 format.

Delayed recipients

The `delayedRecipients` object contains the following values.

Field Name	Description
`emailAddress`	The email address that resulted in the delivery of the message being delayed.
`status`	The SMTP status code associated with the delivery delay.
`diagnosticCode`	The diagnostic code provided by the receiving Message Transfer Agent (MTA).

Subscription object

The JSON object that contains information about a `Subscription` event has the following fields.

Field Name	Description
`contactList`	The name of the list the contact is on.
`timestamp`	The date and time, in ISO8601 format *(YYYY-MM-DDThh:mm:ss.sZ)*, when the ISP sent the subscription notification.
`source`	The email address that the message was sent from (the envelope MAIL FROM address).
`newTopicPreferences`	A JSON data-structure (map) which specifies the subscription status of all the topics in the contact list indicating the status after a change (contact subscribed or unsubscribed).
`oldTopicPreferences`	A JSON data-structure (map) which specifies the subscription status of all the topics in the contact list indicating the status before the change (contact subscribed or unsubscribed).

New/old topic preferences

The `newTopicPreferences` and `oldTopicPreferences` objects contain the following values.

Field Name	Description
`unsubscribeAll`	Specifies if the contact unsubscribed from all the topics in the contact list.
`topicSubscriptionStatus`	Specifies the subscription status of the topic in the `topicName` field indicating whether it is currently subscribed to receive notifications from SES for the specified event type. Possible values are **OptIn** (subscribed) or **OptOut** (unsubscribed) in the `subscriptionStatus` field.
`topicDefaultSubscriptionStatus`	Specifies the default subscription status of the topic in the `topicName` field determining whether new topics added to the event destination will be subscribed or unsubscribed by default. Possible values are **OptIn** (subscribed by default) or **OptOut** (unsubscribed by default) in the `subscriptionStatus` field.

Examples of event data that Amazon SES publishes to Amazon SNS

This section provides examples of the types of email sending event records that Amazon SES publishes to Amazon SNS.

Topics in this section:

- Bounce record
- Complaint record
- Delivery record
- Send record
- Reject record

- Open record
- Click record
- Rendering Failure record
- DeliveryDelay record
- Subscription record

> **Note**
>
> In the following examples where a `tag` field is utilized, it is using event publishing through a configuration set for which SES supports the publishing of tags for all event types. If using feedback notifications directly on the identity, SES does not publish tags. Read about adding tags when creating a configuration set or modifying a configuration set.

Bounce record

The following is an example of a `Bounce` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType":"Bounce",
  "bounce":{
    "bounceType":"Permanent",
    "bounceSubType":"General",
    "bouncedRecipients":[
      {
        "emailAddress":"recipient@example.com",
        "action":"failed",
        "status":"5.1.1",
        "diagnosticCode":"smtp; 550 5.1.1 user unknown"
      }
    ],
    "timestamp":"2017-08-05T00:41:02.669Z",
    "feedbackId":"01000157c44f053b-61b59c11-9236-11e6-8f96-7be8aexample-000000",
    "reportingMTA":"dsn; mta.example.com"
  },
  "mail":{
    "timestamp":"2017-08-05T00:40:02.012Z",
    "source":"Sender Name <sender@example.com>",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
```

```
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "headers":[
      {
        "name":"From",
        "value":"Sender Name <sender@example.com>"
      },
      {
        "name":"To",
        "value":"recipient@example.com"
      },
      {
        "name":"Subject",
        "value":"Message sent from Amazon SES"
      },
      {
        "name":"MIME-Version",
        "value":"1.0"
      },
      {
        "name":"Content-Type",
        "value":"multipart/alternative; boundary=\"----
=_Part_7307378_1629847660.1516840721503\""
      }
    ],
    "commonHeaders":{
      "from":[
        "Sender Name <sender@example.com>"
      ],
      "to":[
        "recipient@example.com"
      ],
      "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject":"Message sent from Amazon SES"
    },
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ],
      "ses:source-ip":[
        "192.0.2.0"
```

```
      ],
      "ses:from-domain":[
        "example.com"
      ],
      "ses:caller-identity":[
        "ses_user"
      ]
    }
  }
}
```

Complaint record

The following is an example of a `Complaint` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType":"Complaint",
  "complaint": {
    "complainedRecipients":[
      {
        "emailAddress":"recipient@example.com"
      }
    ],
    "timestamp":"2017-08-05T00:41:02.669Z",
    "feedbackId":"01000157c44f053b-61b59c11-9236-11e6-8f96-7be8aexample-000000",
    "userAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/60.0.3112.90 Safari/537.36",
    "complaintFeedbackType":"abuse",
    "arrivalDate":"2017-08-05T00:41:02.669Z"
  },
  "mail":{
    "timestamp":"2017-08-05T00:40:01.123Z",
    "source":"Sender Name <sender@example.com>",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "headers":[
      {
```

```
        "name":"From",
        "value":"Sender Name <sender@example.com>"
      },
      {
        "name":"To",
        "value":"recipient@example.com"
      },
      {
        "name":"Subject",
        "value":"Message sent from Amazon SES"
      },
      {
        "name":"MIME-Version","value":"1.0"
      },
      {
        "name":"Content-Type",
        "value":"multipart/alternative; boundary=\"----
=_Part_7298998_679725522.1516840859643\""
      }
    ],
    "commonHeaders":{
      "from":[
        "Sender Name <sender@example.com>"
      ],
      "to":[
        "recipient@example.com"
      ],
      "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject":"Message sent from Amazon SES"
    },
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ],
      "ses:source-ip":[
        "192.0.2.0"
      ],
      "ses:from-domain":[
        "example.com"
      ],
      "ses:caller-identity":[
        "ses_user"
      ]
    }
```

```
  }
}
```

Delivery record

The following is an example of a `Delivery` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType": "Delivery",
  "mail": {
    "timestamp": "2016-10-19T23:20:52.240Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "recipient@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "text/html; charset=UTF-8"
      },
      {
        "name": "Content-Transfer-Encoding",
```

```
        "value": "7bit"
      }
    ],
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "ses:outgoing-ip": [
        "192.0.2.0"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "delivery": {
    "timestamp": "2016-10-19T23:21:04.133Z",
    "processingTimeMillis": 11893,
    "recipients": [
      "recipient@example.com"
    ],
    "smtpResponse": "250 2.6.0 Message received",
```

```
    "remoteMtaIp": "123.456.789.012",
    "reportingMTA": "mta.example.com"
  }
}
```

Send record

The following is an example of a Send event record that Amazon SES publishes to Amazon SNS. Some fields are not always present. For example, with a templated email, the subject is rendered later and included in subsequent events.

```
{
  "eventType": "Send",
  "mail": {
    "timestamp": "2016-10-14T05:02:16.645Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "recipient@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/mixed;  boundary=\"----=_Part_0_716996660.1476421336341\""
```

```
      },
      {
        "name": "X-SES-MESSAGE-TAGS",
        "value": "myCustomTag1=myCustomTagValue1, myCustomTag2=myCustomTagValue2"
      }
    ],
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "send": {}
}
```

Reject record

The following is an example of a `Reject` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType": "Reject",
  "mail": {
    "timestamp": "2016-10-14T17:38:15.211Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination": [
      "sender@example.com"
    ],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/mixed; boundary=\"qMm9M+Fa2AknHoGS\""
      },
      {
        "name": "X-SES-MESSAGE-TAGS",
        "value": "myCustomTag1=myCustomTagValue1, myCustomTag2=myCustomTagValue2"
      }
    ],
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "to": [
```

```
        "recipient@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "myCustomTag1": [
        "myCustomTagValue1"
      ],
      "myCustomTag2": [
        "myCustomTagValue2"
      ]
    }
  },
  "reject": {
    "reason": "Bad content"
  }
}
```

Open record

The following is an example of an Open event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType": "Open",
  "mail": {
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES",
```

```
      "to": [
        "recipient@example.com"
      ]
    },
    "destination": [
      "recipient@example.com"
    ],
    "headers": [
      {
        "name": "X-SES-CONFIGURATION-SET",
        "value": "ConfigSet"
      },
      {
        "name":"X-SES-MESSAGE-TAGS",
        "value":"myCustomTag1=myCustomValue1, myCustomTag2=myCustomValue2"
      },
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/alternative; boundary=\"XBoundary\""
      }
    ],
    "headersTruncated": false,
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "sendingAccountId": "123456789012",
    "source": "sender@example.com",
    "tags": {
      "myCustomTag1":[
        "myCustomValue1"
```

```
      ],
      "myCustomTag2":[
        "myCustomValue2"
      ],
      "ses:caller-identity": [
        "IAM_user_or_role_name"
      ],
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ]
    },
    "timestamp": "2017-08-09T21:59:49.927Z"
  },
  "open": {
    "ipAddress": "192.0.2.1",
    "timestamp": "2017-08-09T22:00:19.652Z",
    "userAgent": "Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac OS X)
 AppleWebKit/603.3.8 (KHTML, like Gecko) Mobile/14G60",
    "isBotEvent": "Unlikely"
  }
}
```

Click record

The following is an example of a `Click` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType": "Click",
  "click": {
    "ipAddress": "192.0.2.1",
    "link": "http://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-email-
smtp.html",
    "linkTags": {
      "samplekey0": [
        "samplevalue0"
      ],
      "samplekey1": [
        "samplevalue1"
```

```
      ]
    },
    "timestamp": "2017-08-09T23:51:25.570Z",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/60.0.3112.90 Safari/537.36",
    "isBotEvent": "Likely"
  },
  "mail": {
    "commonHeaders": {
      "from": [
        "sender@example.com"
      ],
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "subject": "Message sent from Amazon SES",
      "to": [
        "recipient@example.com"
      ]
    },
    "destination": [
      "recipient@example.com"
    ],
    "headers": [
      {
        "name": "X-SES-CONFIGURATION-SET",
        "value": "ConfigSet"
      },
      {
        "name":"X-SES-MESSAGE-TAGS",
        "value":"myCustomTag1=myCustomValue1, myCustomTag2=myCustomValue2"
      },
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
      },
      {
        "name": "MIME-Version",
```

```
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "multipart/alternative; boundary=\"XBoundary\""
      },
      {
        "name": "Message-ID",
        "value": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000"
      }
    ],
    "headersTruncated": false,
    "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "sendingAccountId": "123456789012",
    "source": "sender@example.com",
    "tags": {
      "myCustomTag1":[
        "myCustomValue1"
      ],
      "myCustomTag2":[
        "myCustomValue2"
      ],
      "ses:caller-identity": [
        "ses_user"
      ],
      "ses:configuration-set": [
        "ConfigSet"
      ],
      "ses:from-domain": [
        "example.com"
      ],
      "ses:source-ip": [
        "192.0.2.0"
      ]
    },
    "timestamp": "2017-08-09T23:50:05.795Z"
  }
}
```

Rendering Failure record

The following is an example of a `Rendering Failure` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType":"Rendering Failure",
  "mail":{
    "timestamp":"2018-01-22T18:43:06.197Z",
    "source":"sender@example.com",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "tags":{
      "ses:configuration-set":[
        "ConfigSet"
      ]
    }
  },
  "failure":{
    "errorMessage":"Attribute 'attributeName' is not present in the rendering data.",
    "templateName":"MyTemplate"
  }
}
```

DeliveryDelay record

The following is an example of a `DeliveryDelay` event record that Amazon SES publishes to Amazon SNS.

```
{
  "eventType": "DeliveryDelay",
  "mail":{
    "timestamp":"2020-06-16T00:15:40.641Z",
    "source":"sender@example.com",
    "sourceArn":"arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId":"123456789012",
    "messageId":"EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
    "destination":[
      "recipient@example.com"
    ],
    "headersTruncated":false,
    "tags":{
      "ses:configuration-set":[
```

```
        "ConfigSet"
      ]
    }
  },
  "deliveryDelay": {
    "timestamp": "2020-06-16T00:25:40.095Z",
    "delayType": "TransientCommunicationFailure",
    "expirationTime": "2020-06-16T00:25:40.914Z",
    "delayedRecipients": [{
      "emailAddress": "recipient@example.com",
      "status": "4.4.1",
      "diagnosticCode": "smtp; 421 4.4.1 Unable to connect to remote host"
    }]
  }
}
```

Subscription record

The following is an example of a `Subscription` event record that Amazon SES publishes to Firehose.

```
{
  "eventType": "Subscription",
  "mail": {
    "timestamp": "2022-01-12T01:00:14.340Z",
    "source": "sender@example.com",
    "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
    "sendingAccountId": "123456789012",
    "messageId": "EXAMPLEe4bccb684-777bc8de-afa7-4970-92b0-f515137b1497-000000",
    "destination": ["recipient@example.com"],
    "headersTruncated": false,
    "headers": [
      {
        "name": "From",
        "value": "sender@example.com"
      },
      {
        "name": "To",
        "value": "recipient@example.com"
      },
      {
        "name": "Subject",
        "value": "Message sent from Amazon SES"
```

```
      },
      {
        "name": "MIME-Version",
        "value": "1.0"
      },
      {
        "name": "Content-Type",
        "value": "text/html; charset=UTF-8"
      },
      {
        "name": "Content-Transfer-Encoding",
        "value": "7bit"
      }
    ],
    "commonHeaders": {
      "from": ["sender@example.com"],
      "to": ["recipient@example.com"],
      "messageId": "EXAMPLEe4bccb684-777bc8de-afa7-4970-92b0-f515137b1497-000000",
      "subject": "Message sent from Amazon SES"
    },
    "tags": {
      "ses:operation": ["SendEmail"],
      "ses:configuration-set": ["ConfigSet"],
      "ses:source-ip": ["192.0.2.0"],
      "ses:from-domain": ["example.com"],
      "ses:caller-identity": ["ses_user"],
      "myCustomTag1": ["myCustomValue1"],
      "myCustomTag2": ["myCustomValue2"]
    }
  },
  "subscription": {
    "contactList": "ContactListName",
    "timestamp": "2022-01-12T01:00:17.910Z",
    "source": "UnsubscribeHeader",
    "newTopicPreferences": {
      "unsubscribeAll": true,
      "topicSubscriptionStatus": [
        {
          "topicName": "ExampleTopicName",
          "subscriptionStatus": "OptOut"
        }
      ]
    },
    "oldTopicPreferences": {
```

```
      "unsubscribeAll": false,
      "topicSubscriptionStatus": [
        {
          "topicName": "ExampleTopicName",
          "subscriptionStatus": "OptOut"
        }
      ]
    }
  }
}
```

Monitoring your Amazon SES sender reputation

Amazon SES actively tracks several metrics that may cause your reputation as a sender to be damaged, or that could cause your email delivery rates to decline. Two important metrics that we consider in this process are the bounce and complaint rates for your account. If the bounce or complaint rates for your account are too high, we might place your account under review or pause your account's ability to send email.

Because your bounce and complaint rate are so important to the health of your account, Amazon SES includes a reputation metrics page in the Amazon SES console that you can use to track these metrics. Reputation metrics can also display information about factors unrelated to bounces or complaints that could damage your sender reputation. For example, if you send email to a known spamtrap, you will see a message on this dashboard.

This section contains information about accessing reputation metrics, interpreting the information it contains, and setting up systems to actively notify you of factors that could impact your sender reputation.

In this section, you will find the following topics:

- Using reputation metrics to track bounce and complaint rates
- Reputation metrics messages
- Creating reputation monitoring alarms using CloudWatch
- SNDS metrics for dedicated IPs
- Automatically pausing email sending

Using reputation metrics to track bounce and complaint rates

The reputation metrics console page contains the same information that the Amazon SES team sees when determining the health of individual accounts.

To view reputation metrics

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane on the left side of the screen, choose **Reputation metrics**.

   The dashboard displays the following information:

- **Account status** – A summary of the combined health of your bounce and complaint rates. Possible values include:
  - **Healthy** – There are no issues currently impacting your account.
  - **Under review** – Your account is under review. If the issues that caused us to place your account under review aren't resolved by the end of the review period, we might pause your account's ability to send email.
  - **Pending end of review decision** – Your account is under review. Because of the nature of the issues that caused us to place your account under review, we need to perform a manual review of your account before we take any further action.
  - **Sending paused** – We've paused your account's ability to send email. While your account's ability to send email is paused, you won't be able to send email using Amazon SES. You can request that we review this decision. To learn more about requesting a review, see Amazon SES Sending review process FAQs.
  - **Pending sending pause** – Your account is under review. The issues that caused us to place your account under review haven't been resolved. In this situation, we typically pause your account's ability to send email. However, because of the nature of your account, we need to review your account before any further action is taken.
- **Bounce Rate** – The percentage of emails sent from your account that resulted in a hard bounce. See how your bounce rate's calculated.
- **Complaint Rate** – The percentage of emails sent from your account that resulted in recipients reporting them as spam. See how your complaint rate's calculated

> **Note**
>
> The **Bounce Rate** and **Complaint Rate** sections also include status messages for their respective metrics. The following is a list of status messages that may be displayed for these metrics:
>
> - **Healthy** – The metric is within normal levels.
> - **Almost healed** – The metric caused your account to be placed under review. Since the review period began, the metric has stayed below the maximum rate. If the metric remains below the maximum rate, the status of this metric changes to **Healthy** before the review period ends.
> - **Under review** – The metric caused your account to be placed under review, and is still above the maximum rate. If the issue that caused the metric to exceed the

maximum rate is not resolved by the end of the review period, we might pause your account's ability to send email.

- **Sending pause** – The metric caused us to pause your account's ability to send email. While your account's ability to send email is paused, you can't send email using Amazon SES. You can request that we review this decision. To learn more about submitting a request for review, see Amazon SES Sending review process FAQs.
- **Pending sending pause** – The metric caused us to place your account under review. The issues that caused this review period haven't been resolved. These issues might cause us to pause your account's ability to send email. A member of the Amazon SES team has to review your account before we take any further action.

- *Other Notifications* – If your account is experiencing reputation-related issues that are not related to bounces or complaints, a brief message will be shown here. For more information about the notifications that can be shown in this area, see Reputation metrics messages.

Reputation metrics messages

The Amazon SES Reputation metrics console page provides important metrics related to your account. The following sections describe the messages that might be displayed in this dashboard, and provide tips and information that you might be able to use to resolve issues related to your sender reputation.

This section contains information about the following types of notifications:

- Status Messages
- Bounce Rate Notification
- Complaint Rate Notification
- Anti-Spam Organization Notification
- Listbombing Notification
- Direct Feedback Notification
- Domain Blocklist Notification
- Internal Review Notification
- Mailbox Provider Notification

- Recipient Feedback Notification
- Related Account Notification
- Spamtrap Notification
- Vulnerable Site Notification
- Compromised Credentials Notification
- Other Notification

Status Messages

When you use the reputation metrics console page, you see a message describing the status of your Amazon SES account. The following is a list of possible account status values:

- **Healthy** – There are no issues currently impacting your account.
- **Under review** – Your account is under review. If the issues that caused us to place your account under review aren't resolved by the end of the review period, we might pause your account's ability to send email.
- **Pending end of review decision** – Your account is under review. Because of the nature of the issues that caused us to place your account under review, we need to perform a manual review of your account before we take any further action.
- **Sending paused** – We've paused your account's ability to send email. While your account's ability to send email is paused, you won't be able to send email using Amazon SES. You can request that we review this decision. To learn more about requesting a review, see Amazon SES Sending review process FAQs.
- **Pending sending pause** – Your account is under review. The issues that caused us to place your account under review haven't been resolved. In this situation, we typically pause your account's ability to send email. However, because of the nature of your account, we need to review your account before any further action is taken.

Additionally, the **Bounce Rate** and **Complaint Rate** sections of the reputation metrics page display status summaries for their respective metrics. The following is a list of possible metric status values:

- **Healthy** – The metric is within normal levels.

- **Almost healed** – The metric caused your account to be placed under review. Since the review period began, the metric has stayed below the maximum rate. If the metric remains below the maximum rate, the status of this metric changes to **Healthy** before the review period ends.
- **Under review** – The metric caused your account to be placed under review, and is still above the maximum rate. If the issue that caused the metric to exceed the maximum rate is not resolved by the end of the review period, we might pause your account's ability to send email.
- **Sending pause** – The metric caused us to pause your account's ability to send email. While your account's ability to send email is paused, you can't send email using Amazon SES. You can request that we review this decision. To learn more about submitting a request for review, see Amazon SES Sending review process FAQs.
- **Pending sending pause** – The metric caused us to place your account under review. The issues that caused this review period haven't been resolved. These issues might cause us to pause your account's ability to send email. A member of the Amazon SES team has to review your account before we take any further action.

Bounce Rate Notification

This section contains additional information about bounce rate notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

You received this notification because the bounce rate for your account was too high. The bounce rate is based on the number of hard bounces generated by your Amazon SES account. Email providers interpret a high bounce rate as a sign that a sender isn't properly managing their recipient list, and that the sender might be sending unsolicited email.

A hard bounce occurs when an email is sent to an address that doesn't exist. Amazon SES doesn't consider soft bounces (which occur when a recipient's address is temporarily unable to receive messages) in this calculation. Bounced emails that you send to verified addresses and domains, as well as emails that you send to the Amazon SES inbox simulator, also aren't considered in this calculation.

We calculate your bounce rate based on a *representative volume* of email. A representative volume is an amount of email that represents your typical sending practices. To be fair to both high- and low-volume senders, the representative volume is different for each account and changes as the account's sending patterns change.

For best results, maintain a bounce rate below 5%. Higher bounce rates can impact the delivery of your emails. If your bounce rate is 5% or greater, we automatically place your account under review. If your bounce rate is 10% or greater, we might pause your account's ability to send additional email until you resolve the issue that caused the high bounce rate.

What you can do to resolve the issue

If you haven't done so already, put a process in place to capture and manage bounces and complaints. All Amazon SES accounts are required to have these processes in place. For more information, see Email program success metrics.

Next, determine which email addresses are bouncing, and create and implement a plan for reducing or eliminating these bounces. If your account's ability to send email has already been paused, sign into the AWS Management Console and go to AWS Support. Reply to the case we opened on your behalf.

If your account is under review

At the end of the review period, if the bounce rate for your account remains above 10%, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your response to the case, describe the changes you implemented. If we agree that the changes will reduce your bounce rate, we adjust our calculations to only consider bounces received after your changes were implemented.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you implement changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Complaint Rate Notification

This section contains additional information about complaint rate notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

You received this notification because the complaint rate for your account was too high. The complaint rate is based on the number of complaints generated by your Amazon SES account. Email providers interpret a high complaint rate as a sign that a sender isn't properly managing their recipient list, and that the sender might be sending unsolicited email.

A complaint occurs when a recipient identifies an email that you sent as spam. This usually occurs when the recipient uses the Report Spam button in their email client. Complaints that are generated by emails that you send to the Amazon SES inbox simulator aren't considered in this calculation.

We calculate your complaint rate based on a *representative volume* of email. A representative volume is an amount of email that represents your typical sending practices. To be fair to both high- and low-volume senders, the representative volume is different for each account and changes as the account's sending patterns change.

For best results, maintain a complaint rate below 0.1%. Higher complaint rates can impact the delivery of your emails. If your complaint rate is 0.1% or greater, we automatically place your account under review. If your complaint rate is 0.5% or greater, we might pause your account's ability to send additional email until you resolve the issue that caused the high complaint rate.

What you can do to resolve the issue

If you haven't done so already, put a process in place to capture and manage bounces and complaints. All Amazon SES accounts are required to have these processes in place. For more information, see Email program success metrics.

Next, determine which messages you are sending that result in complaints, and implement a plan for reducing these complaints. If your account's ability to send email has already been paused, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf

While you should immediately stop sending to addresses that have complained, it is important that you identify the factors that are causing recipients to issue complaints. After you identify these factors, adjust your email sending behaviors to address them.

If your account is under review

At the end of the review period, if the complaint rate for your account remains above 0.5%, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your response to the case, describe the changes you implemented. If we agree that the changes will reduce your complaint rate, we adjust our calculations to only consider the complaints that were received after you implemented the changes.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Anti-Spam Organization Notification

This section contains additional information about anti-spam organization notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A reputable anti-spam organization has reported that some of the content being sent from your Amazon SES account has been flagged as unsolicited or problematic by their systems.

We're unable to provide information about the specific messages that caused the anti-spam organization to flag your content as problematic. We can't provide the name of the organization that issued the report. Typically, anti-spam organizations consider a combination of the following factors: recipient feedback, message engagement metrics, attempted deliveries to invalid addresses, content that is flagged by their spam filters, and spamtrap hits. This isn't an exhaustive list; other factors might cause these organizations to flag your content.

What you can do to resolve the issue

To resolve this issue, you need to determine what aspects of your email sending program might be causing the anti-spam organization to flag your email as problematic. You then need to change your sending program to address those issues.

If your account is under review

At the end of the review period, if the anti-spam organization continues to identify the email sent from your account as problematic, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we will extend the review period to ensure that we're only analyzing the anti-spam organization notifications we have received after you implemented your changes. At the end of this extended review period, your account is no longer listed by the anti-spam organization, we will remove the review period for your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Listbombing Notification

This section contains additional information about Listbombing notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

An anti-spam organization has identified that your email-sending processes are vulnerable to "listbombing." Listbombing is a form of abuse in which an attacker registers a very large number

of email addresses on a web-based form. Listbombing can result in service disruptions for users of impacted email services. It can also result in your email being blocked by email providers.

Anti-spam organizations use proprietary methods to identify sites that are vulnerable to listbombing. For this reason, we can't provide additional details about the issue that led the anti-spam organization to identify your email-sending process as problematic. We also can't share the name of the organization that identified the issue.

What you can do to resolve the issue

You should examine all of your web-based sign-up forms to ensure that they aren't vulnerable to this kind of abuse. Every form should include a CAPTCHA to prevent automated scripts from submitting subscription requests. Additionally, when new users sign up for your product or service, send them an email to confirm that they did, in fact, intend to sign up. Don't send any additional email to customers unless they explicitly opt in to your communications.

Finally, you should perform a "permission pass" on your email list. In a permission pass, you send an email to all of your customers asking them if they still want to receive email from you. Only send email to customers who verify that they want to continue to receive email from you.

If your account is under review

At the end of the review period, if the anti-spam organization continues to identify the email sent from your account as problematic, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we will extend the review period to ensure that we're only analyzing the anti-spam organization notifications we have received after you implemented your changes. At the end of this extended review period, your account is no longer listed by the anti-spam organization, we will remove the review period for your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details

of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Direct Feedback Notification

This section contains additional information about direct feedback notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A significant number of users have contacted Amazon SES directly to report messages that they received from an address or domain associated with your Amazon SES account. This type of feedback isn't visible in the complaints reported by mailbox providers directly, and isn't included in the bounce and complaint metrics shown on the reputation metrics page.

To protect the privacy of the users who reported these issues, we can't provide their email addresses.

Recipients can complain to Amazon SES when they receive messages that they didn't sign up to receive, when they don't receive the type of mail they expected to receive, when they don't find the email they receive to be useful or interesting, when they don't recognize that the messages are something that they signed up for, or when they are receiving too many messages. This list isn't exhaustive; the factors that are relevant in your case depend on your specific email sending program.

What you can do to resolve the issue

We recommend that you implement a double opt-in strategy, as described in Building and maintaining your lists, for acquiring new addresses, and that you only send email to addresses that complete the double opt-in process.

Additionally, you should purge your lists of addresses that haven't interacted with your emails recently. You can use open and click tracking, as described in Monitoring your Amazon SES sending activity, to determine which users are viewing and interacting with the content you send.

If your account is under review

At the end of the review period, if Amazon SES continues to receive a significant number of direct complaints about messages sent from your account, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. If we agree that the changes you've made appropriately address the issue, we cancel the review period on your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Domain Blocklist Notification

This section contains additional information about domain blocklist notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

Emails sent from your Amazon SES account contain references to domains that have been listed on a reputable Domain Blocklist. Domains on these lists are typically associated with abusive or malicious behavior. The domains in question might or might not be the domains from which you are sending email. Messages that include references or links to a domain on a blocklist, or that include images hosted on such a domain, might also be flagged.

We're unable to provide the names of the domains that are causing your messages to be flagged, or to identify which emails were flagged in this way.

What you can do to resolve the issue

First, create a list of all of the domains referenced in the emails you send through Amazon SES. Next, use the Spamhaus Domain Lookup tool to determine which domains in your email are on the domain blocklist. More than one domain referenced in the emails you send might be on this blocklist.

The Spamhaus Domain Blocklist isn't affiliated with Amazon SES or AWS. We make no guarantees about the accuracy of the domains on this list. The Spamhaus Domain Blocklist and Domain Lookup Tool are owned, operated, and maintained by the Spamhaus Project.

If your account is under review

We look for references to domains that have been used for malicious purposes in the emails that you send during the review period. If your emails still contain a significant number of references to these domains, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we extend the review period to ensure that we're only analyzing the number of blocklisted domains present in your email after you put your changes in place. At the end of this extended review period, if the number of domain blocklist notifications has been reduced or eliminated, and we believe that you've taken steps to prevent this issue from occurring again in the future, we cancel the review period for your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Internal Review Notification

This section contains additional information about internal review notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A comprehensive review of your account identified several characteristics that may cause mailbox providers or recipients to identify your messages as spam.

To protect our abuse detection process, we can't reveal the specific factors that led to your account being flagged in this way.

Common factors that can lead to this determination include the following:

- Messages being flagged by commercial anti-spam systems.
- Message content that implies the recipient hasn't explicitly requested the email.
- Mismatches between the message sender and the branding within the email body.
- Content that doesn't make it obvious who the sender is.
- Sending messages that deal with content that is associated with unsolicited email.
- Formatting patterns associated with unsolicited email.
- Sending from or making reference to domains with poor reputations.

This isn't a comprehensive list. The specific reason for this notification might be a combination of any of these factors, or the reason might be something not listed.

What you can do to resolve the issue

The following suggestions might help reduce the severity of the issue:

- Ensure that the only recipients you are contacting are those who have explicitly asked to receive email from you.
- Never purchase, rent, or borrow lists of email recipients.
- Don't attempt to hide your identity or the purpose of your communication in the messages you send.
- Create a list of all of the domains referenced in the emails you send through Amazon SES, and then use the Spamhaus Domain Lookup tool at https://www.spamhaus.org/lookup/ to determine if any of those domains are on the Spamhaus Domain Blocklist.
- Ensure that you are following industry best practices when designing your emails.

This list isn't exhaustive, but it should help you identify some of the most common factors that might lead to your email being flagged.

The Spamhaus Domain Blocklist isn't affiliated with Amazon SES or AWS. We make no guarantees about the accuracy of the domains on this list. The Spamhaus Domain Blocklist and Domain Lookup Tool are owned, operated, and maintained by the Spamhaus Project.

If your account is under review, or if your account's ability to send email is paused

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. If we agree that the changes you've made appropriately address the issue, we cancel the review period or remove the sending pause from your account.

If we remove a review period or sending pause from your account, and we observe the same issue at a later time, we might place your account under review or pause your ability to send email again. In extreme cases, or if we observe repeated instances of the same issue, we might permanently suspend your account's ability to send email.

See Amazon SES Sending review process FAQs for more information about what to do if your account is under review, or your account's ability to send email is paused.

Mailbox Provider Notification

This section contains additional information about mailbox provider notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A major mailbox provider has reported to us that unsolicited or malicious email is being sent from an address or domain associated with your Amazon SES account.

We can't share the identity of the organization that issued this report. Additionally, we don't have information about the specific factors that caused the mailbox provider to issue the report. Typically, mailbox providers make this kind of determination based on customer feedback, customer engagement metrics, attempted deliveries to invalid addresses, and content that is flagged by spam filters. This list isn't exhaustive; there might be other factors that caused the mailbox provider to flag your content.

What you can do to resolve the issue

To resolve this issue, you need to determine which aspects of your email sending program might have caused mailbox providers to flag your mail as being problematic. You must then change your sending program to address those issues.

If your account is under review

At the end of the review period, if the mailbox provider continues to identify the email sent from your account as being problematic, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we will extend the review period to ensure that we're only analyzing the number of mailbox provider notifications we receive after you implement your changes. At the end of this extended review period, if the mailbox provider no longer reports your account as being problematic, we might remove the review from your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Recipient Feedback Notification

This section contains additional information about recipient feedback notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A major mailbox provider has reported to us that large numbers of their users are reporting mail sent from your Amazon SES account as unsolicited. This type of feedback isn't visible in the complaints reported by mailbox providers directly, and isn't included in the Amazon SES bounce and complaint notifications.

A large number of complaints can have a negative impact on all Amazon SES users. To protect your reputation and that of other Amazon SES customers, we take immediate action when an account receives a certain number of complaints.

We are unable to provide a list of the specific email addresses that are reporting your email as unsolicited. Additionally, we're unable to share the name of the mailbox provider that has reported this issue to us.

What you can do to resolve the issue

To resolve this issue, you need to determine which aspects of your email sending program might be causing your recipients to issue complaints against the email messages they receive from you. After you identify these factors, change your email sending practices to correct them.

To acquire new addresses, we recommend that you implement a double opt-in strategy, as described in Building and maintaining your lists. We recommend that you only send email to addresses that have completed the double opt-in process.

Additionally, you should purge your lists of addresses that haven't interacted with your emails recently. You can use open and click tracking, as described in Monitoring your Amazon SES sending activity, to determine which users are viewing and interacting with the content you send.

If your account is under review

At the end of the review period, if the mailbox provider continues to report a significant number of complaints, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we extend the review period to ensure that we're only analyzing the number of mailbox provider complaints that we receive after you implement your changes. At the end of this extended review period, if the number of mailbox provider complaints has been reduced or eliminated, we might remove the review from your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that

this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Related Account Notification

This section contains additional information about related account notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

We have detected serious problems related to emails sent from another Amazon SES account. We believe that the problematic account is related to your AWS account, so we have taken action to avoid similar problems.

What you can do to resolve the issue

When we pause an account's ability to send email, we always send information about the reasons for the sending pause to the owner of that account. Refer to the email we sent to the owner of the related account for more information.

You should address the issues with the related account first. After you implement changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. If we agree that the changes you've made appropriately address the issue, we cancel the review period or remove the sending pause from your account.

Spamtrap Notification

This section contains additional information about spamtrap notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A third-party anti-spam organization has reported to us that their spamtrap addresses recently received email from a verified address or domains associated with your Amazon SES account.

A spamtrap is a dormant email address that is used exclusively to lure unsolicited email (spam). A large number of spamtrap reports can have a negative impact on all Amazon SES users. To protect your reputation and that of other Amazon SES customers, we take immediate action when an account sends a particular volume of email to spamtrap addresses.

What you can do to resolve the issue

We can't reveal the email addresses associated with the spamtrap you encountered. These addresses are closely guarded by the organizations that own them, and once the addresses are known, they become worthless.

Sending email to spamtrap addresses typically indicates that there is an issue with how you acquire your customers' email addresses. For example, purchased lists of email addresses can contain spamtrap addresses, which is why sending to purchased or rented lists is prohibited by the Amazon SES terms of service. To acquire new addresses, we recommend that you implement a double opt-in strategy, as described in Building and maintaining your lists. We recommend that you only send email to addresses that have completed the double opt-in process.

Additionally, you should purge your lists of addresses that haven't interacted with your emails recently. You can use open and click tracking, as described in Monitoring your Amazon SES sending activity, to determine which users are viewing and interacting with the content you send.

If your account is under review

At the end of the review period, if messages are still being sent to spamtrap addresses from your account, we might pause your account's ability to send email until you resolve the issue.

If you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide details of the changes you made. When we receive this information, we extend the review period to ensure that we're only analyzing the number of spamtrap reports we receive after you implement your changes. At the end of this extended review period, if the number of spamtrap reports has been reduced or eliminated, we might remove the review from your account.

If your account's ability to send email is paused

You can request that we reconsider this decision. For more information, see Amazon SES Sending review process FAQs.

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

Vulnerable Site Notification

This section contains additional information about vulnerable site notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A comprehensive review has found that messages are being sent from your account that we don't believe you intended to send. These messages are highly likely to be flagged as spam by mailbox providers and recipients.

Most often in these situations, a third party is abusing a feature of your website to send unwanted email. For example, if your website contains an "email to a friend," "contact us," "invite a friend," or similar feature, a third party can use that feature to send unsolicited email.

What you can do to resolve the issue

First, identify features of your website or applications that might allow third parties to send emails using Amazon SES without your knowledge. In your Support Center case, you can request a sample of the messages we believe were sent in this manner.

Next, modify your application or website to prevent unsolicited sending. For example, add a CAPTCHA, limit the rate at which emails can be sent, remove the ability of users to submit custom content, require users to log in to send email, and remove the ability for the application to generate multiple simultaneous notifications.

If your account is under review, or if your account's ability to send email is paused

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

If we remove a review period or sending pause from your account, and we observe the same issue later, we might place your account under review or pause your ability to send email again. If we observe extreme issues or repeated instances of the same issue, we might permanently suspend your account's ability to send email.

See Amazon SES Sending review process FAQs for more information about what to do if your account is under review, or your account's ability to send email is paused.

Compromised Credentials Notification

This section contains additional information about compromised credentials site notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

A comprehensive review has found that messages are being sent from your account that we don't believe you intended to send. These messages are highly likely to be flagged as spam by mailbox providers and recipients.

Some common causes are compromised IAM access keys, compromised SMTP passwords, or other security vulnerabilities.

What you can do to resolve the issue

You should perform a comprehensive security review of your SES utilization mechanisms. Ensure that you have rotated any applicable or SMTP passwords and that you have removed any unauthorized users or resources from your account. Ensure that you are not storing sensitive information such as passwords or access keys on third party web sites or repositories. It is now recommended that you don't use IAM access keys for users, and never for the root user. If you are still using them, you should migrate them over to mechanisms that provide temporary credentials such as creating an user in AWS IAM Identity Center.

If your account is under review, or if your account's ability to send email is paused

When you have implemented changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. Include details of the actions you have taken to resolve this issue, as well as details of your plans to ensure that this issue doesn't occur again. After we receive your request, we review the information that you provided and change the status of your account if necessary.

If we remove a review period or sending pause from your account, and we observe the same issue later, we might place your account under review or pause your ability to send email again. If we observe extreme issues or repeated instances of the same issue, we might permanently suspend your account's ability to send email.

See Amazon SES Sending review process FAQs for more information about what to do if your account is under review, or your account's ability to send email is paused.

Other Notification

This section contains additional information about other notifications shown in the Amazon SES reputation metrics page.

Why you received this notification

An automatic or human review has identified issues that aren't listed in the previous sections of this document.

What you can do to resolve the issue

Refer to the Support Center case that we opened on your behalf for details on the specific issue. To access Support Center, sign into the AWS Management Console and then choose Support Center. In your response to the case, describe the changes you implemented. Depending on your specific situation and the nature of the issues we discovered, we might end the review period or restore your account's ability to send email.

Creating reputation monitoring alarms using CloudWatch

Amazon SES automatically publishes a series of reputation-related metrics to Amazon CloudWatch. You can use these metrics to create alarms that notify you when your bounce or complaint rates reach levels that could impact your account's ability to send email.

> **Note**
>
> The CloudWatch portion of the procedures in this section are intended to just present the core steps for setting up a CloudWatch alarm to monitor your SES sender reputation. They don't explore advanced configurations regarding optional settings for CloudWatch alarms. For complete information about configuring CloudWatch alarms, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

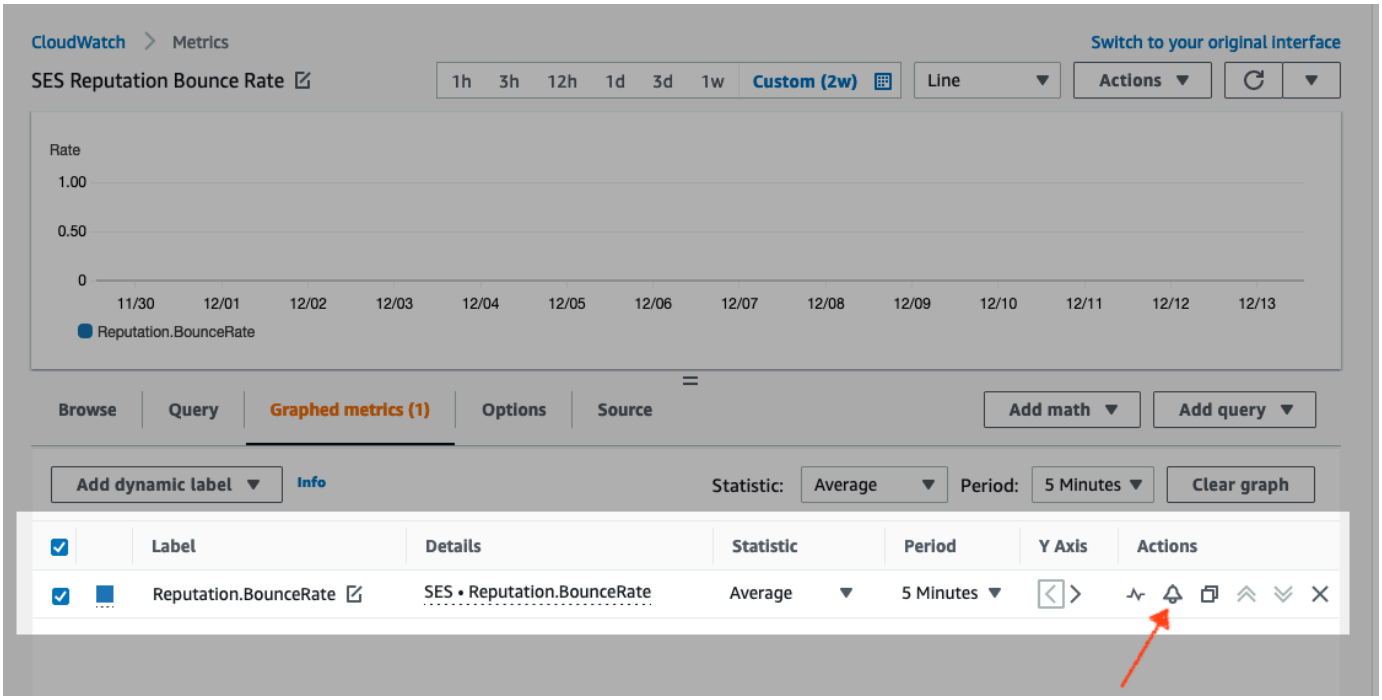
Prerequisites

- Create an Amazon SNS topic, and then subscribe to it using your preferred endpoint (such as email or SMS). For more information, see Creating an Amazon SNS topic and Subscribing to an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

- If you've never sent an email in the current Region, you might not see the **SES** namespace. To ensure that you have metrics, send a test email to the mailbox simulator.

To create a CloudWatch alarm to monitor sending reputation

1. Sign in to the AWS Management Console and open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. In the navigation pane on the left side of the screen, choose **Reputation metrics**.
3. On the **Reputation metrics** page under the **Account-level** tab, in either the **Bounce rate** or **Complaint rate** pane, choose **View in CloudWatch** - this will open the CloudWatch console with your chosen metric.
4. Under the **Graphed metrics** tab, on the line of your chosen metric, for this example, **Reputation.BounceRate**, choose the *alarm bell* icon in the **Actions** column (see image below) - this will open the **Specify metric and conditions** page.



5. Scroll down to the **Conditions** pane, and choose **Static** in the **Threshold type** field.
   a. In the **Whenever *`metric`* is...** field, choose **Greater/Equal**.
   b. In the **than...** field, specify the value that should cause CloudWatch to raise an alarm.
      - If you're creating an alarm to monitor your bounce rate, note that Amazon SES recommends that you maintain a bounce rate under 5%. If the bounce rate for your

account is greater than 10%, we might pause your account's ability to send email. For this reason, you should configure CloudWatch to send you a notification when the bounce rate for your account is greater than or equal to 0.05 (5%).

- If you're creating an alarm to monitor your complaint rate, note that Amazon SES recommends that you maintain a complaint rate under 0.1%. If the complaint rate for your account is greater than 0.5%, we might pause your account's ability to send email. For this reason, you should configure CloudWatch to send you a notification when the complaint rate for your account is greater than or equal to 0.001 (0.1%).

c. Expand **Additional configuration** and choose **Treat missing data as ignore (maintain the alarm state)** in the **Missing data treatment** field.

d. Choose **Next**.

6. On the **Configure actions** pane, choose **In Alarm** in the **Alarm state trigger** field.

   a. Choose **Select an existing SNS topic** in the **Select an SNS topic** field.

   b. Choose the topic that you created and subscribed to in the prerequisites in the **Send a notification to...** search box.

   c. Choose **Next**.

7. On the **Add name and description** pane, enter a name and description for the alarm, and then choose **Next**.

8. On the **Preview and create** pane, confirm your settings, and if satisfied, choose **Create alarm**. If there's something you'd like to change, select the **Previous** button for each section you'd like to go back to and edit.

SNDS metrics for dedicated IPs

You can view Smart Network Data Services (SNDS) data for leased dedicated IP addresses in each AWS Region where you use Amazon SES. This SNDS data is available through the Amazon CloudWatch console.

SNDS is an Outlook program that allows IP owners to help prevent spam within their IP space. Amazon SES provides this important data for those who lease dedicated IPs. The SNDS data provides insight into the IP's mail sending behavior and calls out areas of concern for your sender reputation.

> **Note**
>
> When referring to Outlook, this covers all the domains they track. For example, this can cover Hotmail.com, Outlook.com, and Live.com.

To view SNDS data for your dedicated IP addresses

1. Sign in to the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, expand **Metrics** and choose **All metrics**.

   (Directions are given for the new CloudWatch console interface.)
3. Under the **Browse** tab in the **Metrics** container, select your AWS Region, then choose **SES**.
4. Choose **IP Metrics** which will show you all of your dedicated IPs tracked by SNDS.

   *(Note: if there are no dedicated IP addresses associated with your account in the selected region, **IP Metrics** will not appear in the CloudWatch console.)*
5. View all of your dedicated IPs tracked by SNDS in this list, or select an individual IP address to view only its metrics.

The following metrics are provided for each dedicated IP address and defined by Outlook. For more information, see Outlook's SNDS FAQs.

> **Note**
>
> These metrics represent an activity period that provides updated data once a day. The metrics also have a corresponding timestamp, which reflects a 24-hour period.

- **SNDS.RCPTCommands** - This is the number of RCPT commands perceived by SNDS for the specific IP address during the activity period. RCPT commands are part of the SMTP protocol used to send mail, which specifies the recipient address to which you are trying to deliver email.
- **SNDS.DATACommands** - The number of DATA commands perceived by SNDS for the specific IP address during the activity period. DATA commands are part of the SMTP protocol used to send mail, specifically that part which actually transmits the message to the previously established intended recipient(s).

- **SNDS.MessageRecipients** - The number of recipients on messages perceived by SNDS for the specific IP address during the activity period.
- **SNDS.SpamRate** - Displays the aggregate results of the spam filtering applied to all messages sent by the IP address during the given activity period.
  - A SpamRate of 0 means the IP address has less than 10% spam.
  - A SpamRate of 0.5 means that between 10% and 90% spam is generated from the IP address.
  - A SpamRate of 1 means 90% or more spam is generated from the IP address.
- **SNDS.ComplaintRate** - This is the fraction of the time that a message received from the IP is complained about by an Outlook user during the activity period.
  - A ComplaintRate of 1 means a 100% complaint rate.
  - A ComplaintRate of 0.05 would mean a 5% complaint rate, for example.
  - A ComplaintRate of 0 means the rate is less than 0.1%.
- **SNDS.TrapHits** - Displays the number of messages sent to "trap accounts." Trap accounts are accounts maintained by Outlook that don't solicit any mail. Thus, any messages sent to trap accounts are very likely to be spam.

Troubleshooting questions

Q1. Why does data not populate every day? Either of the following scenarios could apply:

- SNDS data is dependent on Outlook's SNDS program.
- There is a minimum threshold of emails SNDS needs to receive to calculate a value. Data may not be available at times where email volume on an IP was low.

Q2. Why are the SNDS.SpamRate and SNDS.ComplaintRate metrics changing, and what do I do if the rate changes to a value of 1?

This is an indicator that something in your sending behavior has triggered a negative response from the Outlook SNDS program. In this case, you want to check other Internet Service Providers (ISPs) as well as your engagement numbers to make sure it isn't a global problem. If it is a global problem, you may see issues with multiple ISPs, which would suggest a list, content, distribution, or permissions problem. If it is specific to Outlook, review how to best deliver to Outlook.

Q3. What actions will AWS Support take if my SNDS.SpamRate changes from a value of 0 (or 0.5) to 1?

AWS does not have any control over SNDS and therefore has no influence over SNDS. All mitigation requests need to filed directly with Outlook via their New support request form.

Automatically pausing email sending

To protect your sender reputation, you can temporarily pause email sending for messages sent using specific configuration sets, or for all messages sent from your Amazon SES account in a specific AWS Region.

By using Amazon CloudWatch and Lambda, you can create a solution that automatically pauses your email sending when your reputation metrics (such as bounce rate or complaint rate) exceed certain thresholds. This topic contains procedures for setting up this solution.

Topics in this section:

- Automatically pausing email sending for your entire Amazon SES account
- Automatically pausing email sending for a configuration set

Automatically pausing email sending for your entire Amazon SES account

The procedures in this section explain the steps to set up Amazon SES, Amazon SNS, Amazon CloudWatch, and AWS Lambda to automatically pause email sending for your Amazon SES account in a single AWS Region. If you send email from multiple regions, repeat the procedures in this section for each region in which you want to implement this solution.

Topics in this section:

- Part 1: Create an IAM Role
- Part 2: Create the Lambda Function
- Part 3: Re-Enable Email Sending for Your Account
- Part 4: Create an Amazon SNS Topic and Subscription
- Part 5: Create a CloudWatch Alarm
- Part 6: Test the solution

Part 1: Create an IAM Role

The first step in configuring automatic pausing of email sending is to create an IAM role that can execute the `UpdateAccountSendingEnabled` API operation.

To create the IAM role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, choose **AWS service** for the **Trusted entity type**.
5. Under **Use case**, choose **Lambda**, then choose **Next**.
6. On the **Add permissions** page, choose the following policies:

   - **AWSLambdaBasicExecutionRole**
   - **AmazonSESFullAccess**

   > **Tip**
   >
   > Use the search box under **Permission policies** to quickly locate these policies, but note that after searching for and selecting the first policy, you must choose **Clear filters** before searching and selecting the second policy.

   Then choose **Next**.
7. On the **Name, review, and create** page, under **Role details**, enter a meaningful name for the policy in the **Role name** field.
8. Verify that the two policies you selected are listed in the **Permissions policy summary** table, then choose **Create role**.

Part 2: Create the Lambda Function

After you create an IAM role, you can create the Lambda function that pauses email sending for your account.

To create the Lambda function

1. Open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. Use the region selector to choose the region in which you want to deploy this Lambda function.

   > **Note**
   >
   > This function only pauses email sending in the AWS Region you select in this step. If you send email from more than one region, repeat the procedures in this section for each region in which you want to automatically pause email sending.

3. Choose **Create function**.
4. Under **Create function**, choose **Author from scratch**.
5. Under **Basic information**, complete the following steps:

   - For **Function name**, type a name for the Lambda function.
   - For **Runtime**, choose **Node.js 18x** (or the version currently offered in the select list).
   - For **Architecture**, keep the preselected default, **x86_64**.
   - Under Permissions, expand **Change default execution role** and choose **Use an existing role**.
   - Click inside the **Existing role** list box, and choose the IAM role you created in the section called “Part 1: Create an IAM Role”.

   Then choose **Create function**.
6. Under **Code source**, in the code editor, paste the following code:

```
'use strict';


const { SES } = require("@aws-sdk/client-ses")

// Create a new SES object.

var ses = new SES({});

// Specify the parameters for this operation. In this case, there is only one
```

```
// parameter to pass: the Enabled parameter, with a value of false
// (Enabled = false disables email sending, Enabled = true enables it).
var params = {
    Enabled: false
};

exports.handler = (event, context, callback) => {
    // Pause sending for your entire SES account
    ses.updateAccountSendingEnabled(params, function(err, data) {
        if(err) {
            console.log(err.message);
        } else {
            console.log(data);
        }
    });
};
```

Then choose **Deploy**.

7. Choose **Test**. If the **Configure test event** window appears, type a name in the **Event name** field, and then choose **Save**.
8. Expand the **Test** drop box and select the name of the event you just created, and then choose **Test**.
9. The **Execution results** tab will appear - just below it and to the right, ensure that `Status:` `Succeeded` is displayed. If the function failed to execute, do the following:

   - Verify that the IAM role you created in the section called "Part 1: Create an IAM Role" contains the correct policies.
   - Verify that the code in the Lambda function does not contain any errors. The Lambda code editor automatically highlights syntax errors and other potential issues.

Part 3: Re-Enable Email Sending for Your Account

A side effect of testing the Lambda function in the section called "Part 2: Create the Lambda Function" is that email sending for your Amazon SES account is paused. In most cases, you do not want to pause sending for your account until the CloudWatch alarm is triggered.

The procedures in this section re-enable email sending for your Amazon SES account. To complete these procedures, you must install and configure the AWS Command Line Interface. For more information, see the AWS Command Line Interface User Guide.

To re-enable email sending

1. At the command line, type the following command to re-enable email sending for your account. Replace *sending_region* with the name of the Region in which you want to re-enable email sending.

```
aws ses update-account-sending-enabled --enabled --region sending_region
```

2. At the command line, type the following command to check the email sending status for your account:

```
aws ses get-account-sending-enabled --region sending_region
```

   If you see the following output, then you have successfully re-enabled email sending for your account:

```
{
    "Enabled": true
}
```

Part 4: Create an Amazon SNS Topic and Subscription

For CloudWatch to execute your Lambda function when an alarm is triggered, you must first create an Amazon SNS topic and subscribe the Lambda function to it.

To create the Amazon SNS topic and subscribe the Lambda function to it

1. Open the Amazon SNS console at https://eusc-de-east-1.console.amazonaws-eusc.eu/sns/v3/home.
2. Create a topic by following the steps in the *Amazon Simple Notification Service Developer Guide*.
   - The **Type** must be **Standard** (not **FIFO**).
3. Subscribe to the topic by following the steps in the *Amazon Simple Notification Service Developer Guide*.
   a. For **Protocol** choose **AWS Lambda**.

b. For **Endpoint**, choose the Lambda function you created in the section called "Part 2: Create the Lambda Function".

Part 5: Create a CloudWatch Alarm

This section contains procedures for creating an alarm in CloudWatch that is triggered when a metric reaches a certain threshold. When the alarm is triggered, it delivers a notification to the Amazon SNS topic you created in the section called "Part 4: Create an Amazon SNS Topic and Subscription", which then executes the Lambda function you created in the section called "Part 2: Create the Lambda Function".

To create a CloudWatch alarm

1. Open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. Use the region selector to choose the region in which you want to automatically pause email sending.
3. In the navigation pane, choose **Alarms**.
4. Choose **Create Alarm**.
5. On the **Create Alarm** window, under **SES Metrics**, choose **Account Metrics**.
6. Under **Metric Name**, choose one of the following options:

   - **Reputation.BounceRate** – Choose this metric if you want to pause email sending for your account when the overall hard bounce rate for your account crosses a threshold that you define.
   - **Reputation.ComplaintRate** – Choose this metric if you want to pause email sending for your account when the overall complaint rate for your account crosses a threshold that you define.

   Choose **Next**.
7. Complete the following steps:

   - Under **Alarm Threshold**, for **Name**, type a name for the alarm.
   - Under **Whenever: Reputation.BounceRate** or **Whenever: Reputation.ComplaintRate**, specify the threshold that causes the alarm to trigger.

> **Note**
>
> Your account is automatically placed under review if your bounce rate exceeds 5%, or if your complaint rate exceeds 0.1%. When you specify the bounce or complaint rate that causes the CloudWatch alarm to trigger, we recommend that you use values that are below these rates to prevent your account from being placed under review.

- Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose the Amazon SNS topic you created in the section called "Part 4: Create an Amazon SNS Topic and Subscription".

Choose **Create Alarm**.

Part 6: Test the solution

You can now test the alarm to ensure that it executes the Lambda function when it enters the ALARM state. You can use the `SetAlarmState` API operation to temporarily change the state of the alarm.

The procedures in this section are optional, but we recommend that you complete them to ensure that the entire solution is configured correctly.

1. At the command line, type the following command to check the email sending status for your account. Replace *region* with the name of the Region.

   ```
   aws ses get-account-sending-enabled --region region
   ```

   If sending is enabled for your account, you see the following output:

   ```
   {
       "Enabled": true
   }
   ```

2. At the command line, type the following command to temporarily change the alarm state to ALARM: **aws cloudwatch set-alarm-state --alarm-name *MyAlarm* --state-value ALARM --state-reason "Testing execution of Lambda function" --region *region***

Replace *MyAlarm* in the preceding command with the name of the alarm you created in the section called "Part 5: Create a CloudWatch Alarm", and replace *region* with the Region in which you want to automatically pause email sending.

> **Note**
>
> When you execute this command, the status of the alarm switches from OK to ALARM and back to OK within a few seconds. You can view these status changes on the alarm's **History** tab in the CloudWatch console, or by using the DescribeAlarmHistory operation.

3. At the command line, type the following command to check the email sending status for your account.

```
aws ses get-account-sending-enabled --region region
```

If the Lambda function executed successfully, you see the following output:

```
{
    "Enabled": false
}
```

4. Complete the steps in the section called "Part 3: Re-Enable Email Sending for Your Account" to re-enable email sending for your account.

Automatically pausing email sending for a configuration set

You can configure Amazon SES to export reputation metrics that are specific to emails that are sent using a specific configuration set to Amazon CloudWatch. You can then use these metrics to create CloudWatch alarms that are specific to these configuration sets. When these alarms exceed certain thresholds, you can automatically pause the sending of emails that use the specified configuration sets, without impacting the overall email sending capabilities of your Amazon SES account.

> **Note**
>
> The solution described in this section pauses email sending for a specific configuration set in a single AWS Region. If you send email from multiple regions, repeat the procedures in this section for each region in which you want to implement this solution.

Topics in this section:

- Part 1: Enable Reputation Metric Reporting for the Configuration Set
- Part 2: Create an IAM Role
- Part 3: Create the Lambda Function
- Part 4: Re-Enable Email Sending for the Configuration Set
- Part 5: Create an Amazon SNS Topic
- Part 6: Create a CloudWatch Alarm
- Part 7: Test the solution

Part 1: Enable Reputation Metric Reporting for the Configuration Set

Before you can configure Amazon SES to automatically pause email sending for a configuration set, you must first enable the export of reputation metrics for the configuration set.

To enable the export of bounce and complaint metrics for the configuration set, complete the steps in the section called “View and export reputation metrics”.

Part 2: Create an IAM Role

The first step in configuring automatic pausing of email sending is to create an IAM role that can execute the `UpdateConfigurationSetSendingEnabled` API operation.

To create the IAM role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Under **Select type of trusted entity**, choose **AWS service**.

5. Under **Choose the service that will use this role**, choose **Lambda**. Choose **Next: Permissions**.
6. On the **Attach permissions policies** page, choose the following policies:

   - **AWS LambdaBasicExecutionRole**
   - **AmazonSESFullAccess** (We recommend you use a custom role tailored to your needs that includes permissions to call `UpdateConfigurationSetSendingEnabled`.)

   > **Tip**
   >
   > Use the search box at the top of the list of policies to quickly locate these policies.

   Choose **Next: Review**.
7. On the **Review** page, for **Name**, type a name for the role. Choose **Create role**.

Part 3: Create the Lambda Function

After you create an IAM role, you can create the Lambda function that pauses email sending for the configuration set.

To create the Lambda function

1. Open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. Use the region selector to choose the region in which you want to deploy this Lambda function.

   > **Note**
   >
   > This function only pauses email sending for configuration sets in the AWS Region you select in this step. If you send email from more than one region, repeat the procedures in this section for each region in which you want to automatically pause email sending.
3. Choose **Create function**.
4. Under **Create function**, choose **Author from scratch**.
5. Under **Author from scratch**, complete the following steps:

- For **Name**, type a name for the Lambda function.
- For **Runtime**, choose **Node.js 14x** (or the version currently offered in the select list).
- For **Role**, choose **Choose an existing role**.
- For **Existing role**, choose the IAM role you created in the section called “Part 2: Create an IAM Role”.

Choose **Create function**.

6. Under **Function code**, in the code editor, paste the following code:

```
'use strict';

import {
    SES
}
from 'aws-sdk';

const ses = new SES();
const configSet = 'CONFIG_SET_NAME_HERE';

const params = {
    ConfigurationSetName: configSet,
    Enabled: false
};

export const handler = async (event) => {
    try {
        const data = await
 ses.updateConfigurationSetSendingEnabled(params).promise();

        console.log('Configuration Set Update:', data);

        return {
            statusCode: 200,
            body: JSON.stringify({
                message: 'Successfully paused email sending for configuration
 set.',
                data
            }),
        };
    }
```

```
    catch (err) {
        console.error('Error:', err.message);
        return {
            statusCode: 500,
            body: JSON.stringify({
                message: 'Failed to pause email sending for configuration set.',
                error: err.message
            }),
        };
    }
};
```

Replace *ConfigSet* in the preceding code with the name of the configuration set. Choose **Save**.

7. Choose **Test**. If the **Configure test event** window appears, type a name in the **Event name** field, and then choose **Create**.
8. Ensure that the notification bar at the top of the page says `Execution result: succeeded`. If the function failed to execute, do the following:
   - Verify that the IAM role you created in the section called "Part 2: Create an IAM Role" contains the correct policies.
   - Verify that the code in the Lambda function does not contain any errors. The Lambda code editor automatically highlights syntax errors and other potential issues.

Part 4: Re-Enable Email Sending for the Configuration Set

A side effect of testing the Lambda function in the section called "Part 3: Create the Lambda Function" is that email sending for the configuration set is paused. In most cases, you do not want to pause sending for the configuration set until the CloudWatch alarm is triggered.

The procedures in this section re-enable email sending for your configuration set. To complete these procedures, you must install and configure the AWS Command Line Interface. For more information, see the AWS Command Line Interface User Guide.

To re-enable email sending

1. At the command line, type the following command to re-enable email sending for the configuration set:

```
aws ses update-configuration-set-sending-enabled \
--configuration-set-name ConfigSet \
--enabled
```

In the preceding command, replace *ConfigSet* with the name of the configuration set for which you want to pause email sending.

2. At the command line, type the following command to ensure that email sending is enabled:

```
aws ses describe-configuration-set \
--configuration-set-name ConfigSet \
--configuration-set-attribute-names reputationOptions
```

The command produces output that resembles the following example:

```
{
    "ConfigurationSet": {
        "Name": "ConfigSet"
    },
    "ReputationOptions": {
        "ReputationMetricsEnabled": true,
        "SendingEnabled": true
    }
}
```

If the value of `SendingEnabled` is `true`, then email sending for the configuration set was successfully re-enabled.

Part 5: Create an Amazon SNS Topic

For CloudWatch to execute the Lambda function when an alarm is triggered, you must first create an Amazon SNS topic and subscribe the Lambda function to it.

To create the Amazon SNS topic

1. Open the Amazon SNS console at https://eusc-de-east-1.console.amazonaws-eusc.eu/sns/v3/home.
2. Use the region selector to choose the region in which you want to automatically pause email sending.

3. In the navigation pane, choose **Topics**.
4. Choose **Create new topic**.
5. On the **Create new topic** window, for **Topic name**, type a name for the topic. Optionally, you can type a more descriptive name in the **Display name** field.

   Choose **Create topic**.
6. In the list of topics, check the box next to the topic you created in the previous step. On the **Actions** menu, choose **Subscribe to topic**.
7. On the **Create subscription** window, make the following selections:

   - For **Protocol**, choose **AWS Lambda**.
   - For **Endpoint**, choose the Lambda function you created in the section called "Part 3: Create the Lambda Function".
   - For **Version or alias**, choose **default**.
8. Choose **Create subscription**.

Part 6: Create a CloudWatch Alarm

This section contains procedures for creating an alarm in CloudWatch that is triggered when a metric reaches a certain threshold. When the alarm is triggered, it delivers a notification to the Amazon SNS topic you created in the section called "Part 5: Create an Amazon SNS Topic", which then executes the Lambda function you created in the section called "Part 3: Create the Lambda Function".

To create a CloudWatch alarm

1. Open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. Use the region selector to choose the region in which you want to automatically pause email sending.
3. In the navigation pane on the left, choose **Alarms**.
4. Choose **Create Alarm**.
5. On the **Create Alarm** window, under **SES Metrics**, choose **Configuration Set Metrics**.
6. In the **ses:configuration-set** column, locate the configuration set for which you want to create an alarm. Under **Metric Name**, choose one of the following options:

- **Reputation.BounceRate** – Choose this metric if you want to pause email sending for the configuration set when the overall hard bounce rate for the configuration set crosses a threshold that you define.
- **Reputation.ComplaintRate** – Choose this metric if you want to pause email sending for the configuration set when the overall complaint rate for the configuration set crosses a threshold that you define.

Choose **Next**.

7. Complete the following steps:

- Under **Alarm Threshold**, for **Name**, type a name for the alarm.
- Under **Whenever: Reputation.BounceRate** or **Whenever: Reputation.ComplaintRate**, specify the threshold that causes the alarm to trigger.

> **Note**
>
> If the overall bounce rate for your Amazon SES account exceeds 10%, or if the overall complaint rate for your Amazon SES account exceeds .5%, your Amazon SES account is automatically placed under review. When you specify the bounce or complaint rate that causes the CloudWatch alarm to trigger, we recommend that you use values that are far below these rates to prevent your account from being placed under review.

- Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose the Amazon SNS topic you created in the section called “Part 5: Create an Amazon SNS Topic”.

Choose **Create Alarm**.

Part 7: Test the solution

You can now test the alarm to ensure that it executes the Lambda function when it enters the `ALARM` state. You can use the `SetAlarmState` operation in the CloudWatch API to temporarily change the state of the alarm.

The procedures in this section are optional, but we recommend that you complete them to verify that the entire solution is configured correctly.

To test the solution

1. At the command line, type the following command to check the email sending status for the configuration set:

   ```
   aws ses describe-configuration-set --configuration-set-name ConfigSet
   ```

   If sending is enabled for the configuration set, you see the following output:

   ```
   {
       "ConfigurationSet": {
           "Name": "ConfigSet"
       },
       "ReputationOptions": {
           "ReputationMetricsEnabled": true,
           "SendingEnabled": true
       }
   }
   ```

   If the value of `SendingEnabled` is `true`, then email sending is currently enabled for the configuration set.

2. At the command line, type the following command to temporarily change the alarm state to `ALARM`:

   ```
   aws cloudwatch set-alarm-state \
   --alarm-name MyAlarm \
   --state-value ALARM \
   --state-reason "Testing execution of Lambda function"
   ```

   Replace *MyAlarm* in the preceding command with the name of the alarm you created in the section called “Part 6: Create a CloudWatch Alarm”.

   > **Note**
   >
   > When you execute this command, the status of the alarm switches from `OK` to `ALARM` and back to `OK` within a few seconds. You can view these status changes on the

alarm's **History** tab in the CloudWatch console, or by using the DescribeAlarmHistory operation.

3. At the command line, type the following command to check the email sending status for the configuration set:

```
aws ses describe-configuration-set \
--configuration-set-name ConfigSet
```

If the Lambda function executed successfully, you see output that resembles the following example:

```
{
    "ConfigurationSet": {
        "Name": "ConfigSet"
    },
    "ReputationOptions": {
        "ReputationMetricsEnabled": true,
        "SendingEnabled": false
    }
}
```

If the value of `SendingEnabled` is `false`, then email sending for the configuration set is disabled, indicating that the Lambda function executed successfully.

4. Complete the steps in the section called "Part 4: Re-Enable Email Sending for the Configuration Set" to re-enable email sending for the configuration set.

Monitoring SES events using Amazon EventBridge

EventBridge is a serverless service that uses events to connect application components together, making it easier for you to build scalable event-driven applications. Event-driven architecture is a style of building loosely-coupled software systems that work together by emitting and responding to events. Events are JSON-formatted messages that typically represent a change in a resource or environment, or other management event.

Certain SES features will generate and send events that you define when creating an event destination to the EventBridge default event bus. An event bus is a router that receives events and delivers them to zero or more destinations, or *targets*. Rules you associate with the event bus evaluate events as they arrive. Each rule checks whether an event matches the rule's pattern. If the event does match, EventBridge sends the event to the specified targets.

SES sends events to EventBridge when a feature has a state change or status update. You can use EventBridge rules to route events to your defined targets. These events will be delivered on a best-effort basis, and they might be delivered out of order.

Topics

- SES events
- SES events schema reference
- Using EventBridge with SES events
- Additional EventBridge resources

SES events

The following events are generated by SES features and sent to the default event bus in EventBridge. For more information, including detail data for each event type, see ???.

Virtual Deliverability Manager advisor events

Event type	Description
Advisor Recommendation Status Open	An event generated whenever a new recommendation is opened in the Virtual Deliverability Manager advisor.

Event type	Description
Advisor Recommendation Status Resolved	An event generated whenever a recommendation is resolved in the Virtual Deliverability Manager advisor.

SES email sending events

Event type	Description
Email Bounced	A *hard bounce* that the recipient's mail server permanently rejected the email. (*Soft bounces* are only included when SES fails to deliver the email after retrying for a period of time.)
Email Clicked	The recipient clicked one or more links in the email.
Email Complaint Received	The email was successfully delivered to the recipient’s mail server, but the recipient marked it as spam.
Email Delivered	SES successfully delivered the email to the recipient's mail server.
Email Delivery Delayed	The email couldn't be delivered to the recipient’s mail server because a temporary issue occurred. Delivery delays can occur, for example, when the recipient's inbox is full, or when the receiving email server experiences a transient issue.
Email Opened	The recipient received the message and opened it in their email client.
Email Rejected	SES accepted the email, but determined that it contained a virus and didn’t attempt to deliver it to the recipient’s mail server.
Email Rendering Failed	The email wasn't sent because of a template rendering issue. This event type can occur when template data is missing, or when there is a mismatch between template parameters and data. (This event type only occurs when you send email using

Event type	Description
	the SendTemplatedEmail or SendBulkTemplatedEmail API operations.)
Email Sent	The send request was successful and SES will attempt to deliver the message to the recipient's mail server. (If account-level or global suppression is being used, SES will still count it as a send, but delivery is suppressed.)
Email Subscribed	The email was successfully delivered, but the recipient updated the subscription preferences by clicking `List-Unsubscribe` in the email header or the `Unsubscribe` link in the footer.

SES events schema reference

All events from AWS services have a common set of fields containing metadata about the event, such as the AWS service that is the source of the event, the time the event was generated, the account and region in which the event took place, and others. For definitions of these general fields, see Event structure reference in the *EventBridge User Guide*.

In addition, each event has a `detail` field that contains data specific to that particular event. The reference below defines the detail fields for the various SES events.

When using EventBridge to select and manage SES events, it's useful to keep the following in mind:

- The `source` field for all events from SES is set to `aws.ses`.
- The `detail-type` field specifies the event type. See the event type table in the section called "SES events".
- The `detail` field contains the data that is specific to that particular event.

  For some event types, such as those for Virtual Deliverability Manager, the detail field is a rather simplistic data string that is populated from a finite set of static values. Conversely, the detail field for email sending events is more complex as it can consist of many detail sub-fields which are a combination of both static and dynamic values such as the timestamp of when an email was sent, the recipient address, and many other email attributes.

Topics

- Virtual Deliverability Manager advisor status schema
- SES email sending status schema

Virtual Deliverability Manager advisor status schema

The following schema reference defines the fields specific to Virtual Deliverability Manager advisor status events.

Definitions for the general fields that appear in all event schemas (such as `version`, `id`, `account`, and others) can be found in Event structure reference in the *EventBridge User Guide*. The `source` and `detail-type` fields are included in the reference below because they contain SES-specific values for SES events.

`source`

Identifies the service that generated the event. For SES events, this value is `aws.ses`.

`detail-type`

Identifies the type of event.

The values for this field are listed in the *Virtual Deliverability Manager advisor events* table in the section called “SES events”.

`detail`

A JSON object that contains information about the event. The service generating the event determines the content of this field.

The values for this field can be:

- `DKIM verification is not enabled.`
- `DKIM verification has failed.`
- `DKIM signing key length is below 2048 bits.`
- `DMARC configuration was not found.`
- `DMARC configuration could not be parsed.`
- `DKIM record was not found.`

- `DKIM record is not aligned.`
- `MAIL FROM record is not aligned.`
- `SPF record was not found.`
- `SPF record for Amazon SES was not found.`
- `SPF all qualifier is missing.`
- `An SPF configuration issue was found.`
- `BIMI record not found or configured without default selector.`
- `BIMI has malformed TXT record.`

Example: Virtual Deliverability Manager advisor status event

The following is an example Virtual Deliverability Manager advisor status event for the event type `Advisor Recommendation Status Open`. The detail event value in this example is `SPF record was not found.`.

```
{
 "version": "0",
  "id": "abcd9999-ef33-0123-90ab-abcdef666666",
  "detail-type": "Advisor Recommendation Status Open",
  "source": "aws.ses",
  "account": "012345678901",
  "time": "2023-11-15T17:00:59Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ses:us-east-1:012345678901:identity/vdm.events-publishing.cajun.syster-
games.example.com"
  ],
  "detail": { "version": "1.0.0", "data": "SPF record was not found." }
}
```

SES email sending status schema

The following schema reference defines the fields specific to SES email sending status events.

Definitions for the general fields that appear in all event schemas (such as `version`, `id`, `account`, and others) can be found in Event structure reference in the *EventBridge User Guide*. The `source` and `detail-type` fields are included in the reference below because they contain SES-specific values for SES events.

`source`

Identifies the service that generated the event. For SES events, this value is `aws.ses`.

`detail-type`

Identifies the type of event.

The values for this field are listed in the *SES email sending events* table in the section called "SES events".

`detail`

A JSON object that contains information about the event. The service generating the event determines the content of this field.

All of the possible values for this field cannot be listed here because they are comprised of static and dynamic values that are generated by each unique email that is sent at any given moment. However, an example is provided to give you an idea of the type data that this field can contain. *Example detail data for all of the email sending event types can be found using the EventBridge Sandbox, see Specify a sample event in EventBridge.*

An example of detail data generated for the SES email sending event `Email Rendering Failed`:

```
...,
  "detail": {
    "eventType": "Rendering Failure",
    "mail": {
      "timestamp": "2018-01-22T18:43:06.197Z",
      "source": "sender@example.com",
      "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
      "sendingAccountId": "123456789012",
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "destination": ["recipient@example.com"],
      "headersTruncated": false,
      "tags": {
        "ses:configuration-set": ["ConfigSet"]
      }
    },
    "failure": {
      "errorMessage": "Attribute 'attributeName' is not present in the rendering
 data.",
```

```
      "templateName": "MyTemplate"
    }
  }
```

Example: Email sending status event

The following is an example of the full email sending status event for the event type `Email Rendering Failed`. The detail event value in this example is a combination of static and dynamic values based on the email sending event for a specific email.

```
{
  "version": "0",
  "id": "12a18625-3328-fafd-2809-a5e16004f112",
  "detail-type": "Email Rendering Failed",
  "source": "aws.ses",
  "account": "123456789012",
  "time": "2023-07-17T16:48:05Z",
  "region": "us-east-1",
  "resources": ["arn:aws:ses:us-east-1:123456789012:identity/example.com"],
  "detail": {
    "eventType": "Rendering Failure",
    "mail": {
      "timestamp": "2018-01-22T18:43:06.197Z",
      "source": "sender@example.com",
      "sourceArn": "arn:aws:ses:us-east-1:123456789012:identity/sender@example.com",
      "sendingAccountId": "123456789012",
      "messageId": "EXAMPLE7c191be45-e9aedb9a-02f9-4d12-a87d-dd0099a07f8a-000000",
      "destination": ["recipient@example.com"],
      "headersTruncated": false,
      "tags": {
        "ses:configuration-set": ["ConfigSet"]
      }
    },
    "failure": {
      "errorMessage": "Attribute 'attributeName' is not present in the rendering
 data.",
      "templateName": "MyTemplate"
    }
  }
}
```

Using EventBridge with SES events

By default, SES sends events to the EventBridge default event bus. You can create rules on the default event bus to identify specific events for EventBridge to send to one or more specified targets. Each rule contains an *event pattern* that EventBridge uses to match events as they arrive on the event bus. If an event matches the event pattern for a given rule, EventBridge sends the event to the target specified in the rule.

In EventBridge, defining an event pattern is typically part of the larger process of creating a new rule or editing an existing one. To learn how to create EventBridge rules, see Creating Amazon EventBridge rules that react to events in the *EventBridge User Guide*.

By using the Sandbox feature in EventBridge, you can quickly define an event pattern and use a sample event to confirm the pattern matches the desired events, without having to first create or edit a rule. For detailed instructions on using the Sandbox, see Testing an event pattern using the EventBridge Sandbox in the *EventBridge User Guide*.

Specify a SES sample event in the EventBridge Sandbox

You can select sample events for SES events to use them in testing the event patterns you create.

To specify a SES sample event in the EventBridge Sandbox

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Developer resources,** then select **Sandbox**, and on the **Sandbox** page choose the **Event pattern** tab.
3. For **Event source**, choose **AWS events or EventBridge partner events**.
4. In the **Sample event** section, for **Sample event type**, select **AWS events**.
5. For **Sample events**, scroll down to **SES** and then select the desired SES event.

   EventBridge displays a sample event, along with all of its detail data, for the event type.

   You can then use this event to test the event pattern you create in the **Event pattern** section, or use it as the basis for creating your own sample events for pattern testing covered in the following section.

Creating and testing event patterns for SES events

Once you've selected a sample event, as explained in the previous section, you can create an event pattern and use the sample event to make sure it is matching events as desired.

To create and test an event pattern that matches SES events in the EventBridge Sandbox

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Developer resources,** then select **Sandbox**, and on the **Sandbox** page choose the **Event pattern** tab.
3. For **Event source**, choose **AWS events or EventBridge partner events**, and select the sample event you want to test as explained in the previous section.
4. Scroll down to **Creation method**, and choose **Use pattern form**.
5. In the **Event pattern** section, for **Event source** choose **AWS services**.
6. Under **AWS service**, select **SES**.
7. For **Event type**, select the SES event type you want to match.

   EventBridge displays the minimum event pattern, comprised of `source` and `detail-type` fields, that matches the selected SES event.

   In the two examples, the first event pattern matches against all `Advisor Recommendation Status Resolved` events, and in the second, all `Email Bounced` events:

   ```
   {
     "source": ["aws.ses"],
     "detail-type": ["Advisor Recommendation Status Resolved"]
   }
   ```

   ```
   {
     "source": ["aws.ses"],
     "detail-type": ["Email Bounced"]
   }
   ```

8. To make changes to the event pattern, select **Edit pattern** and make your changes in the JSON editor.

You can also match on values in one or more detail data fields. This includes specifying multiple possible values for a field value.

In the following example, the detail field was added to the generated minimum event pattern with the `data` field value specified as `DKIM record was not found` in order to find all Virtual Deliverability Manager advisor events with the same detail value:

```
{
  "source": ["aws.ses"],
  "detail-type": ["Advisor Recommendation Status Resolved"],
  "detail": {
    "data": ["DKIM record was not found."]
  }
}
```

In this example, detail sub-fields were added to report on events generated by all the emails sent from *noreply@example.com* on *2024-08-05* that bounced. (Prefix matching is being used here as part of Content filtering.):

```
{
  "source": ["aws.ses"],
  "detail-type": ["Email Bounced"],
  "detail": {
    "mail": {
      "timestamp": [{
        "prefix": "2024-08-05"
      }],
      "source": ["noreply@example.com"]
    }
  }
}
```

It's important that you read Event patterns in the *EventBridge User Guide*—it explains that the event pattern value you enter in the JSON editor must be surrounded by square brackets `[...]` because it's considered an array. This and more information on how to construct advanced event patterns is also provided.

9. To test if your event pattern matches against the sample event you specified in the **Sample event** pane above, select **Test pattern**. If it matches, a green banner at the bottom of the JSON editor will display, *"Sample event matched the event pattern"*.

10. To troubleshoot errors after selecting **Test pattern**:

    - If there are JSON related errors, the message will indicate the reason, such as, *"Event pattern is not valid. Reason: "data" must be an object or an array at line: 5, column: 14"*. To remedy this, enclose the value on line 5 with square brackets `[ . . . ]`.
    - If there's a discrepancy between the values in the *Sample event* and your *Event pattern*, the message will be, *"Sample event did not match the event pattern"*. This means that one or more values you want to test are different than the example values generated by the *Sample events* generator. To remedy this, proceed with the remaining steps.
11. To change the sample values in the *Sample event* in order to successfully test your *Event pattern*, in the **Sample event** pane, select **Copy** under the JSON editor.
12. Select the radio button next to **Enter my own** for **Sample event type** above the editor.
13. Paste the sample event into the JSON editor, and for any field you're using in your event pattern, replace that same field's value to match the value you specified in your event pattern.
14. Scroll back down to the Event pattern pane and select **Test pattern** again. If all values were entered correctly and match, a green banner at the bottom of the JSON editor will display, *"Sample event matched the event pattern"*.

Additional EventBridge resources

Refer to the following topics in the *Amazon EventBridge User Guide* for more information on how to use EventBridge to process and manage events.

- For detailed information on how event buses work, see Amazon EventBridge event bus.
- For information on event structure, see Events
- For information on constructing event patterns for EventBridge to use when matching events against rules, see Event patterns
- For information on creating rules to specify which events EventBridge processes, see Rules
- For information on to specify what services or other destinations EventBridge sends matched events to, see Targets

Code examples for Amazon SES using AWS SDKs

The following code examples show how to use Amazon SES with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Code examples for Amazon SES using AWS SDKs
  - Basic examples for Amazon SES using AWS SDKs
    - Actions for Amazon SES using AWS SDKs
      - Use CreateReceiptFilter with an AWS SDK
      - Use CreateReceiptRule with an AWS SDK
      - Use CreateReceiptRuleSet with an AWS SDK
      - Use CreateTemplate with an AWS SDK
      - Use DeleteIdentity with an AWS SDK or CLI
      - Use DeleteReceiptFilter with an AWS SDK
      - Use DeleteReceiptRule with an AWS SDK
      - Use DeleteReceiptRuleSet with an AWS SDK
      - Use DeleteTemplate with an AWS SDK
      - Use DescribeReceiptRuleSet with an AWS SDK
      - Use GetIdentityVerificationAttributes with an AWS SDK or CLI
      - Use GetSendQuota with an AWS SDK or CLI
      - Use GetSendStatistics with a CLI
      - Use GetTemplate with an AWS SDK
      - Use ListIdentities with an AWS SDK or CLI
      - Use ListReceiptFilters with an AWS SDK
      - Use ListTemplates with an AWS SDK
      - Use SendBulkTemplatedEmail with an AWS SDK
      - Use SendEmail with an AWS SDK or CLI

        - Use SendRawEmail with an AWS SDK or CLI
        - Use SendTemplatedEmail with an AWS SDK
        - Use UpdateTemplate with an AWS SDK
        - Use VerifyDomainIdentity with an AWS SDK or CLI
        - Use VerifyEmailIdentity with an AWS SDK or CLI
    - Scenarios for Amazon SES using AWS SDKs
        - Build an Amazon Transcribe streaming app
        - Copy Amazon SES email and domain identities from one AWS Region to another using an AWS SDK
        - Create a web application to track DynamoDB data
        - Create an Amazon Redshift item tracker
        - Create an Aurora Serverless work item tracker
        - Detect PPE in images with Amazon Rekognition using an AWS SDK
        - Detect objects in images with Amazon Rekognition using an AWS SDK
        - Detect people and objects in a video with Amazon Rekognition using an AWS SDK
        - Generate credentials to connect to an Amazon SES SMTP endpoint
        - Setting up email sending services
        - Use Step Functions to invoke Lambda functions
        - Verify an email identity and send messages with Amazon SES using an AWS SDK
- Code examples for Amazon SES API v2 using AWS SDKs
    - Basic examples for Amazon SES API v2 using AWS SDKs
        - Hello Amazon SES API v2
        - Actions for Amazon SES API v2 using AWS SDKs
            - Use CreateContact with an AWS SDK
            - Use CreateContactList with an AWS SDK
            - Use CreateEmailIdentity with an AWS SDK
            - Use CreateEmailTemplate with an AWS SDK
            - Use DeleteContactList with an AWS SDK
            - Use DeleteEmailIdentity with an AWS SDK
            - Use DeleteEmailTemplate with an AWS SDK

        - Use GetEmailIdentity with an AWS SDK
        - Use ListContactLists with an AWS SDK
        - Use ListContacts with an AWS SDK
        - Use SendBulkEmail with an AWS SDK
        - Use SendEmail with an AWS SDK
- Scenarios for Amazon SES API v2 using AWS SDKs
    - Send emails with attachments using Amazon SES API v2
    - A complete Amazon SES API v2 Newsletter scenario using an AWS SDK

Code examples for Amazon SES using AWS SDKs

The following code examples show how to use Amazon SES with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon SES

- Basic examples for Amazon SES using AWS SDKs
    - Actions for Amazon SES using AWS SDKs
        - Use CreateReceiptFilter with an AWS SDK
        - Use CreateReceiptRule with an AWS SDK
        - Use CreateReceiptRuleSet with an AWS SDK
        - Use CreateTemplate with an AWS SDK
        - Use DeleteIdentity with an AWS SDK or CLI
        - Use DeleteReceiptFilter with an AWS SDK
        - Use DeleteReceiptRule with an AWS SDK

- Use DeleteReceiptRuleSet with an AWS SDK
- Use DeleteTemplate with an AWS SDK
- Use DescribeReceiptRuleSet with an AWS SDK
- Use GetIdentityVerificationAttributes with an AWS SDK or CLI
- Use GetSendQuota with an AWS SDK or CLI
- Use GetSendStatistics with a CLI
- Use GetTemplate with an AWS SDK
- Use ListIdentities with an AWS SDK or CLI
- Use ListReceiptFilters with an AWS SDK
- Use ListTemplates with an AWS SDK
- Use SendBulkTemplatedEmail with an AWS SDK
- Use SendEmail with an AWS SDK or CLI
- Use SendRawEmail with an AWS SDK or CLI
- Use SendTemplatedEmail with an AWS SDK
- Use UpdateTemplate with an AWS SDK
- Use VerifyDomainIdentity with an AWS SDK or CLI
- Use VerifyEmailIdentity with an AWS SDK or CLI

- Scenarios for Amazon SES using AWS SDKs
  - Build an Amazon Transcribe streaming app
  - Copy Amazon SES email and domain identities from one AWS Region to another using an AWS SDK
  - Create a web application to track DynamoDB data
  - Create an Amazon Redshift item tracker
  - Create an Aurora Serverless work item tracker
  - Detect PPE in images with Amazon Rekognition using an AWS SDK
  - Detect objects in images with Amazon Rekognition using an AWS SDK
  - Detect people and objects in a video with Amazon Rekognition using an AWS SDK
  - Generate credentials to connect to an Amazon SES SMTP endpoint
  - Setting up email sending services
  - Use Step Functions to invoke Lambda functions

- Verify an email identity and send messages with Amazon SES using an AWS SDK

Basic examples for Amazon SES using AWS SDKs

The following code examples show how to use the basics of Amazon Simple Email Service with AWS SDKs.

Examples

- Actions for Amazon SES using AWS SDKs
  - Use CreateReceiptFilter with an AWS SDK
  - Use CreateReceiptRule with an AWS SDK
  - Use CreateReceiptRuleSet with an AWS SDK
  - Use CreateTemplate with an AWS SDK
  - Use DeleteIdentity with an AWS SDK or CLI
  - Use DeleteReceiptFilter with an AWS SDK
  - Use DeleteReceiptRule with an AWS SDK
  - Use DeleteReceiptRuleSet with an AWS SDK
  - Use DeleteTemplate with an AWS SDK
  - Use DescribeReceiptRuleSet with an AWS SDK
  - Use GetIdentityVerificationAttributes with an AWS SDK or CLI
  - Use GetSendQuota with an AWS SDK or CLI
  - Use GetSendStatistics with a CLI
  - Use GetTemplate with an AWS SDK
  - Use ListIdentities with an AWS SDK or CLI
  - Use ListReceiptFilters with an AWS SDK
  - Use ListTemplates with an AWS SDK
  - Use SendBulkTemplatedEmail with an AWS SDK
  - Use SendEmail with an AWS SDK or CLI
  - Use SendRawEmail with an AWS SDK or CLI
  - Use SendTemplatedEmail with an AWS SDK
  - Use UpdateTemplate with an AWS SDK
  - Use VerifyDomainIdentity with an AWS SDK or CLI

- Use VerifyEmailIdentity with an AWS SDK or CLI

Actions for Amazon SES using AWS SDKs

The following code examples demonstrate how to perform individual Amazon SES actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon SES API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon SES using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Simple Email Service API Reference.

Examples

- Use CreateReceiptFilter with an AWS SDK
- Use CreateReceiptRule with an AWS SDK
- Use CreateReceiptRuleSet with an AWS SDK
- Use CreateTemplate with an AWS SDK
- Use DeleteIdentity with an AWS SDK or CLI
- Use DeleteReceiptFilter with an AWS SDK
- Use DeleteReceiptRule with an AWS SDK
- Use DeleteReceiptRuleSet with an AWS SDK
- Use DeleteTemplate with an AWS SDK
- Use DescribeReceiptRuleSet with an AWS SDK
- Use GetIdentityVerificationAttributes with an AWS SDK or CLI
- Use GetSendQuota with an AWS SDK or CLI
- Use GetSendStatistics with a CLI
- Use GetTemplate with an AWS SDK
- Use ListIdentities with an AWS SDK or CLI
- Use ListReceiptFilters with an AWS SDK
- Use ListTemplates with an AWS SDK

- Use SendBulkTemplatedEmail with an AWS SDK
- Use SendEmail with an AWS SDK or CLI
- Use SendRawEmail with an AWS SDK or CLI
- Use SendTemplatedEmail with an AWS SDK
- Use UpdateTemplate with an AWS SDK
- Use VerifyDomainIdentity with an AWS SDK or CLI
- Use VerifyEmailIdentity with an AWS SDK or CLI

Use `CreateReceiptFilter` with an AWS SDK

The following code examples show how to use `CreateReceiptFilter`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Create an Amazon Simple Email Service (Amazon SES) receipt filter..
/*!
  \param receiptFilterName: The name for the receipt filter.
  \param cidr: IP address or IP address range in Classless Inter-Domain Routing
 (CIDR) notation.
  \param policy: Block or allow enum of type ReceiptFilterPolicy.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::createReceiptFilter(const Aws::String &receiptFilterName,
                                      const Aws::String &cidr,
                                      Aws::SES::Model::ReceiptFilterPolicy
 policy,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);
    Aws::SES::Model::CreateReceiptFilterRequest createReceiptFilterRequest;
```

```
    Aws::SES::Model::ReceiptFilter receiptFilter;
    Aws::SES::Model::ReceiptIpFilter receiptIpFilter;
    receiptIpFilter.SetCidr(cidr);
    receiptIpFilter.SetPolicy(policy);
    receiptFilter.SetName(receiptFilterName);
    receiptFilter.SetIpFilter(receiptIpFilter);
    createReceiptFilterRequest.SetFilter(receiptFilter);
    Aws::SES::Model::CreateReceiptFilterOutcome createReceiptFilterOutcome =
 sesClient.CreateReceiptFilter(
            createReceiptFilterRequest);
    if (createReceiptFilterOutcome.IsSuccess()) {
        std::cout << "Successfully created receipt filter." << std::endl;
    }
    else {
        std::cerr << "Error creating receipt filter: " <<
                  createReceiptFilterOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return createReceiptFilterOutcome.IsSuccess();
}
```

- For API details, see CreateReceiptFilter in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```

```
const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an
 IP address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the
 filtered addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers,
 underscores, or dashes.
        Must be less than 64 characters and start and end with a letter or
 number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- For API details, see CreateReceiptFilter in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def create_receipt_filter(self, filter_name, ip_address_or_range, allow):
        """
        Creates a filter that allows or blocks incoming mail from an IP address
 or
        range.

        :param filter_name: The name to give the filter.
        :param ip_address_or_range: The IP address or range to block or allow.
        :param allow: When True, incoming mail is allowed from the specified IP
                      address or range; otherwise, it is blocked.
        """
        try:
            policy = "Allow" if allow else "Block"
            self.ses_client.create_receipt_filter(
                Filter={
                    "Name": filter_name,
                    "IpFilter": {"Cidr": ip_address_or_range, "Policy": policy},
                }
            )
            logger.info(
```

```
                "Created receipt filter %s to %s IP of %s.",
                filter_name,
                policy,
                ip_address_or_range,
            )
        except ClientError:
            logger.exception("Couldn't create receipt filter %s.", filter_name)
            raise
```

- For API details, see CreateReceiptFilter in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" iv_allow = abap_true means 'Allow', abap_false means 'Block'
DATA(lv_policy) = COND /aws1/sesreceiptfilterpolicy(
  WHEN iv_allow = abap_true THEN 'Allow'
  ELSE 'Block'
).

DATA(lo_ip_filter) = NEW /aws1/cl_sesreceiptipfilter(
  iv_policy = lv_policy
  iv_cidr = iv_ip_address_or_range
).

DATA(lo_filter) = NEW /aws1/cl_sesreceiptfilter(
  iv_name = iv_filter_name
  io_ipfilter = lo_ip_filter
).

TRY.
    lo_ses->createreceiptfilter( io_filter = lo_filter ).
    MESSAGE 'Receipt filter created successfully' TYPE 'I'.
```

```
      CATCH /aws1/cx_sesalreadyexistsex INTO DATA(lo_ex1).
        DATA(lv_error) = |Filter already exists: { lo_ex1->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex1.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
        lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex_generic.
    ENDTRY.
```

- For API details, see CreateReceiptFilter in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateReceiptRule` with an AWS SDK

The following code examples show how to use `CreateReceiptRule`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Create an Amazon Simple Email Service (Amazon SES) receipt rule.
/*!
  \param receiptRuleName: The name for the receipt rule.
  \param s3BucketName: The name of the S3 bucket for incoming mail.
  \param s3ObjectKeyPrefix: The prefix for the objects in the S3 bucket.
  \param ruleSetName: The name of the rule set where the receipt rule is added.
  \param recipients: Aws::Vector of recipients.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::SES::createReceiptRule(const Aws::String &receiptRuleName,
                                    const Aws::String &s3BucketName,
                                    const Aws::String &s3ObjectKeyPrefix,
                                    const Aws::String &ruleSetName,
                                    const Aws::Vector<Aws::String> &recipients,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::CreateReceiptRuleRequest createReceiptRuleRequest;

    Aws::SES::Model::S3Action s3Action;
    s3Action.SetBucketName(s3BucketName);
    s3Action.SetObjectKeyPrefix(s3ObjectKeyPrefix);

    Aws::SES::Model::ReceiptAction receiptAction;
    receiptAction.SetS3Action(s3Action);

    Aws::SES::Model::ReceiptRule receiptRule;
    receiptRule.SetName(receiptRuleName);
    receiptRule.WithRecipients(recipients);

    Aws::Vector<Aws::SES::Model::ReceiptAction> receiptActionList;
    receiptActionList.emplace_back(receiptAction);
    receiptRule.SetActions(receiptActionList);

    createReceiptRuleRequest.SetRuleSetName(ruleSetName);
    createReceiptRuleRequest.SetRule(receiptRule);

    auto outcome = sesClient.CreateReceiptRule(createReceiptRuleRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully created receipt rule." << std::endl;
    }
    else {
        std::cerr << "Error creating receipt rule. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see CreateReceiptRule in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
```

```
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- For API details, see CreateReceiptRule in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon S3 bucket where Amazon SES can put copies of incoming emails and create a rule that copies incoming email to the bucket for a specific list of recipients.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
```

```
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def create_bucket_for_copy(self, bucket_name):
        """
        Creates a bucket that can receive copies of emails from Amazon SES. This
        includes adding a policy to the bucket that grants Amazon SES permission
        to put objects in the bucket.

        :param bucket_name: The name of the bucket to create.
        :return: The newly created bucket.
        """
        allow_ses_put_policy = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Sid": "AllowSESPut",
                    "Effect": "Allow",
                    "Principal": {"Service": "ses.amazonaws.com"},
                    "Action": "s3:PutObject",
                    "Resource": f"arn:aws:s3:::{bucket_name}/*",
                }
            ],
        }
        bucket = None
        try:
            bucket = self.s3_resource.create_bucket(
                Bucket=bucket_name,
                CreateBucketConfiguration={
                    "LocationConstraint":
self.s3_resource.meta.client.meta.region_name
                },
            )
            bucket.wait_until_exists()
            bucket.Policy().put(Policy=json.dumps(allow_ses_put_policy))
            logger.info("Created bucket %s to receive copies of emails.",
bucket_name)
        except ClientError:
```

```
            logger.exception("Couldn't create bucket to receive copies of
emails.")
            if bucket is not None:
                bucket.delete()
            raise
        else:
            return bucket


    def create_s3_copy_rule(
        self, rule_set_name, rule_name, recipients, bucket_name, prefix
    ):
        """
        Creates a rule so that all emails received by the specified recipients
are
        copied to an Amazon S3 bucket.

        :param rule_set_name: The name of a previously created rule set to
contain
                              this rule.
        :param rule_name: The name to give the rule.
        :param recipients: When an email is received by one of these recipients,
it
                           is copied to the Amazon S3 bucket.
        :param bucket_name: The name of the bucket to receive email copies. This
                            bucket must allow Amazon SES to put objects into it.
        :param prefix: An object key prefix to give the emails copied to the
bucket.
        """
        try:
            self.ses_client.create_receipt_rule(
                RuleSetName=rule_set_name,
                Rule={
                    "Name": rule_name,
                    "Enabled": True,
                    "Recipients": recipients,
                    "Actions": [
                        {
                            "S3Action": {
                                "BucketName": bucket_name,
                                "ObjectKeyPrefix": prefix,
                            }
                        }
                    ],
```

```
                },
            )
            logger.info(
                "Created rule %s to copy mail received by %s to bucket %s.",
                rule_name,
                recipients,
                bucket_name,
            )
        except ClientError:
            logger.exception("Couldn't create rule %s.", rule_name)
            raise
```

- For API details, see CreateReceiptRule in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" Create S3 action for copying emails to S3
DATA(lo_s3_action) = NEW /aws1/cl_sess3action(
  iv_bucketname = iv_bucket_name
  iv_objectkeyprefix = iv_prefix
).

" Create receipt action with S3 action
DATA(lo_action) = NEW /aws1/cl_sesreceiptaction(
  io_s3action = lo_s3_action
).

" Create list of actions
DATA lt_actions TYPE /aws1/cl_sesreceiptaction=>tt_receiptactionslist.
APPEND lo_action TO lt_actions.

" Create receipt rule
```

```
    DATA(lo_rule) = NEW /aws1/cl_sesreceiptrule(
      iv_name = iv_rule_name
      iv_enabled = abap_true
      it_recipients = it_recipients
      it_actions = lt_actions
    ).

    TRY.
        lo_ses->createreceiptrule(
          iv_rulesetname = iv_rule_set_name
          io_rule = lo_rule
        ).
        MESSAGE 'Receipt rule created successfully' TYPE 'I'.
      CATCH /aws1/cx_sesinvalids3confex INTO DATA(lo_ex1).
        DATA(lv_error) = |Invalid S3 configuration: { lo_ex1->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex1.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
        lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex_generic.
    ENDTRY.
```

- For API details, see CreateReceiptRule in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateReceiptRuleSet` with an AWS SDK

The following code examples show how to use `CreateReceiptRuleSet`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Create an Amazon Simple Email Service (Amazon SES) receipt rule set.
/*!
  \param ruleSetName: The name of the rule set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::createReceiptRuleSet(const Aws::String &ruleSetName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::CreateReceiptRuleSetRequest createReceiptRuleSetRequest;

    createReceiptRuleSetRequest.SetRuleSetName(ruleSetName);

    Aws::SES::Model::CreateReceiptRuleSetOutcome outcome =
 sesClient.CreateReceiptRuleSet(
            createReceiptRuleSetRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully created receipt rule set." << std::endl;
    }
    else {
        std::cerr << "Error creating receipt rule set. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see CreateReceiptRuleSet in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- For API details, see CreateReceiptRuleSet in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def create_receipt_rule_set(self, rule_set_name):
        """
        Creates an empty rule set. Rule sets contain individual rules and can be
        used to organize rules.

        :param rule_set_name: The name to give the rule set.
        """
        try:
            self.ses_client.create_receipt_rule_set(RuleSetName=rule_set_name)
            logger.info("Created receipt rule set %s.", rule_set_name)
        except ClientError:
            logger.exception("Couldn't create receipt rule set %s.",
 rule_set_name)
            raise
```

- For API details, see CreateReceiptRuleSet in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->createreceiptruleset( iv_rulesetname = iv_rule_set_name ).
    MESSAGE 'Receipt rule set created successfully' TYPE 'I'.
  CATCH /aws1/cx_sesalreadyexistsex INTO DATA(lo_ex1).
    DATA(lv_error) = |Rule set already exists: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see CreateReceiptRuleSet in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateTemplate` with an AWS SDK

The following code examples show how to use `CreateTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create an email template.
    /// </summary>
    /// <param name="name">Name of the template.</param>
    /// <param name="subject">Email subject.</param>
    /// <param name="text">Email body text.</param>
    /// <param name="html">Email HTML body text.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string name, string subject,
string text,
        string html)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.CreateTemplateAsync(
                new CreateTemplateRequest
                {
                    Template = new Template
                    {
                        TemplateName = name,
                        SubjectPart = subject,
                        TextPart = text,
                        HtmlPart = html
                    }
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine("CreateEmailTemplateAsync failed with exception: "
 + ex.Message);
        }

        return success;
    }
```

- For API details, see CreateTemplate in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Create an Amazon Simple Email Service (Amazon SES) template.
/*!
  \param templateName: The name of the template.
  \param htmlPart: The HTML body of the email.
  \param subjectPart: The subject line of the email.
  \param textPart: The plain text version of the email.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::createTemplate(const Aws::String &templateName,
                                 const Aws::String &htmlPart,
                                 const Aws::String &subjectPart,
                                 const Aws::String &textPart,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::CreateTemplateRequest createTemplateRequest;
    Aws::SES::Model::Template aTemplate;

    aTemplate.SetTemplateName(templateName);
```

```
    aTemplate.SetHtmlPart(htmlPart);
    aTemplate.SetSubjectPart(subjectPart);
    aTemplate.SetTextPart(textPart);

    createTemplateRequest.SetTemplate(aTemplate);

    Aws::SES::Model::CreateTemplateOutcome outcome = sesClient.CreateTemplate(
            createTemplateRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully created template." << templateName << "."
                  << std::endl;
    }
    else {
        std::cerr << "Error creating template. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see CreateTemplate in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");
```

```
const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
 system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- For API details, see CreateTemplate in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.

        :param subject: The subject of the email.
        :param text: The text version of the email.
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
 html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def create_template(self, name, subject, text, html):
        """
        Creates an email template.

        :param name: The name of the template.
        :param subject: The subject of the email.
        :param text: The plain text version of the email.
        :param html: The HTML version of the email.
```

```
        """
        try:
            template = {
                "TemplateName": name,
                "SubjectPart": subject,
                "TextPart": text,
                "HtmlPart": html,
            }
            self.ses_client.create_template(Template=template)
            logger.info("Created template %s.", name)
            self.template = template
            self._extract_tags(subject, text, html)
        except ClientError:
            logger.exception("Couldn't create template %s.", name)
            raise
```

- For API details, see CreateTemplate in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
DATA(lo_template) = NEW /aws1/cl_sestemplate(
  iv_templatename = iv_name
  iv_subjectpart = iv_subject
  iv_textpart = iv_text
  iv_htmlpart = iv_html
).

TRY.
    lo_ses->createtemplate( io_template = lo_template ).
    MESSAGE 'Template created successfully' TYPE 'I'.
  CATCH /aws1/cx_sesalreadyexistsex INTO DATA(lo_ex1).
    DATA(lv_error) = |Template already exists: { lo_ex1->get_text( ) }|.
```

```
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex1.
      CATCH /aws1/cx_sesinvalidtemplateex INTO DATA(lo_ex2).
        lv_error = |Invalid template: { lo_ex2->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex2.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
        lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex_generic.
    ENDTRY.
```

- For API details, see CreateTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteIdentity` with an AWS SDK or CLI

The following code examples show how to use `DeleteIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Setting up email sending services
- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an email identity.
    /// </summary>
    /// <param name="identityEmail">The identity email to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteIdentityAsync(string identityEmail)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.DeleteIdentityAsync(
                new DeleteIdentityRequest
                {
                    Identity = identityEmail
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteIdentityAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- For API details, see DeleteIdentity in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete the specified identity (an email address or a domain).
/*!
```

```
  \param identity: The identity to delete.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::deleteIdentity(const Aws::String &identity,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::DeleteIdentityRequest deleteIdentityRequest;

    deleteIdentityRequest.SetIdentity(identity);

    Aws::SES::Model::DeleteIdentityOutcome outcome = sesClient.DeleteIdentity(
            deleteIdentityRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted identity." << std::endl;
    }
    else {
        std::cerr << "Error deleting identity. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteIdentity in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an identity

The following example uses the `delete-identity` command to delete an identity from the list of identities verified with Amazon SES:

```
aws ses delete-identity --identity user@example.com
```

For more information about verified identities, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

- For API details, see DeleteIdentity in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- For API details, see DeleteIdentity in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def delete_identity(self, identity):
        """
        Deletes an identity.

        :param identity: The identity to remove.
        """
        try:
            self.ses_client.delete_identity(Identity=identity)
            logger.info("Deleted identity %s.", identity)
        except ClientError:
            logger.exception("Couldn't delete identity %s.", identity)
            raise
```

- For API details, see DeleteIdentity in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->deleteidentity( iv_identity = iv_identity ).
    MESSAGE 'Identity deleted successfully' TYPE 'I'.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
    DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- For API details, see DeleteIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteReceiptFilter` with an AWS SDK

The following code examples show how to use `DeleteReceiptFilter`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Email Service (Amazon SES) receipt filter.
/*!
  \param receiptFilterName: The name for the receipt filter.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::deleteReceiptFilter(const Aws::String &receiptFilterName,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::DeleteReceiptFilterRequest deleteReceiptFilterRequest;

    deleteReceiptFilterRequest.SetFilterName(receiptFilterName);

    Aws::SES::Model::DeleteReceiptFilterOutcome outcome =
 sesClient.DeleteReceiptFilter(
            deleteReceiptFilterRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted receipt filter." << std::endl;
    }
    else {
        std::cerr << "Error deleting receipt filter. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteReceiptFilter in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- For API details, see DeleteReceiptFilter in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def delete_receipt_filter(self, filter_name):
        """
        Deletes a receipt filter.

        :param filter_name: The name of the filter to delete.
        """
        try:
            self.ses_client.delete_receipt_filter(FilterName=filter_name)
            logger.info("Deleted receipt filter %s.", filter_name)
        except ClientError:
            logger.exception("Couldn't delete receipt filter %s.", filter_name)
            raise
```

- For API details, see DeleteReceiptFilter in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->deletereceiptfilter( iv_filtername = iv_filter_name ).
    MESSAGE 'Receipt filter deleted successfully' TYPE 'I'.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
    DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- For API details, see DeleteReceiptFilter in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteReceiptRule` with an AWS SDK

The following code examples show how to use `DeleteReceiptRule`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Email Service (Amazon SES) receipt rule.
/*!
  \param receiptRuleName: The name for the receipt rule.
  \param receiptRuleSetName: The name for the receipt rule set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::deleteReceiptRule(const Aws::String &receiptRuleName,
                                    const Aws::String &receiptRuleSetName,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::DeleteReceiptRuleRequest deleteReceiptRuleRequest;

    deleteReceiptRuleRequest.SetRuleName(receiptRuleName);
    deleteReceiptRuleRequest.SetRuleSetName(receiptRuleSetName);

    Aws::SES::Model::DeleteReceiptRuleOutcome outcome =
 sesClient.DeleteReceiptRule(
            deleteReceiptRuleRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted receipt rule." << std::endl;
    }
    else {
        std::cout << "Error deleting receipt rule. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteReceiptRule in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- For API details, see DeleteReceiptRule in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def delete_receipt_rule(self, rule_set_name, rule_name):
        """
        Deletes a rule.

        :param rule_set_name: The rule set that contains the rule to delete.
        :param rule_name: The rule to delete.
        """
        try:
            self.ses_client.delete_receipt_rule(
                RuleSetName=rule_set_name, RuleName=rule_name
            )
            logger.info("Removed rule %s from rule set %s.", rule_name,
 rule_set_name)
        except ClientError:
            logger.exception(
                "Couldn't remove rule %s from rule set %s.", rule_name,
 rule_set_name
            )
            raise
```

- For API details, see DeleteReceiptRule in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->deletereceiptrule(
      iv_rulesetname = iv_rule_set_name
      iv_rulename = iv_rule_name
    ).
    MESSAGE 'Receipt rule deleted successfully' TYPE 'I'.
  CATCH /aws1/cx_sesrulesetdoesnotexex INTO DATA(lo_ex1).
    DATA(lv_error) = |Rule set does not exist: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see DeleteReceiptRule in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteReceiptRuleSet` with an AWS SDK

The following code examples show how to use `DeleteReceiptRuleSet`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Email Service (Amazon SES) receipt rule set.
/*!
  \param receiptRuleSetName: The name for the receipt rule set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::deleteReceiptRuleSet(const Aws::String &receiptRuleSetName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::DeleteReceiptRuleSetRequest deleteReceiptRuleSetRequest;

    deleteReceiptRuleSetRequest.SetRuleSetName(receiptRuleSetName);

    Aws::SES::Model::DeleteReceiptRuleSetOutcome outcome =
 sesClient.DeleteReceiptRuleSet(
            deleteReceiptRuleSetRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted receipt rule set." << std::endl;
    }

    else {
        std::cerr << "Error deleting receipt rule set. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteReceiptRuleSet in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- For API details, see DeleteReceiptRuleSet in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def delete_receipt_rule_set(self, rule_set_name):
        """
        Deletes a rule set. When a rule set is deleted, all of the rules it
 contains
        are also deleted.

        :param rule_set_name: The name of the rule set to delete.
        """
        try:
            self.ses_client.delete_receipt_rule_set(RuleSetName=rule_set_name)
            logger.info("Deleted rule set %s.", rule_set_name)
        except ClientError:
            logger.exception("Couldn't delete rule set %s.", rule_set_name)
            raise
```

- For API details, see DeleteReceiptRuleSet in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->deletereceiptruleset( iv_rulesetname = iv_rule_set_name ).
    MESSAGE 'Receipt rule set deleted successfully' TYPE 'I'.
  CATCH /aws1/cx_sescannotdeleteex INTO DATA(lo_ex1).
    DATA(lv_error) = |Cannot delete rule set: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see DeleteReceiptRuleSet in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteTemplate` with an AWS SDK

The following code examples show how to use `DeleteTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an email template.
    /// </summary>
    /// <param name="templateName">Name of the template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.DeleteTemplateAsync(
                new DeleteTemplateRequest
                {
                    TemplateName = templateName
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteEmailTemplateAsync failed with exception: "
+ ex.Message);
        }

        return success;
    }
```

- For API details, see DeleteTemplate in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Email Service (Amazon SES) template.
/*!
  \param templateName: The name for the template.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::deleteTemplate(const Aws::String &templateName,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::DeleteTemplateRequest deleteTemplateRequest;

    deleteTemplateRequest.SetTemplateName(templateName);

    Aws::SES::Model::DeleteTemplateOutcome outcome = sesClient.DeleteTemplate(
            deleteTemplateRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted template." << std::endl;
    }
    else {
        std::cerr << "Error deleting template. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteTemplate in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- For API details, see DeleteTemplate in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.

        :param subject: The subject of the email.
        :param text: The text version of the email.
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
 html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def delete_template(self):
        """
        Deletes an email template.
        """
        try:

 self.ses_client.delete_template(TemplateName=self.template["TemplateName"])
            logger.info("Deleted template %s.", self.template["TemplateName"])
```

```
            self.template = None
            self.template_tags = None
        except ClientError:
            logger.exception(
                "Couldn't delete template %s.", self.template["TemplateName"]
            )
            raise
```

- For API details, see DeleteTemplate in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->deletetemplate( iv_templatename = iv_template_name ).
    MESSAGE 'Template deleted successfully' TYPE 'I'.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
    DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- For API details, see DeleteTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeReceiptRuleSet` with an AWS SDK

The following code examples show how to use `DescribeReceiptRuleSet`.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""

    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def describe_receipt_rule_set(self, rule_set_name):
        """
        Gets data about a rule set.

        :param rule_set_name: The name of the rule set to retrieve.
        :return: Data about the rule set.
        """
        try:
            response = self.ses_client.describe_receipt_rule_set(
                RuleSetName=rule_set_name
            )
            logger.info("Got data for rule set %s.", rule_set_name)
        except ClientError:
            logger.exception("Couldn't get data for rule set %s.", rule_set_name)
            raise
        else:
            return response
```

- For API details, see DescribeReceiptRuleSet in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_ses->describereceiptruleset(
      iv_rulesetname = iv_rule_set_name
    ).
    MESSAGE 'Receipt rule set described successfully' TYPE 'I'.
  CATCH /aws1/cx_sesrulesetdoesnotexex INTO DATA(lo_ex1).
    DATA(lv_error) = |Rule set does not exist: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see DescribeReceiptRuleSet in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetIdentityVerificationAttributes` with an AWS SDK or CLI

The following code examples show how to use `GetIdentityVerificationAttributes`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Setting up email sending services
- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get identity verification status for an email.
    /// </summary>
    /// <returns>The verification status of the email.</returns>
    public async Task<VerificationStatus> GetIdentityStatusAsync(string email)
    {
        var result = VerificationStatus.TemporaryFailure;
        try
        {
            var response =
                await
_amazonSimpleEmailService.GetIdentityVerificationAttributesAsync(
                    new GetIdentityVerificationAttributesRequest
                    {
                        Identities = new List<string> { email }
                    });

            if (response.VerificationAttributes.ContainsKey(email))
                result =
response.VerificationAttributes[email].VerificationStatus;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine("GetIdentityStatusAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- For API details, see GetIdentityVerificationAttributes in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get the Amazon SES verification status for a list of identities

The following example uses the `get-identity-verification-attributes` command to retrieve the Amazon SES verification status for a list of identities:

```
aws ses get-identity-verification-attributes --
identities "user1@example.com" "user2@example.com"
```

Output:

```
{
   "VerificationAttributes": {
       "user1@example.com": {
           "VerificationStatus": "Success"
       },
       "user2@example.com": {
           "VerificationStatus": "Pending"
       }
   }
}
```

If you call this command with an identity that you have never submitted for verification, that identity won't appear in the output.

For more information about verified identities, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

- For API details, see GetIdentityVerificationAttributes in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def get_identity_status(self, identity):
        """
        Gets the status of an identity. This can be used to discover whether
        an identity has been successfully verified.

        :param identity: The identity to query.
        :return: The status of the identity.
        """
        try:
            response = self.ses_client.get_identity_verification_attributes(
                Identities=[identity]
            )
            status = response["VerificationAttributes"].get(
                identity, {"VerificationStatus": "NotFound"}
            )["VerificationStatus"]
            logger.info("Got status of %s for %s.", status, identity)
        except ClientError:
            logger.exception("Couldn't get status for %s.", identity)
            raise
        else:
```

```
            return status
```

- For API details, see GetIdentityVerificationAttributes in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-ses' # v2: require 'aws-sdk'

# Create client in us-west-2 region
# Replace us-west-2 with the AWS Region you're using for Amazon SES.
client = Aws::SES::Client.new(region: 'us-west-2')

# Get up to 1000 identities
ids = client.list_identities({
                               identity_type: 'EmailAddress'
                             })

ids.identities.each do |email|
  attrs = client.get_identity_verification_attributes({
                                                        identities: [email]
                                                      })

  status = attrs.verification_attributes[email].verification_status

  # Display email addresses that have been verified
  puts email if status == 'Success'
end
```

- For API details, see GetIdentityVerificationAttributes in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   DATA lt_identities TYPE /aws1/cl_sesidentitylist_w=>tt_identitylist.
   APPEND NEW /aws1/cl_sesidentitylist_w( iv_value = iv_identity ) TO
lt_identities.

   TRY.
       DATA(lo_result) = lo_ses->getidentityverificationattrs(
         it_identities = lt_identities
       ).

       DATA(lt_attrs) = lo_result->get_verificationattributes( ).
       IF lt_attrs IS NOT INITIAL.
         LOOP AT lt_attrs ASSIGNING FIELD-SYMBOL(<ls_attr>).
           ov_status = <ls_attr>-value->get_verificationstatus( ).
           EXIT.
         ENDLOOP.
       ELSE.
         ov_status = 'NotFound'.
       ENDIF.
     CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
       DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
       MESSAGE lv_error TYPE 'I'.
       RAISE EXCEPTION lo_ex.
   ENDTRY.
```

- For API details, see GetIdentityVerificationAttributes in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetSendQuota` with an AWS SDK or CLI

The following code examples show how to use `GetSendQuota`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Setting up email sending services

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get information on the current account's send quota.
    /// </summary>
    /// <returns>The send quota response data.</returns>
    public async Task<GetSendQuotaResponse> GetSendQuotaAsync()
    {
        var result = new GetSendQuotaResponse();
        try
        {
            var response = await _amazonSimpleEmailService.GetSendQuotaAsync(
                new GetSendQuotaRequest());
            result = response;
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetSendQuotaAsync failed with exception: " +
ex.Message);
```

```
        }

        return result;
    }
```

- For API details, see GetSendQuota in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get your Amazon SES sending limits

The following example uses the `get-send-quota` command to return your Amazon SES sending limits:

```
aws ses get-send-quota
```

Output:

```
{
   "Max24HourSend": 200.0,
   "SentLast24Hours": 1.0,
   "MaxSendRate": 1.0
}
```

Max24HourSend is your sending quota, which is the maximum number of emails that you can send in a 24-hour period. The sending quota reflects a rolling time period. Every time you try to send an email, Amazon SES checks how many emails you sent in the previous 24 hours. As long as the total number of emails that you have sent is less than your quota, your send request will be accepted and your email will be sent.

SentLast24Hours is the number of emails that you have sent in the previous 24 hours.

MaxSendRate is the maximum number of emails that you can send per second.

Note that sending limits are based on recipients rather than on messages. For example, an email that has 10 recipients counts as 10 against your sending quota.

For more information, see Managing Your Amazon SES Sending Limits in the *Amazon Simple Email Service Developer Guide*.

- For API details, see GetSendQuota in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the user's current sending limits.

```
Get-SESSendQuota
```

- For API details, see GetSendQuota in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the user's current sending limits.

```
Get-SESSendQuota
```

- For API details, see GetSendQuota in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetSendStatistics` with a CLI

The following code examples show how to use `GetSendStatistics`.

CLI

AWS CLI

To get your Amazon SES sending statistics

The following example uses the `get-send-statistics` command to return your Amazon SES sending statistics

```
aws ses get-send-statistics
```

Output:

```
{
   "SendDataPoints": [
       {
           "Complaints": 0,
           "Timestamp": "2013-06-12T19:32:00Z",
           "DeliveryAttempts": 2,
           "Bounces": 0,
           "Rejects": 0
       },
       {
           "Complaints": 0,
           "Timestamp": "2013-06-12T00:47:00Z",
           "DeliveryAttempts": 1,
           "Bounces": 0,
           "Rejects": 0
       }
   ]
}
```

The result is a list of data points, representing the last two weeks of sending activity. Each data point in the list contains statistics for a 15-minute interval.

In this example, there are only two data points because the only emails that the user sent in the last two weeks fell within two 15-minute intervals.

For more information, see Monitoring Your Amazon SES Usage Statistics in the *Amazon Simple Email Service Developer Guide*.

- For API details, see GetSendStatistics in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the user's sending statistics. The result is a list of data points, representing the last two weeks of sending activity. Each data point in the list contains statistics for a 15-minute interval.

```
Get-SESSendStatistic
```

- For API details, see GetSendStatistics in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the user's sending statistics. The result is a list of data points, representing the last two weeks of sending activity. Each data point in the list contains statistics for a 15-minute interval.

```
Get-SESSendStatistic
```

- For API details, see GetSendStatistics in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetTemplate` with an AWS SDK

The following code examples show how to use `GetTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Get a template's attributes.
/*!
  \param templateName: The name for the template.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::getTemplate(const Aws::String &templateName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::GetTemplateRequest getTemplateRequest;

    getTemplateRequest.SetTemplateName(templateName);

    Aws::SES::Model::GetTemplateOutcome outcome = sesClient.GetTemplate(
            getTemplateRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully got template." << std::endl;
    }

    else {
        std::cerr << "Error getting template. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see GetTemplate in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
```

```
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- For API details, see GetTemplate in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
```

```
        """
        self.ses_client = ses_client
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.

        :param subject: The subject of the email.
        :param text: The text version of the email.
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def get_template(self, name):
        """
        Gets a previously created email template.

        :param name: The name of the template to retrieve.
        :return: The retrieved email template.
        """
        try:
            response = self.ses_client.get_template(TemplateName=name)
            self.template = response["Template"]
            logger.info("Got template %s.", name)
            self._extract_tags(
                self.template["SubjectPart"],
                self.template["TextPart"],
                self.template["HtmlPart"],
            )
        except ClientError:
            logger.exception("Couldn't get template %s.", name)
            raise
        else:
            return self.template
```

- For API details, see GetTemplate in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_ses->gettemplate( iv_templatename =
iv_template_name ).
        oo_template = lo_result->get_template( ).
        MESSAGE 'Template retrieved successfully' TYPE 'I'.
      CATCH /aws1/cx_sestmpldoesnotexistex INTO DATA(lo_ex1).
        DATA(lv_error) = |Template does not exist: { lo_ex1->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex1.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
        lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex_generic.
    ENDTRY.
```

- For API details, see GetTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListIdentities` with an AWS SDK or CLI

The following code examples show how to use `ListIdentities`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Copy email and domain identities across Regions

- Setting up email sending services
- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get the identities of a specified type for the current account.
   /// </summary>
   /// <param name="identityType">IdentityType to list.</param>
   /// <returns>The list of identities.</returns>
   public async Task<List<string>> ListIdentitiesAsync(IdentityType
identityType)
   {
       var result = new List<string>();
       try
       {
           var response = await _amazonSimpleEmailService.ListIdentitiesAsync(
               new ListIdentitiesRequest
               {
                   IdentityType = identityType
               });
           result = response.Identities;
       }
       catch (Exception ex)
       {
           Console.WriteLine("ListIdentitiesAsync failed with exception: " +
ex.Message);
       }

       return result;
   }
```

- For API details, see ListIdentities in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! List the identities associated with this account.
/*!
  \param identityType: The identity type enum. "NOT_SET" is a valid option.
  \param identities; A vector to receive the retrieved identities.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::listIdentities(Aws::SES::Model::IdentityType identityType,
                                 Aws::Vector<Aws::String> &identities,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::ListIdentitiesRequest listIdentitiesRequest;

    if (identityType != Aws::SES::Model::IdentityType::NOT_SET) {
        listIdentitiesRequest.SetIdentityType(identityType);
    }

    Aws::String nextToken; // Used for paginated results.
    do {
        if (!nextToken.empty()) {
            listIdentitiesRequest.SetNextToken(nextToken);
        }
        Aws::SES::Model::ListIdentitiesOutcome outcome =
 sesClient.ListIdentities(
                listIdentitiesRequest);
```

```
        if (outcome.IsSuccess()) {
            const auto &retrievedIdentities =
 outcome.GetResult().GetIdentities();
            if (!retrievedIdentities.empty()) {
                identities.insert(identities.cend(),
 retrievedIdentities.cbegin(),
                                  retrievedIdentities.cend());
            }
            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cout << "Error listing identities. " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    return true;
}
```

- For API details, see ListIdentities in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list all identities (email addresses and domains) for a specific AWS account

The following example uses the `list-identities` command to list all identities that have been submitted for verification with Amazon SES:

```
aws ses list-identities
```

Output:

```
{
    "Identities": [
      "user@example.com",
      "example.com"
```

```
    ]
}
```

The list that is returned contains all identities regardless of verification status (verified, pending verification, failure, etc.).

In this example, email addresses *and* domains are returned because we did not specify the identity-type parameter.

For more information about verification, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

- For API details, see ListIdentities in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ses.SesClient;
import software.amazon.awssdk.services.ses.model.ListIdentitiesResponse;
import software.amazon.awssdk.services.ses.model.SesException;
import java.io.IOException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListIdentities {
```

```
    public static void main(String[] args) throws IOException {
        Region region = Region.US_WEST_2;
        SesClient client = SesClient.builder()
                .region(region)
                .build();

        listSESIdentities(client);
    }

    public static void listSESIdentities(SesClient client) {
        try {
            ListIdentitiesResponse identitiesResponse = client.listIdentities();
            List<String> identities = identitiesResponse.identities();
            for (String identity : identities) {
                System.out.println("The identity is " + identity);
            }

        } catch (SesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListIdentities in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
```

```
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- For API details, see ListIdentities in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns a list containing all of the identities (email addresses and domains) for a specific AWS Account, regardless of verification status.

```
Get-SESIdentity
```

- For API details, see ListIdentities in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns a list containing all of the identities (email addresses and domains) for a specific AWS Account, regardless of verification status.

```
Get-SESIdentity
```

- For API details, see ListIdentities in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def list_identities(self, identity_type, max_items):
        """
        Gets the identities of the specified type for the current account.

        :param identity_type: The type of identity to retrieve, such as
 EmailAddress.
        :param max_items: The maximum number of identities to retrieve.
        :return: The list of retrieved identities.
        """
        try:
            response = self.ses_client.list_identities(
                IdentityType=identity_type, MaxItems=max_items
            )
            identities = response["Identities"]
            logger.info("Got %s identities for the current account.",
 len(identities))
        except ClientError:
            logger.exception("Couldn't list identities for the current account.")
            raise
        else:
            return identities
```

- For API details, see ListIdentities in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-ses' # v2: require 'aws-sdk'

# Create client in us-west-2 region
# Replace us-west-2 with the AWS Region you're using for Amazon SES.
client = Aws::SES::Client.new(region: 'us-west-2')

# Get up to 1000 identities
ids = client.list_identities({
                               identity_type: 'EmailAddress'
                             })

ids.identities.each do |email|
  attrs = client.get_identity_verification_attributes({
                                                        identities: [email]
                                                      })

  status = attrs.verification_attributes[email].verification_status

  # Display email addresses that have been verified
  puts email if status == 'Success'
end
```

- For API details, see ListIdentities in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    DATA(lo_result) = lo_ses->listidentities(
      iv_identitytype = iv_identity_type
      iv_maxitems = iv_max_items
    ).
    ot_identities = lo_result->get_identities( ).
    MESSAGE 'Identities retrieved successfully' TYPE 'I'.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
    DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- For API details, see ListIdentities in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListReceiptFilters` with an AWS SDK

The following code examples show how to use `ListReceiptFilters`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! List the receipt filters associated with this account.
/*!
  \param filters; A vector of "ReceiptFilter" to receive the retrieved filters.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SES::listReceiptFilters(Aws::Vector<Aws::SES::Model::ReceiptFilter>
 &filters,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);
    Aws::SES::Model::ListReceiptFiltersRequest listReceiptFiltersRequest;

    Aws::SES::Model::ListReceiptFiltersOutcome outcome =
 sesClient.ListReceiptFilters(
            listReceiptFiltersRequest);
    if (outcome.IsSuccess()) {
        auto &retrievedFilters = outcome.GetResult().GetFilters();
        if (!retrievedFilters.empty()) {
            filters.insert(filters.cend(), retrievedFilters.cbegin(),
                           retrievedFilters.cend());
        }
    }
    else {
        std::cerr << "Error retrieving IP address filters: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see ListReceiptFilters in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- For API details, see ListReceiptFilters in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesReceiptHandler:
    """Encapsulates Amazon SES receipt handling functions."""
```

```
    def __init__(self, ses_client, s3_resource):
        """
        :param ses_client: A Boto3 Amazon SES client.
        :param s3_resource: A Boto3 Amazon S3 resource.
        """
        self.ses_client = ses_client
        self.s3_resource = s3_resource


    def list_receipt_filters(self):
        """
        Gets the list of receipt filters for the current account.

        :return: The list of receipt filters.
        """
        try:
            response = self.ses_client.list_receipt_filters()
            filters = response["Filters"]
            logger.info("Got %s receipt filters.", len(filters))
        except ClientError:
            logger.exception("Couldn't get receipt filters.")
            raise
        else:
            return filters
```

- For API details, see ListReceiptFilters in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_ses->listreceiptfilters( ).
        ot_filters = lo_result->get_filters( ).
```

```
      MESSAGE 'Receipt filters retrieved successfully' TYPE 'I'.
    CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
      DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
      MESSAGE lv_error TYPE 'I'.
      RAISE EXCEPTION lo_ex.
  ENDTRY.
```

- For API details, see ListReceiptFilters in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListTemplates` with an AWS SDK

The following code examples show how to use `ListTemplates`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// List email templates for the current account.
/// </summary>
/// <returns>A list of template metadata.</returns>
public async Task<List<TemplateMetadata>> ListEmailTemplatesAsync()
{
    var result = new List<TemplateMetadata>();
```

```
        try
        {
            var response = await _amazonSimpleEmailService.ListTemplatesAsync(
                new ListTemplatesRequest());
            result = response.TemplatesMetadata;
        }
        catch (Exception ex)
        {
            Console.WriteLine("ListEmailTemplatesAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- For API details, see ListTemplates in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sesv2.SesV2Client;
import software.amazon.awssdk.services.sesv2.model.ListEmailTemplatesRequest;
import software.amazon.awssdk.services.sesv2.model.ListEmailTemplatesResponse;
import software.amazon.awssdk.services.sesv2.model.SesV2Exception;

public class ListTemplates {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        SesV2Client sesv2Client = SesV2Client.builder()
                .region(region)
                .build();
```

```
        listAllTemplates(sesv2Client);
    }

    public static void listAllTemplates(SesV2Client sesv2Client) {
        try {
            ListEmailTemplatesRequest templatesRequest =
 ListEmailTemplatesRequest.builder()
                    .pageSize(1)
                    .build();

            ListEmailTemplatesResponse response =
 sesv2Client.listEmailTemplates(templatesRequest);
            response.templatesMetadata()
                    .forEach(template -> System.out.println("Template name: " +
 template.templateName()));

        } catch (SesV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListTemplates in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });
```

```
const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- For API details, see ListTemplates in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.

        :param subject: The subject of the email.
        :param text: The text version of the email.
```

```
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def list_templates(self):
        """
        Gets a list of all email templates for the current account.

        :return: The list of retrieved email templates.
        """
        try:
            response = self.ses_client.list_templates()
            templates = response["TemplatesMetadata"]
            logger.info("Got %s templates.", len(templates))
        except ClientError:
            logger.exception("Couldn't get templates.")
            raise
        else:
            return templates
```

- For API details, see ListTemplates in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_ses->listtemplates( iv_maxitems = iv_max_items ).
        ot_templates = lo_result->get_templatesmetadata( ).
        MESSAGE 'Templates retrieved successfully' TYPE 'I'.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
```

```
        DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- For API details, see ListTemplates in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendBulkTemplatedEmail` with an AWS SDK

The following code example shows how to use `SendBulkTemplatedEmail`.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
```

```
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can
 map each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</
p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!
</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!
</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
```

```
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- For API details, see SendBulkTemplatedEmail in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendEmail` with an AWS SDK or CLI

The following code examples show how to use `SendEmail`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Setting up email sending services
- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
///  Send an email by using Amazon SES.
/// </summary>
/// <param name="toAddresses">List of recipients.</param>
/// <param name="ccAddresses">List of cc recipients.</param>
/// <param name="bccAddresses">List of bcc recipients.</param>
/// <param name="bodyHtml">Body of the email in HTML.</param>
/// <param name="bodyText">Body of the email in plain text.</param>
/// <param name="subject">Subject line of the email.</param>
/// <param name="senderAddress">From address.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendEmailAsync(List<string> toAddresses,
    List<string> ccAddresses, List<string> bccAddresses,
    string bodyHtml, string bodyText, string subject, string senderAddress)
{
    var messageId = "";
    try
    {
        var response = await _amazonSimpleEmailService.SendEmailAsync(
            new SendEmailRequest
            {
                Destination = new Destination
                {
                    BccAddresses = bccAddresses,
                    CcAddresses = ccAddresses,
                    ToAddresses = toAddresses
                },
                Message = new Message
                {
                    Body = new Body
                    {
                        Html = new Content
                        {
                            Charset = "UTF-8",
                            Data = bodyHtml
                        },
                        Text = new Content
                        {
                            Charset = "UTF-8",
                            Data = bodyText
                        }
                    },
```

```
                        Subject = new Content
                        {
                            Charset = "UTF-8",
                            Data = subject
                        }
                    },
                    Source = senderAddress
                });
            messageId = response.MessageId;
        }
        catch (Exception ex)
        {
            Console.WriteLine("SendEmailAsync failed with exception: " +
ex.Message);
        }

        return messageId;
    }
```

- For API details, see SendEmail in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Send an email to a list of recipients.
/*!
  \param recipients; Vector of recipient email addresses.
  \param subject: Email subject.
  \param htmlBody: Email body as HTML. At least one body data is required.
  \param textBody: Email body as plain text. At least one body data is required.
  \param senderEmailAddress: Email address of sender. Ignored if empty string.
  \param ccAddresses: Vector of cc addresses. Ignored if empty.
  \param replyToAddress: Reply to email address. Ignored if empty string.
```

```
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::sendEmail(const Aws::Vector<Aws::String> &recipients,
                            const Aws::String &subject,
                            const Aws::String &htmlBody,
                            const Aws::String &textBody,
                            const Aws::String &senderEmailAddress,
                            const Aws::Vector<Aws::String> &ccAddresses,
                            const Aws::String &replyToAddress,
                            const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::Destination destination;
    if (!ccAddresses.empty()) {
        destination.WithCcAddresses(ccAddresses);
    }
    if (!recipients.empty()) {
        destination.WithToAddresses(recipients);
    }

    Aws::SES::Model::Body message_body;
    if (!htmlBody.empty()) {
        message_body.SetHtml(

 Aws::SES::Model::Content().WithCharset("UTF-8").WithData(htmlBody));
    }

    if (!textBody.empty()) {
        message_body.SetText(

 Aws::SES::Model::Content().WithCharset("UTF-8").WithData(textBody));
    }

    Aws::SES::Model::Message message;
    message.SetBody(message_body);
    message.SetSubject(
            Aws::SES::Model::Content().WithCharset("UTF-8").WithData(subject));

    Aws::SES::Model::SendEmailRequest sendEmailRequest;
    sendEmailRequest.SetDestination(destination);
    sendEmailRequest.SetMessage(message);
    if (!senderEmailAddress.empty()) {
```

```
        sendEmailRequest.SetSource(senderEmailAddress);
    }
    if (!replyToAddress.empty()) {
        sendEmailRequest.AddReplyToAddresses(replyToAddress);
    }

    auto outcome = sesClient.SendEmail(sendEmailRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully sent message with ID "
                  << outcome.GetResult().GetMessageId()
                  << "." << std::endl;
    }
    else {
        std::cerr << "Error sending message. " << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see SendEmail in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To send a formatted email using Amazon SES

The following example uses the `send-email` command to send a formatted email:

```
aws ses send-email --from sender@example.com --destination file://
destination.json --message file://message.json
```

Output:

```
{
   "MessageId": "EXAMPLEf3a5efcd1-51adec81-d2a4-4e3f-9fe2-5d85c1b23783-000000"
}
```

The destination and the message are JSON data structures saved in .json files in the current directory. These files are as follows:

`destination.json`:

```
{
  "ToAddresses":  ["recipient1@example.com", "recipient2@example.com"],
  "CcAddresses":  ["recipient3@example.com"],
  "BccAddresses": []
}
```

`message.json`:

```
{
   "Subject": {
       "Data": "Test email sent using the AWS CLI",
       "Charset": "UTF-8"
   },
   "Body": {
       "Text": {
           "Data": "This is the message body in text format.",
           "Charset": "UTF-8"
       },
       "Html": {
           "Data": "This message body contains HTML formatting. It can, for
 example, contain links like this one: <a class=\"ulink\" href=\"http://
docs.aws.amazon.com/ses/latest/DeveloperGuide\" target=\"_blank\">Amazon SES
 Developer Guide</a>.",
           "Charset": "UTF-8"
       }
   }
}
```

Replace the sender and recipient email addresses with the ones you want to use. Note that the sender's email address must be verified with Amazon SES. Until you are granted production access to Amazon SES, you must also verify the email address of each recipient unless the recipient is the Amazon SES mailbox simulator. For more information on verification, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

The Message ID in the output indicates that the call to send-email was successful.

If you don't receive the email, check your Junk box.

For more information on sending formatted email, see Sending Formatted Email Using the Amazon SES API in the *Amazon Simple Email Service Developer Guide*.

- For API details, see SendEmail in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ses.SesClient;
import software.amazon.awssdk.services.ses.model.Content;
import software.amazon.awssdk.services.ses.model.Destination;
import software.amazon.awssdk.services.ses.model.Message;
import software.amazon.awssdk.services.ses.model.Body;
import software.amazon.awssdk.services.ses.model.SendEmailRequest;
import software.amazon.awssdk.services.ses.model.SesException;

import javax.mail.MessagingException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SendMessageEmailRequest {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <sender> <recipient> <subject>\s

                Where:
                    sender - An email address that represents the sender.\s
                    recipient -  An email address that represents the recipient.
\s
                    subject - The  subject line.\s
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String sender = args[0];
        String recipient = args[1];
        String subject = args[2];

        Region region = Region.US_EAST_1;
        SesClient client = SesClient.builder()
                .region(region)
                .build();

        // The HTML body of the email.
        String bodyHTML = "<html>" + "<head></head>" + "<body>" + "<h1>Hello!</
h1>"
                + "<p> See the list of customers.</p>" + "</body>" + "</html>";

        try {
            send(client, sender, recipient, subject, bodyHTML);
            client.close();
            System.out.println("Done");

        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }

    public static void send(SesClient client,
            String sender,
            String recipient,
            String subject,
            String bodyHTML) throws MessagingException {
```

```
        Destination destination = Destination.builder()
                .toAddresses(recipient)
                .build();

        Content content = Content.builder()
                .data(bodyHTML)
                .build();

        Content sub = Content.builder()
                .data(subject)
                .build();

        Body body = Body.builder()
                .html(content)
                .build();

        Message msg = Message.builder()
                .subject(sub)
                .body(body)
                .build();

        SendEmailRequest emailRequest = SendEmailRequest.builder()
                .destination(destination)
                .message(msg)
                .source(sender)
                .build();

        try {
            System.out.println("Attempting to send an email through Amazon SES "
 + "using the AWS SDK for Java...");
            client.sendEmail(emailRequest);

        } catch (SesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ses.SesClient;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.mail.Message;
```

```
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.internet.MimeBodyPart;
import javax.mail.util.ByteArrayDataSource;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.file.Files;
import java.util.Properties;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.ses.model.SendRawEmailRequest;
import software.amazon.awssdk.services.ses.model.RawMessage;
import software.amazon.awssdk.services.ses.model.SesException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class SendMessageAttachment {
    public static void main(String[] args) throws IOException {
        final String usage = """

                Usage:
                    <sender> <recipient> <subject> <fileLocation>\s

                Where:
                    sender - An email address that represents the sender.\s
                    recipient -  An email address that represents the recipient.
\s
                    subject - The  subject line.\s
                    fileLocation - The location of a Microsoft Excel file to use
 as an attachment (C:/AWS/customers.xls).\s
                """;
```

```
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String sender = args[0];
        String recipient = args[1];
        String subject = args[2];
        String fileLocation = args[3];

        // The email body for recipients with non-HTML email clients.
        String bodyText = "Hello,\r\n" + "Please see the attached file for a list
 "
                + "of customers to contact.";

        // The HTML body of the email.
        String bodyHTML = "<html>" + "<head></head>" + "<body>" + "<h1>Hello!</
h1>"
                + "<p>Please see the attached file for a " + "list of customers
 to contact.</p>" + "</body>"
                + "</html>";

        Region region = Region.US_WEST_2;
        SesClient client = SesClient.builder()
                .region(region)
                .build();

        try {
            sendemailAttachment(client, sender, recipient, subject, bodyText,
 bodyHTML, fileLocation);
            client.close();
            System.out.println("Done");

        } catch (IOException | MessagingException e) {
            e.printStackTrace();
        }
    }

    public static void sendemailAttachment(SesClient client,
            String sender,
            String recipient,
            String subject,
            String bodyText,
            String bodyHTML,
```

```
            String fileLocation) throws AddressException, MessagingException,
IOException {

        java.io.File theFile = new java.io.File(fileLocation);
        byte[] fileContent = Files.readAllBytes(theFile.toPath());

        Session session = Session.getDefaultInstance(new Properties());

        // Create a new MimeMessage object.
        MimeMessage message = new MimeMessage(session);

        // Add subject, from and to lines.
        message.setSubject(subject, "UTF-8");
        message.setFrom(new InternetAddress(sender));
        message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(recipient));

        // Create a multipart/alternative child container.
        MimeMultipart msgBody = new MimeMultipart("alternative");

        // Create a wrapper for the HTML and text parts.
        MimeBodyPart wrap = new MimeBodyPart();

        // Define the text part.
        MimeBodyPart textPart = new MimeBodyPart();
        textPart.setContent(bodyText, "text/plain; charset=UTF-8");

        // Define the HTML part.
        MimeBodyPart htmlPart = new MimeBodyPart();
        htmlPart.setContent(bodyHTML, "text/html; charset=UTF-8");

        // Add the text and HTML parts to the child container.
        msgBody.addBodyPart(textPart);
        msgBody.addBodyPart(htmlPart);

        // Add the child container to the wrapper object.
        wrap.setContent(msgBody);

        // Create a multipart/mixed parent container.
        MimeMultipart msg = new MimeMultipart("mixed");

        // Add the parent container to the message.
        message.setContent(msg);
        msg.addBodyPart(wrap);
```

```
        // Define the attachment.
        MimeBodyPart att = new MimeBodyPart();
        DataSource fds = new ByteArrayDataSource(fileContent,
                "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
        att.setDataHandler(new DataHandler(fds));

        String reportName = "WorkReport.xls";
        att.setFileName(reportName);

        // Add the attachment to the message.
        msg.addBodyPart(att);

        try {
            System.out.println("Attempting to send an email through Amazon SES "
 + "using the AWS SDK for Java...");

            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            message.writeTo(outputStream);

            ByteBuffer buf = ByteBuffer.wrap(outputStream.toByteArray());

            byte[] arr = new byte[buf.remaining()];
            buf.get(arr);

            SdkBytes data = SdkBytes.fromByteArray(arr);
            RawMessage rawMessage = RawMessage.builder()
                    .data(data)
                    .build();

            SendRawEmailRequest rawEmailRequest = SendRawEmailRequest.builder()
                    .rawMessage(rawMessage)
                    .build();

            client.sendRawEmail(rawEmailRequest);

        } catch (SesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Email sent using SesClient with attachment");
    }
}
```

- For API details, see SendEmail in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
```

```
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- For API details, see SendEmail in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesMailSender:
    """Encapsulates functions to send emails with Amazon SES."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def send_email(self, source, destination, subject, text, html,
 reply_tos=None):
        """
        Sends an email.

        Note: If your account is in the Amazon SES  sandbox, the source and
        destination email accounts must both be verified.

        :param source: The source email account.
        :param destination: The destination email account.
        :param subject: The subject of the email.
        :param text: The plain text version of the body of the email.
        :param html: The HTML version of the body of the email.
        :param reply_tos: Email accounts that will receive a reply if the
 recipient
                          replies to the message.
        :return: The ID of the message, assigned by Amazon SES.
        """
        send_args = {
            "Source": source,
            "Destination": destination.to_service_format(),
            "Message": {
                "Subject": {"Data": subject},
                "Body": {"Text": {"Data": text}, "Html": {"Data": html}},
            },
        }
        if reply_tos is not None:
            send_args["ReplyToAddresses"] = reply_tos
        try:
            response = self.ses_client.send_email(**send_args)
            message_id = response["MessageId"]
            logger.info(
```

```
                "Sent mail %s from %s to %s.", message_id, source,
 destination.tos
            )
        except ClientError:
            logger.exception(
                "Couldn't send mail from %s to %s.", source, destination.tos
            )
            raise
        else:
            return message_id
```

- For API details, see SendEmail in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-ses' # v2: require 'aws-sdk'

# Replace sender@example.com with your "From" address.
# This address must be verified with Amazon SES.
sender = 'sender@example.com'

# Replace recipient@example.com with a "To" address. If your account
# is still in the sandbox, this address must be verified.
recipient = 'recipient@example.com'

# Specify a configuration set. To use a configuration
# set, uncomment the next line and line 74.
#   configsetname = "ConfigSet"

# The subject line for the email.
subject = 'Amazon SES test (AWS SDK for Ruby)'
```

```
# The HTML body of the email.
htmlbody =
  '<h1>Amazon SES test (AWS SDK for Ruby)</h1>'\
  '<p>This email was sent with <a href="https://aws.amazon.com/ses/">'\
  'Amazon SES</a> using the <a href="https://aws.amazon.com/sdk-for-ruby/">'\
  'AWS SDK for Ruby</a>.'

# The email body for recipients with non-HTML email clients.
textbody = 'This email was sent with Amazon SES using the AWS SDK for Ruby.'

# Specify the text encoding scheme.
encoding = 'UTF-8'

# Create a new SES client in the us-west-2 region.
# Replace us-west-2 with the AWS Region you're using for Amazon SES.
ses = Aws::SES::Client.new(region: 'us-west-2')

# Try to send the email.
begin
  # Provide the contents of the email.
  ses.send_email(
    destination: {
      to_addresses: [
        recipient
      ]
    },
    message: {
      body: {
        html: {
          charset: encoding,
          data: htmlbody
        },
        text: {
          charset: encoding,
          data: textbody
        }
      },
      subject: {
        charset: encoding,
        data: subject
      }
    },
    source: sender
```

```
    # Uncomment the following line to use a configuration set.
    # configuration_set_name: configsetname,
  )

  puts "Email sent to #{recipient}"

# If something goes wrong, display an error message.
rescue Aws::SES::Errors::ServiceError => e
  puts "Email not sent. Error message: #{e}"
end
```

- For API details, see SendEmail in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    " Create message object
    DATA(lo_subject) = NEW /aws1/cl_sescontent( iv_data = iv_subject ).
    DATA(lo_text_body) = NEW /aws1/cl_sescontent( iv_data = iv_text ).
    DATA(lo_html_body) = NEW /aws1/cl_sescontent( iv_data = iv_html ).
    DATA(lo_body) = NEW /aws1/cl_sesbody(
      io_text = lo_text_body
      io_html = lo_html_body
    ).
    DATA(lo_message) = NEW /aws1/cl_sesmessage(
      io_subject = lo_subject
      io_body = lo_body
    ).

    TRY.
        " Send email
        DATA(lo_result) = lo_ses->sendemail(
          iv_source = iv_source
          io_destination = io_destination
```

```
          io_message = lo_message
          it_replytoaddresses = it_reply_tos
        ).
        ov_msg_id = lo_result->get_messageid( ).
        MESSAGE 'Email sent successfully' TYPE 'I'.
      CATCH /aws1/cx_sesacctsendingpause00 INTO DATA(lo_ex1).
        DATA(lv_error) = |Account sending paused: { lo_ex1->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex1.
      CATCH /aws1/cx_sesmessagerejected INTO DATA(lo_ex2).
        lv_error = |Message rejected: { lo_ex2->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex2.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
        lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex_generic.
    ENDTRY.
```

- For API details, see SendEmail in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendRawEmail` with an AWS SDK or CLI

The following code examples show how to use `SendRawEmail`.

CLI

AWS CLI

To send a raw email using Amazon SES

The following example uses the `send-raw-email` command to send an email with a TXT attachment:

```
aws ses send-raw-email \
--cli-binary-format raw-in-base64-out \
--raw-message file://message.json
```

Output:

```
{
   "MessageId": "EXAMPLEf3f73d99b-c63fb06f-d263-41f8-a0fb-d0dc67d56c07-000000"
}
```

The raw message is a JSON data structure saved in a file named `message.json` in the current directory. It contains the following:

```
{
   "Data": "From: sender@example.com\nTo: recipient@example.com\nSubject:
 Test email sent using the AWS CLI (contains an attachment)\nMIME-Version:
 1.0\nContent-type: Multipart/Mixed; boundary=\"NextPart\"\n\n--NextPart
\nContent-Type: text/plain\n\nThis is the message body.\n\n--NextPart\nContent-
Type: text/plain;\nContent-Disposition: attachment; filename=\"attachment.txt\"\n
\nThis is the text in the attachment.\n\n--NextPart--"
}
```

As you can see, "Data" is one long string that contains the entire raw email content in MIME format, including an attachment called attachment.txt.

Replace sender@example.com and recipient@example.com with the addresses you want to use. Note that the sender's email address must be verified with Amazon SES. Until you are granted production access to Amazon SES, you must also verify the email address of the recipient unless the recipient is the Amazon SES mailbox simulator. For more information on verification, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

The Message ID in the output indicates that the call to send-raw-email was successful.

If you don't receive the email, check your Junk box.

For more information on sending raw email, see Sending Raw Email Using the Amazon SES API in the *Amazon Simple Email Service Developer Guide*.

- For API details, see SendRawEmail in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use nodemailer to send an email with an attachment.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more
 advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
```

```
      },
      (err, info) => {
        if (err) {
          reject(err);
        } else {
          resolve(info);
        }
      },
    );
  });
};
```

- For API details, see SendRawEmail in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendTemplatedEmail` with an AWS SDK

The following code examples show how to use `SendTemplatedEmail`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
```

```
    /// Send an email using a template.
    /// </summary>
    /// <param name="sender">Address of the sender.</param>
    /// <param name="recipients">Addresses of the recipients.</param>
    /// <param name="templateName">Name of the email template.</param>
    /// <param name="templateDataObject">Data for the email template.</param>
    /// <returns>The messageId of the email.</returns>
    public async Task<string> SendTemplateEmailAsync(string sender, List<string>
recipients,
        string templateName, object templateDataObject)
    {
        var messageId = "";
        try
        {
            // Template data should be serialized JSON from either a class or a
dynamic object.
            var templateData = JsonSerializer.Serialize(templateDataObject);

            var response = await
_amazonSimpleEmailService.SendTemplatedEmailAsync(
                new SendTemplatedEmailRequest
                {
                    Source = sender,
                    Destination = new Destination
                    {
                        ToAddresses = recipients
                    },
                    Template = templateName,
                    TemplateData = templateData
                });
            messageId = response.MessageId;
        }
        catch (Exception ex)
        {
            Console.WriteLine("SendTemplateEmailAsync failed with exception: " +
ex.Message);
        }

        return messageId;
    }
```

- For API details, see SendTemplatedEmail in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Send a templated email to a list of recipients.
/*!
  \param recipients; Vector of recipient email addresses.
  \param templateName: The name of the template to use.
  \param templateData: Map of key-value pairs for replacing text in template.
  \param senderEmailAddress: Email address of sender. Ignored if empty string.
  \param ccAddresses: Vector of cc addresses. Ignored if empty.
  \param replyToAddress: Reply to email address. Ignored if empty string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::sendTemplatedEmail(const Aws::Vector<Aws::String> &recipients,
                                     const Aws::String &templateName,
                                     const Aws::Map<Aws::String, Aws::String>
 &templateData,
                                     const Aws::String &senderEmailAddress,
                                     const Aws::Vector<Aws::String> &ccAddresses,
                                     const Aws::String &replyToAddress,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::Destination destination;
    if (!ccAddresses.empty()) {
        destination.WithCcAddresses(ccAddresses);
    }
    if (!recipients.empty()) {
        destination.WithToAddresses(recipients);
    }

    Aws::SES::Model::SendTemplatedEmailRequest sendTemplatedEmailRequest;
    sendTemplatedEmailRequest.SetDestination(destination);
```

```
    sendTemplatedEmailRequest.SetTemplate(templateName);

    std::ostringstream templateDataStream;
    templateDataStream << "{";
    size_t dataCount = 0;
    for (auto &pair: templateData) {
        templateDataStream << "\"" << pair.first << "\":\"" << pair.second <<
 "\"";
        dataCount++;
        if (dataCount < templateData.size()) {
            templateDataStream << ",";
        }
    }
    templateDataStream << "}";

    sendTemplatedEmailRequest.SetTemplateData(templateDataStream.str());

    if (!senderEmailAddress.empty()) {
        sendTemplatedEmailRequest.SetSource(senderEmailAddress);
    }
    if (!replyToAddress.empty()) {
        sendTemplatedEmailRequest.AddReplyToAddresses(replyToAddress);
    }

    auto outcome = sesClient.SendTemplatedEmail(sendTemplatedEmailRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully sent templated message with ID "
                  << outcome.GetResult().GetMessageId()
                  << "." << std::endl;
    }
    else {
        std::cerr << "Error sending templated message. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see SendTemplatedEmail in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sesv2.model.Destination;
import software.amazon.awssdk.services.sesv2.model.EmailContent;
import software.amazon.awssdk.services.sesv2.model.SendEmailRequest;
import software.amazon.awssdk.services.sesv2.model.SesV2Exception;
import software.amazon.awssdk.services.sesv2.SesV2Client;
import software.amazon.awssdk.services.sesv2.model.Template;

/**
 * Before running this AWS SDK for Java (v2) example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Also, make sure that you create a template. See the following documentation
 * topic:
 *
 * https://docs.aws.amazon.com/ses/latest/dg/send-personalized-email-api.html
 */

public class SendEmailTemplate {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <template> <sender> <recipient>\s

                Where:
                    template - The name of the email template.
```

```
                    sender - An email address that represents the sender.\s
                    recipient - An email address that represents the recipient.\s
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String templateName = args[0];
        String sender = args[1];
        String recipient = args[2];
        Region region = Region.US_EAST_1;
        SesV2Client sesv2Client = SesV2Client.builder()
                .region(region)
                .build();

        send(sesv2Client, sender, recipient, templateName);
    }

    public static void send(SesV2Client client, String sender, String recipient,
String templateName) {
        Destination destination = Destination.builder()
                .toAddresses(recipient)
                .build();

        /*
         * Specify both name and favorite animal (favoriteanimal) in your code
when
         * defining the Template object.
         * If you don't specify all the variables in the template, Amazon SES
doesn't
         * send the email.
         */
        Template myTemplate = Template.builder()
                .templateName(templateName)
                .templateData("{\n" +
                        "  \"name\": \"Jason\"\n," +
                        "  \"favoriteanimal\": \"Cat\"\n" +
                        "}")
                .build();

        EmailContent emailContent = EmailContent.builder()
                .template(myTemplate)
```

```
                .build();

        SendEmailRequest emailRequest = SendEmailRequest.builder()
                .destination(destination)
                .content(emailContent)
                .fromEmailAddress(sender)
                .build();

        try {
            System.out.println("Attempting to send an email based on a template
 using the AWS SDK for Java (v2)...");
            client.sendEmail(emailRequest);
            System.out.println("email based on a template was sent");

        } catch (SesV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see SendTemplatedEmail in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
```

```
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon
 SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
```

```
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- For API details, see SendTemplatedEmail in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesMailSender:
    """Encapsulates functions to send emails with Amazon SES."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def send_templated_email(
        self, source, destination, template_name, template_data, reply_tos=None
    ):
        """
        Sends an email based on a template. A template contains replaceable tags
        each enclosed in two curly braces, such as {{name}}. The template data
 passed
        in this function contains key-value pairs that define the values to
 insert
        in place of the template tags.

        Note: If your account is in the Amazon SES  sandbox, the source and
```

```
        destination email accounts must both be verified.

        :param source: The source email account.
        :param destination: The destination email account.
        :param template_name: The name of a previously created template.
        :param template_data: JSON-formatted key-value pairs of replacement
values
                              that are inserted in the template before it is
sent.
        :return: The ID of the message, assigned by Amazon SES.
        """
        send_args = {
            "Source": source,
            "Destination": destination.to_service_format(),
            "Template": template_name,
            "TemplateData": json.dumps(template_data),
        }
        if reply_tos is not None:
            send_args["ReplyToAddresses"] = reply_tos
        try:
            response = self.ses_client.send_templated_email(**send_args)
            message_id = response["MessageId"]
            logger.info(
                "Sent templated mail %s from %s to %s.",
                message_id,
                source,
                destination.tos,
            )
        except ClientError:
            logger.exception(
                "Couldn't send templated mail from %s to %s.", source,
destination.tos
            )
            raise
        else:
            return message_id
```

- For API details, see SendTemplatedEmail in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    " Send templated email
    DATA(lo_result) = lo_ses->sendtemplatedemail(
      iv_source = iv_source
      io_destination = io_destination
      iv_template = iv_template_name
      iv_templatedata = iv_template_data
      it_replytoaddresses = it_reply_tos
    ).
    ov_msg_id = lo_result->get_messageid( ).
    MESSAGE 'Templated email sent successfully' TYPE 'I'.
  CATCH /aws1/cx_sestmpldoesnotexistex INTO DATA(lo_ex1).
    DATA(lv_error) = |Template does not exist: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see SendTemplatedEmail in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateTemplate` with an AWS SDK

The following code examples show how to use `UpdateTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Verify an email identity and send messages

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Update an Amazon Simple Email Service (Amazon SES) template.
/*!
  \param templateName: The name of the template.
  \param htmlPart: The HTML body of the email.
  \param subjectPart: The subject line of the email.
  \param textPart: The plain text version of the email.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::updateTemplate(const Aws::String &templateName,
                                 const Aws::String &htmlPart,
                                 const Aws::String &subjectPart,
                                 const Aws::String &textPart,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::Template templateValues;

    templateValues.SetTemplateName(templateName);
    templateValues.SetSubjectPart(subjectPart);
    templateValues.SetHtmlPart(htmlPart);
    templateValues.SetTextPart(textPart);

    Aws::SES::Model::UpdateTemplateRequest updateTemplateRequest;
    updateTemplateRequest.SetTemplate(templateValues);
```

```
    Aws::SES::Model::UpdateTemplateOutcome outcome =
 sesClient.UpdateTemplate(updateTemplateRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated template." << std::endl;
    } else {
        std::cerr << "Error updating template. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see UpdateTemplate in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
```

```
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- For API details, see UpdateTemplate in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.
```

```
        :param subject: The subject of the email.
        :param text: The text version of the email.
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def update_template(self, name, subject, text, html):
        """
        Updates a previously created email template.

        :param name: The name of the template.
        :param subject: The subject of the email.
        :param text: The plain text version of the email.
        :param html: The HTML version of the email.
        """
        try:
            template = {
                "TemplateName": name,
                "SubjectPart": subject,
                "TextPart": text,
                "HtmlPart": html,
            }
            self.ses_client.update_template(Template=template)
            logger.info("Updated template %s.", name)
            self.template = template
            self._extract_tags(subject, text, html)
        except ClientError:
            logger.exception("Couldn't update template %s.", name)
            raise
```

- For API details, see UpdateTemplate in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
DATA(lo_template) = NEW /aws1/cl_sestemplate(
  iv_templatename = iv_name
  iv_subjectpart = iv_subject
  iv_textpart = iv_text
  iv_htmlpart = iv_html
).

TRY.
    lo_ses->updatetemplate( io_template = lo_template ).
    MESSAGE 'Template updated successfully' TYPE 'I'.
  CATCH /aws1/cx_sestmpldoesnotexistex INTO DATA(lo_ex1).
    DATA(lv_error) = |Template does not exist: { lo_ex1->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex1.
  CATCH /aws1/cx_sesinvalidtemplateex INTO DATA(lo_ex2).
    lv_error = |Invalid template: { lo_ex2->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex2.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex_generic).
    lv_error = |An error occurred: { lo_ex_generic->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex_generic.
ENDTRY.
```

- For API details, see UpdateTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `VerifyDomainIdentity` with an AWS SDK or CLI

The following code examples show how to use `VerifyDomainIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Copy email and domain identities across Regions
- Setting up email sending services
- Verify an email identity and send messages

CLI

AWS CLI

To verify a domain with Amazon SES

The following example uses the `verify-domain-identity` command to verify a domain:

```
aws ses verify-domain-identity --domain example.com
```

Output:

```
{
   "VerificationToken": "eoEmxw+YaYhb3h3iVJHuXMJXqeu1q1/wwmvjuEXAMPLE"
}
```

To complete domain verification, you must add a TXT record with the returned verification token to your domain's DNS settings. For more information, see Verifying Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

- For API details, see VerifyDomainIdentity in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- For API details, see VerifyDomainIdentity in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def verify_domain_identity(self, domain_name):
        """
        Starts verification of a domain identity. To complete verification, you
 must
        create a TXT record with a specific format through your DNS provider.

        For more information, see *Verifying a domain with Amazon SES* in the
        Amazon SES documentation:
            https://docs.aws.amazon.com/ses/latest/DeveloperGuide/verify-domain-
procedure.html

        :param domain_name: The name of the domain to verify.
        :return: The token to include in the TXT record with your DNS provider.
        """
        try:
            response = self.ses_client.verify_domain_identity(Domain=domain_name)
            token = response["VerificationToken"]
            logger.info("Got domain verification token for %s.", domain_name)
        except ClientError:
            logger.exception("Couldn't verify domain %s.", domain_name)
            raise
        else:
```

```
            return token
```

- For API details, see VerifyDomainIdentity in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_ses->verifydomainidentity( iv_domain =
iv_domain_name ).
        ov_token = lo_result->get_verificationtoken( ).
        MESSAGE 'Domain verification initiated' TYPE 'I'.
      CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
        DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
        MESSAGE lv_error TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- For API details, see VerifyDomainIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `VerifyEmailIdentity` with an AWS SDK or CLI

The following code examples show how to use `VerifyEmailIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Copy email and domain identities across Regions
- Setting up email sending services
- Verify an email identity and send messages

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Starts verification of an email identity. This request sends an email
   /// from Amazon SES to the specified email address. To complete
   /// verification, follow the instructions in the email.
   /// </summary>
   /// <param name="recipientEmailAddress">Email address to verify.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> VerifyEmailIdentityAsync(string
recipientEmailAddress)
   {
       var success = false;
       try
       {
           var response = await
_amazonSimpleEmailService.VerifyEmailIdentityAsync(
               new VerifyEmailIdentityRequest
               {
                   EmailAddress = recipientEmailAddress
               });

           success = response.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (Exception ex)
       {
           Console.WriteLine("VerifyEmailIdentityAsync failed with exception: "
+ ex.Message);
```

```
        }

        return success;
    }
```

- For API details, see VerifyEmailIdentity in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Add an email address to the list of identities associated with this account
 and
//! initiate verification.
/*!
  \param emailAddress; The email address to add.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SES::verifyEmailIdentity(const Aws::String &emailAddress,
                          const Aws::Client::ClientConfiguration
 &clientConfiguration)
{
    Aws::SES::SESClient sesClient(clientConfiguration);

    Aws::SES::Model::VerifyEmailIdentityRequest verifyEmailIdentityRequest;

    verifyEmailIdentityRequest.SetEmailAddress(emailAddress);

    Aws::SES::Model::VerifyEmailIdentityOutcome outcome =
 sesClient.VerifyEmailIdentity(verifyEmailIdentityRequest);

    if (outcome.IsSuccess())
    {
```

```
        std::cout << "Email verification initiated." << std::endl;
    }

    else
    {
        std::cerr << "Error initiating email verification. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see VerifyEmailIdentity in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To verify an email address with Amazon SES

The following example uses the `verify-email-identity` command to verify an email address:

```
aws ses verify-email-identity --email-address user@example.com
```

Before you can send an email using Amazon SES, you must verify the address or domain that you are sending the email from to prove that you own it. If you do not have production access yet, you also need to verify any email addresses that you send emails to except for email addresses provided by the Amazon SES mailbox simulator.

After verify-email-identity is called, the email address will receive a verification email. The user must click on the link in the email to complete the verification process.

For more information, see Verifying Email Addresses in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

- For API details, see VerifyEmailIdentity in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- For API details, see VerifyEmailIdentity in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def verify_email_identity(self, email_address):
        """
        Starts verification of an email identity. This function causes an email
        to be sent to the specified email address from Amazon SES. To complete
        verification, follow the instructions in the email.

        :param email_address: The email address to verify.
        """
        try:
            self.ses_client.verify_email_identity(EmailAddress=email_address)
            logger.info("Started verification of %s.", email_address)
        except ClientError:
            logger.exception("Couldn't start verification of %s.", email_address)
            raise
```

- For API details, see VerifyEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-ses' # v2: require 'aws-sdk'

# Replace recipient@example.com with a "To" address.
recipient = 'recipient@example.com'

# Create a new SES resource in the us-west-2 region.
# Replace us-west-2 with the AWS Region you're using for Amazon SES.
ses = Aws::SES::Client.new(region: 'us-west-2')

# Try to verify email address.
begin
  ses.verify_email_identity({
                              email_address: recipient
                            })

  puts "Email sent to #{recipient}"

# If something goes wrong, display an error message.
rescue Aws::SES::Errors::ServiceError => e
  puts "Email not sent. Error message: #{e}"
end
```

- For API details, see VerifyEmailIdentity in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_ses->verifyemailidentity( iv_emailaddress = iv_email_address ).
    MESSAGE 'Email verification initiated' TYPE 'I'.
  CATCH /aws1/cx_rt_generic INTO DATA(lo_ex).
    DATA(lv_error) = |An error occurred: { lo_ex->get_text( ) }|.
    MESSAGE lv_error TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- For API details, see VerifyEmailIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon SES using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon SES with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon SES or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Build an Amazon Transcribe streaming app

- Copy Amazon SES email and domain identities from one AWS Region to another using an AWS SDK
- Create a web application to track DynamoDB data
- Create an Amazon Redshift item tracker
- Create an Aurora Serverless work item tracker
- Detect PPE in images with Amazon Rekognition using an AWS SDK
- Detect objects in images with Amazon Rekognition using an AWS SDK
- Detect people and objects in a video with Amazon Rekognition using an AWS SDK
- Generate credentials to connect to an Amazon SES SMTP endpoint
- Setting up email sending services
- Use Step Functions to invoke Lambda functions
- Verify an email identity and send messages with Amazon SES using an AWS SDK

Build an Amazon Transcribe streaming app

The following code example shows how to build an app that records, transcribes, and translates live audio in real-time, and emails the results.

JavaScript

SDK for JavaScript (v3)

Shows how to use Amazon Transcribe to build an app that records, transcribes, and translates live audio in real-time, and emails the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Copy Amazon SES email and domain identities from one AWS Region to another using an AWS SDK

The following code example shows how to copy Amazon SES email and domain identities from one AWS Region to another. When domain identities are managed by Route 53, verification records are copied to the domain for the destination Region.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import argparse
import json
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


def get_identities(ses_client):
    """
    Gets the identities for the current Region. The Region is specified in the
    Boto3 Amazon SES client object.

    :param ses_client: A Boto3 Amazon SES client.
    :return: The list of email identities and the list of domain identities.
    """
    email_identities = []
    domain_identities = []
```

```
    try:
        identity_paginator = ses_client.get_paginator("list_identities")
        identity_iterator = identity_paginator.paginate(
            PaginationConfig={"PageSize": 20}
        )
        for identity_page in identity_iterator:
            for identity in identity_page["Identities"]:
                if "@" in identity:
                    email_identities.append(identity)
                else:
                    domain_identities.append(identity)
        logger.info(
            "Found %s email and %s domain identities.",
            len(email_identities),
            len(domain_identities),
        )
    except ClientError:
        logger.exception("Couldn't get identities.")
        raise
    else:
        return email_identities, domain_identities


def verify_emails(email_list, ses_client):
    """
    Starts verification of a list of email addresses. Verification causes an
 email
    to be sent to each address. To complete verification, the recipient must
 follow
    the instructions in the email.

    :param email_list: The list of email addresses to verify.
    :param ses_client: A Boto3 Amazon SES client.
    :return: The list of emails that were successfully submitted for
 verification.
    """
    verified_emails = []
    for email in email_list:
        try:
            ses_client.verify_email_identity(EmailAddress=email)
            verified_emails.append(email)
            logger.info("Started verification of %s.", email)
        except ClientError:
            logger.warning("Couldn't start verification of %s.", email)
```

```
    return verified_emails


def verify_domains(domain_list, ses_client):
    """
    Starts verification for a list of domain identities. This returns a token for
    each domain, which must be registered as a TXT record with the DNS provider
 for
    the domain.

    :param domain_list: The list of domains to verify.
    :param ses_client: A Boto3 Amazon SES client.
    :return: The generated domain tokens to use to completed verification.
    """
    domain_tokens = {}
    for domain in domain_list:
        try:
            response = ses_client.verify_domain_identity(Domain=domain)
            token = response["VerificationToken"]
            domain_tokens[domain] = token
            logger.info("Got verification token %s for domain %s.", token,
 domain)
        except ClientError:
            logger.warning("Couldn't get verification token for domain %s.",
 domain)
    return domain_tokens


def get_hosted_zones(route53_client):
    """
    Gets the Amazon Route 53 hosted zones for the current account.

    :param route53_client: A Boto3 Route 53 client.
    :return: The list of hosted zones.
    """
    zones = []
    try:
        zone_paginator = route53_client.get_paginator("list_hosted_zones")
        zone_iterator = zone_paginator.paginate(PaginationConfig={"PageSize":
 20})
        zones = [
            zone for zone_page in zone_iterator for zone in
 zone_page["HostedZones"]
        ]
```

```
        logger.info("Found %s hosted zones.", len(zones))
    except ClientError:
        logger.warning("Couldn't get hosted zones.")
    return zones


def find_domain_zone_matches(domains, zones):
    """
    Finds matches between Amazon SES verified domains and Route 53 hosted zones.
    Subdomain matches are taken when found, otherwise root domain matches are
 taken.

    :param domains: The list of domains to match.
    :param zones: The list of hosted zones to match.
    :return: The set of matched domain-zone pairs. When a match is not found, the
             domain is included in the set with a zone value of None.
    """
    domain_zones = {}
    for domain in domains:
        domain_zones[domain] = None
        # Start at the most specific sub-domain and walk up to the root domain
 until a
        # zone match is found.
        domain_split = domain.split(".")
        for index in range(0, len(domain_split) - 1):
            sub_domain = ".".join(domain_split[index:])
            for zone in zones:
                # Normalize the zone name from Route 53 by removing the trailing
 '.'.
                zone_name = zone["Name"][:-1]
                if sub_domain == zone_name:
                    domain_zones[domain] = zone
                    break
            if domain_zones[domain] is not None:
                break
    return domain_zones


def add_route53_verification_record(domain, token, zone, route53_client):
    """
    Adds a domain verification TXT record to the specified Route 53 hosted zone.
    When a TXT record already exists in the hosted zone for the specified domain,
    the existing values are preserved and the new token is added to the list.
```

```
    :param domain: The domain to add.
    :param token: The verification token for the domain.
    :param zone: The hosted zone where the domain verification record is added.
    :param route53_client: A Boto3 Route 53 client.
    """
    domain_token_record_set_name = f"_amazonses.{domain}"
    record_set_paginator =
route53_client.get_paginator("list_resource_record_sets")
    record_set_iterator = record_set_paginator.paginate(
        HostedZoneId=zone["Id"], PaginationConfig={"PageSize": 20}
    )
    records = []
    for record_set_page in record_set_iterator:
        try:
            txt_record_set = next(
                record_set
                for record_set in record_set_page["ResourceRecordSets"]
                if record_set["Name"][:-1] == domain_token_record_set_name
                and record_set["Type"] == "TXT"
            )
            records = txt_record_set["ResourceRecords"]
            logger.info(
                "Existing TXT record found in set %s for zone %s.",
                domain_token_record_set_name,
                zone["Name"],
            )
            break
        except StopIteration:
            pass
    records.append({"Value": json.dumps(token)})
    changes = [
        {
            "Action": "UPSERT",
            "ResourceRecordSet": {
                "Name": domain_token_record_set_name,
                "Type": "TXT",
                "TTL": 1800,
                "ResourceRecords": records,
            },
        }
    ]
    try:
        route53_client.change_resource_record_sets(
            HostedZoneId=zone["Id"], ChangeBatch={"Changes": changes}
```

```
        )
        logger.info(
            "Created or updated the TXT record in set %s for zone %s.",
            domain_token_record_set_name,
            zone["Name"],
        )
    except ClientError as err:
        logger.warning(
            "Got error %s. Couldn't create or update the TXT record for zone
 %s.",
            err.response["Error"]["Code"],
            zone["Name"],
        )


def generate_dkim_tokens(domain, ses_client):
    """
    Generates DKIM tokens for a domain. These must be added as CNAME records to
 the
    DNS provider for the domain.

    :param domain: The domain to generate tokens for.
    :param ses_client: A Boto3 Amazon SES client.
    :return: The list of generated DKIM tokens.
    """
    dkim_tokens = []
    try:
        dkim_tokens = ses_client.verify_domain_dkim(Domain=domain)["DkimTokens"]
        logger.info("Generated %s DKIM tokens for domain %s.", len(dkim_tokens),
 domain)
    except ClientError:
        logger.warning("Couldn't generate DKIM tokens for domain %s.", domain)
    return dkim_tokens


def add_dkim_domain_tokens(hosted_zone, domain, tokens, route53_client):
    """
    Adds DKIM domain token CNAME records to a Route 53 hosted zone.

    :param hosted_zone: The hosted zone where the records are added.
    :param domain: The domain to add.
    :param tokens: The DKIM tokens for the domain to add.
    :param route53_client: A Boto3 Route 53 client.
    """
```

```
    try:
        changes = [
            {
                "Action": "UPSERT",
                "ResourceRecordSet": {
                    "Name": f"{token}._domainkey.{domain}",
                    "Type": "CNAME",
                    "TTL": 1800,
                    "ResourceRecords": [{"Value":
 f"{token}.dkim.amazonses.com"}],
                },
            }
            for token in tokens
        ]
        route53_client.change_resource_record_sets(
            HostedZoneId=hosted_zone["Id"], ChangeBatch={"Changes": changes}
        )
        logger.info(
            "Added %s DKIM CNAME records to %s in zone %s.",
            len(tokens),
            domain,
            hosted_zone["Name"],
        )
    except ClientError:
        logger.warning(
            "Couldn't add DKIM CNAME records for %s to zone %s.",
            domain,
            hosted_zone["Name"],
        )


def configure_sns_topics(identity, topics, ses_client):
    """
    Configures Amazon Simple Notification Service (Amazon SNS) notifications for
    an identity. The Amazon SNS topics must already exist.

    :param identity: The identity to configure.
    :param topics: The list of topics to configure. The choices are Bounce,
 Delivery,
                   or Complaint.
    :param ses_client: A Boto3 Amazon SES client.
    """
    for topic in topics:
        topic_arn = input(
```

```
            f"Enter the Amazon Resource Name (ARN) of the {topic} topic or press
 "
            f"Enter to skip: "
        )
        if topic_arn != "":
            try:
                ses_client.set_identity_notification_topic(
                    Identity=identity, NotificationType=topic, SnsTopic=topic_arn
                )
                logger.info("Configured %s for %s notifications.", identity,
 topic)
            except ClientError:
                logger.warning(
                    "Couldn't configure %s for %s notifications.", identity,
 topic
                )


def replicate(source_client, destination_client, route53_client):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print(
        f"Replicating Amazon SES identities and other configuration from "
        f"{source_client.meta.region_name} to
 {destination_client.meta.region_name}."
    )
    print("-" * 88)

    print(f"Retrieving identities from {source_client.meta.region_name}.")
    source_emails, source_domains = get_identities(source_client)
    print("Email addresses found:")
    print(*source_emails)
    print("Domains found:")
    print(*source_domains)

    print("Starting verification for email identities.")
    dest_emails = verify_emails(source_emails, destination_client)
    print("Getting domain tokens for domain identities.")
    dest_domain_tokens = verify_domains(source_domains, destination_client)

    # Get Route 53 hosted zones and match them with Amazon SES domains.
    answer = input(
```

```
        "Is the DNS configuration for your domains managed by Amazon Route 53 (y/
n)? "
    )
    use_route53 = answer.lower() == "y"
    hosted_zones = get_hosted_zones(route53_client) if use_route53 else []
    if use_route53:
        print("Adding or updating Route 53 TXT records for your domains.")
        domain_zones = find_domain_zone_matches(dest_domain_tokens.keys(),
 hosted_zones)
        for domain in domain_zones:
            add_route53_verification_record(
                domain, dest_domain_tokens[domain], domain_zones[domain],
 route53_client
            )
    else:
        print(
            "Use these verification tokens to create TXT records through your DNS
 "
            "provider:"
        )
        pprint(dest_domain_tokens)

    answer = input("Do you want to configure DKIM signing for your identities (y/
n)? ")
    if answer.lower() == "y":
        # Build a set of unique domains from email and domain identities.
        domains = {email.split("@")[1] for email in dest_emails}
        domains.update(dest_domain_tokens)
        domain_zones = find_domain_zone_matches(domains, hosted_zones)
        for domain, zone in domain_zones.items():
            answer = input(
                f"Do you want to configure DKIM signing for {domain} (y/n)? "
            )
            if answer.lower() == "y":
                dkim_tokens = generate_dkim_tokens(domain, destination_client)
                if use_route53 and zone is not None:
                    add_dkim_domain_tokens(zone, domain, dkim_tokens,
 route53_client)
                else:
                    print(
                        "Add the following DKIM tokens as CNAME records through
 your "
                        "DNS provider:"
                    )
```

```
                    print(*dkim_tokens, sep="\n")

    answer = input(
        "Do you want to configure Amazon SNS notifications for your identities
 (y/n)? "
    )
    if answer.lower() == "y":
        for identity in dest_emails + list(dest_domain_tokens.keys()):
            answer = input(
                f"Do you want to configure Amazon SNS topics for {identity} (y/
n)? "
            )
            if answer.lower() == "y":
                configure_sns_topics(
                    identity, ["Bounce", "Delivery", "Complaint"],
 destination_client
                )

    print(f"Replication complete for {destination_client.meta.region_name}.")
    print("-" * 88)


def main():
    boto3_session = boto3.Session()
    ses_regions = boto3_session.get_available_regions("ses")
    parser = argparse.ArgumentParser(
        description="Copies email address and domain identities from one AWS
 Region to "
        "another. Optionally adds records for domain verification and DKIM "
        "signing to domains that are managed by Amazon Route 53, "
        "and sets up Amazon SNS notifications for events of interest."
    )
    parser.add_argument(
        "source_region", choices=ses_regions, help="The region to copy from."
    )
    parser.add_argument(
        "destination_region", choices=ses_regions, help="The region to copy to."
    )
    args = parser.parse_args()
    source_client = boto3.client("ses", region_name=args.source_region)
    destination_client = boto3.client("ses", region_name=args.destination_region)
    route53_client = boto3.client("route53")
    replicate(source_client, destination_client, route53_client)
```

```
if __name__ == "__main__":
    main()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - ListIdentities
  - SetIdentityNotificationTopic
  - VerifyDomainDkim
  - VerifyDomainIdentity
  - VerifyEmailIdentity

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a web application to track DynamoDB data

The following code examples show how to create a web application that tracks work items in an Amazon DynamoDB table and uses Amazon Simple Email Service (Amazon SES) to send reports.

.NET

SDK for .NET

Shows how to use the Amazon DynamoDB .NET API to create a dynamic web application that tracks DynamoDB work data.

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

Services used in this example

- DynamoDB
- Amazon SES

Java

SDK for Java 2.x

Shows how to use the Amazon DynamoDB API to create a dynamic web application that tracks DynamoDB work data.

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

Services used in this example

- DynamoDB
- Amazon SES

Kotlin

SDK for Kotlin

Shows how to use the Amazon DynamoDB API to create a dynamic web application that tracks DynamoDB work data.

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

Services used in this example

- DynamoDB
- Amazon SES

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) to create a REST service that tracks work items in Amazon DynamoDB and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses the Flask web framework to handle HTTP routing and integrates with a React webpage to present a fully functional web application.

- Build a Flask REST service that integrates with AWS services.
- Read, write, and update work items that are stored in a DynamoDB table.

- Use Amazon SES to send email reports of work items.

For complete source code and instructions on how to set up and run, see the full example in the AWS Code Examples Repository on GitHub.

Services used in this example

- DynamoDB
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon Redshift item tracker

The following code examples show how to create a web application that tracks and reports on work items using an Amazon Redshift database.

Java

SDK for Java 2.x

Shows how to create a web application that tracks and reports on work items stored in an Amazon Redshift database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Redshift data and for use by a React application, see the full example on GitHub.

Services used in this example

- Amazon Redshift
- Amazon SES

Kotlin

SDK for Kotlin

Shows how to create a web application that tracks and reports on work items stored in an Amazon Redshift database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Redshift data and for use by a React application, see the full example on GitHub.

Services used in this example

- Amazon Redshift
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create an Aurora Serverless work item tracker

The following code examples show how to create a web application that tracks work items in an Amazon Aurora Serverless database and uses Amazon Simple Email Service (Amazon SES) to send reports.

.NET

SDK for .NET

Shows how to use the AWS SDK for .NET to create a web application that tracks work items in an Amazon Aurora database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with a RESTful .NET backend.

- Integrate a React web application with AWS services.
- List, add, update, and delete items in an Aurora table.
- Send an email report of filtered work items using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

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

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service

- Amazon SES

C++

SDK for C++

Shows how to create a web application that tracks and reports on work items stored in an Amazon Aurora Serverless database.

For complete source code and instructions on how to set up a C++ REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Java

SDK for Java 2.x

Shows how to create a web application that tracks and reports on work items stored in an Amazon RDS database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

For complete source code and instructions on how to set up and run an example that uses the JDBC API, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service

- Amazon SES

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript (v3) to create a web application that tracks work items in an Amazon Aurora database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with an Express Node.js backend.

- Integrate a React.js web application with AWS services.
- List, add, and update items in an Aurora table.
- Send an email report of filtered work items by using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

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

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Kotlin

SDK for Kotlin

Shows how to create a web application that tracks and reports on work items stored in an Amazon RDS database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

Services used in this example

- Aurora

- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

PHP

SDK for PHP

Shows how to use the AWS SDK for PHP to create a web application that tracks work items in an Amazon RDS database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with a RESTful PHP backend.

- Integrate a React.js web application with AWS services.
- List, add, update, and delete items in an Amazon RDS table.
- Send an email report of filtered work items using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

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

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) to create a REST service that tracks work items in an Amazon Aurora Serverless database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses the Flask web framework to handle HTTP routing and integrates with a React webpage to present a fully functional web application.

- Build a Flask REST service that integrates with AWS services.

- Read, write, and update work items that are stored in an Aurora Serverless database.
- Create an AWS Secrets Manager secret that contains database credentials and use it to authenticate calls to the database.
- Use Amazon SES to send email reports of work items.

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

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Detect PPE in images with Amazon Rekognition using an AWS SDK

The following code example shows how to build an app that uses Amazon Rekognition to detect Personal Protective Equipment (PPE) in images.

Java

SDK for Java 2.x

Shows how to create an AWS Lambda function that detects images with Personal Protective Equipment.

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

Services used in this example

- DynamoDB
- Amazon Rekognition

- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Detect objects in images with Amazon Rekognition using an AWS SDK

The following code examples show how to build an app that uses Amazon Rekognition to detect objects by category in images.

.NET

SDK for .NET

Shows how to use Amazon Rekognition .NET API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript (v3)

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for objects using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Send an email notification using Amazon SES.

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

SDK for Kotlin

Shows how to use Amazon Rekognition Kotlin API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage

Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python (Boto3)

Shows you how to use the AWS SDK for Python (Boto3) to create a web application that lets you do the following:

- Upload photos to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Rekognition to analyze and label the photos.
- Use Amazon Simple Email Service (Amazon SES) to send email reports of image analysis.

This example contains two main components: a webpage written in JavaScript that is built with React, and a REST service written in Python that is built with Flask-RESTful.

You can use the React webpage to:

- Display a list of images that are stored in your S3 bucket.
- Upload images from your computer to your S3 bucket.
- Display images and labels that identify items that are detected in the image.
- Get a report of all images in your S3 bucket and send an email of the report.

The webpage calls the REST service. The service sends requests to AWS to perform the following actions:

- Get and filter the list of images in your S3 bucket.
- Upload photos to your S3 bucket.
- Use Amazon Rekognition to analyze individual photos and get a list of labels that identify items that are detected in the photo.

- Analyze all photos in your S3 bucket and use Amazon SES to email a report.

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Detect people and objects in a video with Amazon Rekognition using an AWS SDK

The following code examples show how to detect people and objects in a video with Amazon Rekognition.

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python (Boto3)

Use Amazon Rekognition to detect faces, objects, and people in videos by starting asynchronous detection jobs. This example also configures Amazon Rekognition to notify an Amazon Simple Notification Service (Amazon SNS) topic when jobs complete and subscribes an Amazon Simple Queue Service (Amazon SQS) queue to the topic. When the queue receives a message about a job, the job is retrieved and the results are output.

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

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Generate credentials to connect to an Amazon SES SMTP endpoint

The following code example shows how to generate credentials to connect to an Amazon SES SMTP endpoint.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
#!/usr/bin/env python3

import hmac
import hashlib
import base64
import argparse

SMTP_REGIONS = [
    "us-east-2",  # US East (Ohio)
    "us-east-1",  # US East (N. Virginia)
    "us-west-2",  # US West (Oregon)
    "ap-south-1",  # Asia Pacific (Mumbai)
    "ap-northeast-2",  # Asia Pacific (Seoul)
    "ap-southeast-1",  # Asia Pacific (Singapore)
    "ap-southeast-2",  # Asia Pacific (Sydney)
    "ap-northeast-1",  # Asia Pacific (Tokyo)
    "ca-central-1",  # Canada (Central)
    "eu-central-1",  # Europe (Frankfurt)
    "eu-west-1",  # Europe (Ireland)
    "eu-west-2",  # Europe (London)
    "eu-south-1",  # Europe (Milan)
    "eu-north-1",  # Europe (Stockholm)
    "sa-east-1",  # South America (Sao Paulo)
    "us-gov-west-1",  # AWS GovCloud (US)
    "us-gov-east-1",  # AWS GovCloud (US)
]

# These values are required to calculate the signature. Do not change them.
DATE = "11111111"
SERVICE = "ses"
MESSAGE = "SendRawEmail"
TERMINAL = "aws4_request"
VERSION = 0x04


def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()


def calculate_key(secret_access_key, region):
    if region not in SMTP_REGIONS:
        raise ValueError(f"The {region} Region doesn't have an SMTP endpoint.")
```

```
    signature = sign(("AWS4" + secret_access_key).encode("utf-8"), DATE)
    signature = sign(signature, region)
    signature = sign(signature, SERVICE)
    signature = sign(signature, TERMINAL)
    signature = sign(signature, MESSAGE)
    signature_and_version = bytes([VERSION]) + signature
    smtp_password = base64.b64encode(signature_and_version)
    return smtp_password.decode("utf-8")


def main():
    parser = argparse.ArgumentParser(
        description="Convert a Secret Access Key to an SMTP password."
    )
    parser.add_argument("secret", help="The Secret Access Key to convert.")
    parser.add_argument(
        "region",
        help="The AWS Region where the SMTP password will be used.",
        choices=SMTP_REGIONS,
    )
    args = parser.parse_args()
    print(calculate_key(args.secret, args.region))


if __name__ == "__main__":
    main()
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Setting up email sending services

The following code example shows how to:

- Verify an email address
- Verify a domain (optional)
- Check your sending limits
- Send a test email

- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon SES Setup Script (v2)
# This script helps you set up Amazon SES for sending emails

# Initialize log file
LOG_FILE="ses-setup.log"
echo "Starting Amazon SES setup at $(date)" > "$LOG_FILE"

# Function to log commands and their output
log_cmd() {
    echo "Running: $1" | tee -a "$LOG_FILE"
    eval "$1" 2>&1 | tee -a "$LOG_FILE"
    return ${PIPESTATUS[0]}
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"
    local ignore_error="${4:-false}"

    if [[ $cmd_status -ne 0 || "$cmd_output" =~ [Ee][Rr][Rr][Oo][Rr] ]]; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        if [[ "$ignore_error" != "true" ]]; then
            cleanup_resources
            exit 1
        fi
    fi
```

```
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..." | tee -a "$LOG_FILE"

    # No physical resources to clean up for SES setup
    # Email identities can be deleted if needed
    if [[ -n "$EMAIL_ADDRESS" ]]; then
        echo "Deleting email identity: $EMAIL_ADDRESS" | tee -a "$LOG_FILE"
        log_cmd "aws ses delete-identity --identity \"$EMAIL_ADDRESS\""
    fi

    if [[ -n "$RECIPIENT_EMAIL" && "$RECIPIENT_EMAIL" != "$EMAIL_ADDRESS" ]];
 then
        echo "Deleting recipient email identity: $RECIPIENT_EMAIL" | tee -a
 "$LOG_FILE"
        log_cmd "aws ses delete-identity --identity \"$RECIPIENT_EMAIL\""
    fi

    if [[ -n "$DOMAIN_NAME" ]]; then
        echo "Deleting domain identity: $DOMAIN_NAME" | tee -a "$LOG_FILE"
        log_cmd "aws ses delete-identity --identity \"$DOMAIN_NAME\""
    fi
}

# Track created resources
CREATED_RESOURCES=()

# Welcome message
echo "============================================="
echo "Amazon SES Setup Script"
echo "============================================="
echo "This script will help you set up Amazon SES for sending emails."
echo "You'll need to verify at least one email address that you own."
echo ""
echo "NOTE: New SES accounts are placed in the sandbox environment."
echo "In the sandbox, both sender AND recipient email addresses must be
 verified."
echo ""

# Get email address to verify
echo "Please enter an email address that you own and can access:"
```

```
EMAIL_ADDRESS="user-$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 8 | head -n
 1)@example.com"
echo "$EMAIL_ADDRESS"

# Verify email identity
echo "Verifying email address: $EMAIL_ADDRESS" | tee -a "$LOG_FILE"
OUTPUT=$(log_cmd "aws ses verify-email-identity --email-address \"$EMAIL_ADDRESS
\"")
check_error "$OUTPUT" $? "Failed to verify email address"

CREATED_RESOURCES+=("Email identity: $EMAIL_ADDRESS")
echo "A verification email has been sent to $EMAIL_ADDRESS."
echo "Please check your inbox and click the verification link before continuing."
echo ""
echo "Press Enter after you've verified your email address..."
sleep 1

# Check verification status
echo "Checking verification status..." | tee -a "$LOG_FILE"
OUTPUT=$(log_cmd "aws ses list-identities --identity-type EmailAddress")
check_error "$OUTPUT" $? "Failed to list identities"

OUTPUT=$(log_cmd "aws ses get-identity-verification-attributes --identities
 \"$EMAIL_ADDRESS\"")
check_error "$OUTPUT" $? "Failed to get verification attributes"

# Check if the email is verified
VERIFICATION_STATUS=$(echo "$OUTPUT" | grep -o '"VerificationStatus": "[^"]*' |
 cut -d'"' -f4)
if [[ "$VERIFICATION_STATUS" != "Success" ]]; then
    echo "Email address $EMAIL_ADDRESS is not verified yet. Please check your
 inbox and verify before continuing."
    echo "Exiting script..."
    exit 1
fi

# Ask if user wants to verify a domain
echo ""
echo "Do you want to verify a domain for sending emails? (y/n):"
VERIFY_DOMAIN="y"
echo "$VERIFY_DOMAIN"

if [[ "$VERIFY_DOMAIN" =~ ^[Yy] ]]; then
    echo "Please enter the domain name you want to verify:"
```

```
    DOMAIN_NAME="test-$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 8 | head -n
1).example.com"
    echo "$DOMAIN_NAME"

    # Verify domain identity
    echo "Verifying domain: $DOMAIN_NAME" | tee -a "$LOG_FILE"
    OUTPUT=$(log_cmd "aws ses verify-domain-identity --domain \"$DOMAIN_NAME\"")
    check_error "$OUTPUT" $? "Failed to verify domain identity"

    # Extract verification token
    VERIFICATION_TOKEN=$(echo "$OUTPUT" | grep -o '"VerificationToken": "[^"]*' |
cut -d'"' -f4)

    CREATED_RESOURCES+=("Domain identity: $DOMAIN_NAME")

    echo ""
    echo "============================================="
    echo "Domain Verification Instructions"
    echo "============================================="
    echo "To verify your domain ownership, you need to add a TXT record"
    echo "to your domain's DNS settings with the following values:"
    echo ""
    echo "Record Type: TXT"
    echo "Record Name: _amazonses.$DOMAIN_NAME"
    echo "Record Value: $VERIFICATION_TOKEN"
    echo ""
    echo "After adding this DNS record, verification may take up to 72 hours."
    echo ""

    # Set up DKIM for the domain
    echo "Setting up DKIM for domain: $DOMAIN_NAME" | tee -a "$LOG_FILE"
    OUTPUT=$(log_cmd "aws ses verify-domain-dkim --domain \"$DOMAIN_NAME\"")
    check_error "$OUTPUT" $? "Failed to set up DKIM"

    # Extract DKIM tokens
    DKIM_TOKENS=$(echo "$OUTPUT" | grep -o '"DkimTokens": \[[^]]*\]' | sed
's/"DkimTokens": \[\|\]//g' | sed 's/,//g' | sed 's/"//g')

    echo "============================================="
    echo "DKIM Configuration Instructions"
    echo "============================================="
    echo "To configure DKIM for your domain, add the following CNAME records"
    echo "to your domain's DNS settings:"
    echo ""
```

```
    for token in $DKIM_TOKENS; do
        echo "Record Type: CNAME"
        echo "Record Name: ${token}._domainkey.$DOMAIN_NAME"
        echo "Record Value: ${token}.dkim.amazonses.com"
        echo ""
    done

    echo "After adding these DNS records, DKIM verification may take up to 72
 hours."
    echo ""
fi

# Check sending limits
echo "Checking your SES sending limits..." | tee -a "$LOG_FILE"
OUTPUT=$(log_cmd "aws ses get-send-quota")
check_error "$OUTPUT" $? "Failed to get sending quota"

# Extract quota information
MAX_SEND_RATE=$(echo "$OUTPUT" | grep -o '"MaxSendRate": [0-9.]*' | cut -d' ' -
f2)
MAX_24_HOUR_SEND=$(echo "$OUTPUT" | grep -o '"Max24HourSend": [0-9.]*' | cut -d'
 ' -f2)
SENT_LAST_24_HOURS=$(echo "$OUTPUT" | grep -o '"SentLast24Hours": [0-9.]*' | cut
 -d' ' -f2)

echo ""
echo "================================================"
echo "Your SES Sending Limits"
echo "================================================"
echo "Maximum send rate: $MAX_SEND_RATE emails/second"
echo "Maximum 24-hour send: $MAX_24_HOUR_SEND emails"
echo "Sent in the last 24 hours: $SENT_LAST_24_HOURS emails"
echo ""

# Ask if user wants to send a test email
echo "Do you want to send a test email? (y/n):"
SEND_TEST="y"
echo "$SEND_TEST"

if [[ "$SEND_TEST" =~ ^[Yy] ]]; then
    echo ""
    echo "================================================"
    echo "SANDBOX ENVIRONMENT NOTICE"
```

```
    echo "============================================="
    echo "Your account is likely in the SES sandbox environment."
    echo "In the sandbox, you can only send emails to verified email addresses."
    echo ""
    echo "Do you want to:"
    echo "1. Send a test email to the same verified address ($EMAIL_ADDRESS)"
    echo "2. Verify another email address to use as recipient"
    echo ""
    echo "Enter your choice (1 or 2):"
    RECIPIENT_CHOICE="1"
    echo "$RECIPIENT_CHOICE"

    if [[ "$RECIPIENT_CHOICE" == "1" ]]; then
        RECIPIENT_EMAIL="$EMAIL_ADDRESS"
    else
        echo "Please enter the recipient email address you want to verify:"
        RECIPIENT_EMAIL="user-$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 8 |
 head -n 1)@example.com"
        echo "$RECIPIENT_EMAIL"

        # Verify recipient email identity if different from sender
        if [[ "$RECIPIENT_EMAIL" != "$EMAIL_ADDRESS" ]]; then
            echo "Verifying recipient email address: $RECIPIENT_EMAIL" | tee -a
 "$LOG_FILE"
            OUTPUT=$(log_cmd "aws ses verify-email-identity --email-address
 \"$RECIPIENT_EMAIL\"")
            check_error "$OUTPUT" $? "Failed to verify recipient email address"

            CREATED_RESOURCES+=("Email identity: $RECIPIENT_EMAIL")
            echo "A verification email has been sent to $RECIPIENT_EMAIL."
            echo "Please check the inbox and click the verification link before
 continuing."
            echo ""
            echo "Press Enter after you've verified the recipient email
 address..."
            sleep 1

            # Check recipient verification status
            OUTPUT=$(log_cmd "aws ses get-identity-verification-attributes --
identities \"$RECIPIENT_EMAIL\"")
            check_error "$OUTPUT" $? "Failed to get recipient verification
 attributes"

            # Check if the recipient email is verified
```

```
            RECIPIENT_VERIFICATION_STATUS=$(echo "$OUTPUT" | grep -o
 '"VerificationStatus": "[^"]*' | cut -d'"' -f4)
            if [[ "$RECIPIENT_VERIFICATION_STATUS" != "Success" ]]; then
                echo "Recipient email address $RECIPIENT_EMAIL is not verified
 yet."
                echo "You can try sending the email anyway, but it may fail."
            fi
        fi
    fi

    echo "Sending test email from $EMAIL_ADDRESS to $RECIPIENT_EMAIL..." | tee -a
 "$LOG_FILE"
    OUTPUT=$(log_cmd "aws ses send-email \
        --from \"$EMAIL_ADDRESS\" \
        --destination \"ToAddresses=$RECIPIENT_EMAIL\" \
        --message \"Subject={Data=SES Test
 Email,Charset=UTF-8},Body={Text={Data=This is a test email sent from Amazon SES
 using the AWS CLI,Charset=UTF-8}}\"")

    # Don't exit on send email error, just report it
    check_error "$OUTPUT" $? "Failed to send test email" "true"

    # Check if the email was sent successfully
    if [[ "$OUTPUT" =~ "MessageId" ]]; then
        # Extract message ID
        MESSAGE_ID=$(echo "$OUTPUT" | grep -o '"MessageId": "[^"]*' | cut -d'"' -
f4)
        echo "Test email sent successfully! Message ID: $MESSAGE_ID"
    else
        echo "Failed to send test email. This is likely because your account is
 in the sandbox environment."
        echo "In the sandbox, both sender AND recipient email addresses must be
 verified."
    fi
    echo ""
fi

# Summary of created resources
echo ""
echo "============================================="
echo "Setup Complete - Resources Created"
echo "============================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
```

```
done
echo ""

# Ask if user wants to clean up resources
echo "============================================="
echo "CLEANUP CONFIRMATION"
echo "============================================="
echo "Do you want to clean up all created resources? (y/n):"
CLEANUP_CHOICE="y"
echo "$CLEANUP_CHOICE"

if [[ "$CLEANUP_CHOICE" =~ ^[Yy] ]]; then
    cleanup_resources
    echo "Cleanup completed."
else
    echo "Resources have been preserved."
fi

echo ""
echo "============================================="
echo "Amazon SES Setup Complete"
echo "============================================="
echo "For production use, you may need to request to be moved out of the SES
 sandbox."
echo "Visit the SES console and navigate to 'Account dashboard' to request
 production access."
echo ""
echo "Log file: $LOG_FILE"
echo "============================================="
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - DeleteIdentity
  - GetIdentityVerificationAttributes
  - GetSendQuota
  - ListIdentities
  - SendEmail
  - VerifyDomainDkim
  - VerifyDomainIdentity
  - VerifyEmailIdentity

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use Step Functions to invoke Lambda functions

The following code example shows how to create an AWS Step Functions state machine that invokes AWS Lambda functions in sequence.

Java

SDK for Java 2.x

Shows how to create an AWS serverless workflow by using AWS Step Functions and the AWS SDK for Java 2.x. Each workflow step is implemented using an AWS Lambda function.

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

Services used in this example

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Verify an email identity and send messages with Amazon SES using an AWS SDK

The following code example shows how to:

- Add and verify an email address with Amazon SES.
- Send a standard email message.
- Create a template and send a templated email message.
- Send a message by using an Amazon SES SMTP server.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Verify an email address with Amazon SES and send messages.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Simple Email Service (Amazon SES) email demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    ses_client = boto3.client("ses")
    ses_identity = SesIdentity(ses_client)
    ses_mail_sender = SesMailSender(ses_client)
    ses_template = SesTemplate(ses_client)
    email = input("Enter an email address to send mail with Amazon SES: ")
    status = ses_identity.get_identity_status(email)
    verified = status == "Success"
    if not verified:
        answer = input(
            f"The address '{email}' is not verified with Amazon SES. Unless your
 "
            f"Amazon SES account is out of sandbox, you can send mail only from "
            f"and to verified accounts. Do you want to verify this account for
 use "
            f"with Amazon SES? If yes, the address will receive a verification "
            f"email (y/n): "
        )
        if answer.lower() == "y":
            ses_identity.verify_email_identity(email)
            print(f"Follow the steps in the email to {email} to complete
 verification.")
            print("Waiting for verification...")
            try:
                ses_identity.wait_until_identity_exists(email)
```

```
                print(f"Identity verified for {email}.")
                verified = True
            except WaiterError:
                print(
                    f"Verification timeout exceeded. You must complete the "
                    f"steps in the email sent to {email} to verify the address."
                )

    if verified:
        test_message_text = "Hello from the Amazon SES mail demo!"
        test_message_html = "<p>Hello!</p><p>From the <b>Amazon SES</b> mail
demo!</p>"

        print(f"Sending mail from {email} to {email}.")
        ses_mail_sender.send_email(
            email,
            SesDestination([email]),
            "Amazon SES demo",
            test_message_text,
            test_message_html,
        )
        input("Mail sent. Check your inbox and press Enter to continue.")

        template = {
            "name": "doc-example-template",
            "subject": "Example of an email template.",
            "text": "This is what {{name}} will {{action}} if {{name}} can't
display "
            "HTML.",
            "html": "<p><i>This</i> is what {{name}} will {{action}} if {{name}}
"
            "<b>can</b> display HTML.</p>",
        }
        print("Creating a template and sending a templated email.")
        ses_template.create_template(**template)
        template_data = {"name": email.split("@")[0], "action": "read"}
        if ses_template.verify_tags(template_data):
            ses_mail_sender.send_templated_email(
                email, SesDestination([email]), ses_template.name(),
template_data
            )
            input("Mail sent. Check your inbox and press Enter to continue.")

        print("Sending mail through the Amazon SES SMTP server.")
```

```
        boto3_session = boto3.Session()
        region = boto3_session.region_name
        credentials = boto3_session.get_credentials()
        port = 587
        smtp_server = f"email-smtp.{region}.amazonaws.com"
        password = calculate_key(credentials.secret_key, region)
        message = """
Subject: Hi there

This message is sent from the Amazon SES SMTP mail demo."""
        context = ssl.create_default_context()
        with smtplib.SMTP(smtp_server, port) as server:
            server.starttls(context=context)
            server.login(credentials.access_key, password)
            server.sendmail(email, email, message)
        print("Mail sent. Check your inbox!")

    if ses_template.template is not None:
        print("Deleting demo template.")
        ses_template.delete_template()
    if verified:
        answer = input(f"Do you want to remove {email} from Amazon SES (y/n)? ")
        if answer.lower() == "y":
            ses_identity.delete_identity(email)
    print("Thanks for watching!")
    print("-" * 88)
```

Create functions to wrap Amazon SES identity actions.

```
class SesIdentity:
    """Encapsulates Amazon SES identity functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def verify_domain_identity(self, domain_name):
```

```
        """
        Starts verification of a domain identity. To complete verification, you
 must
        create a TXT record with a specific format through your DNS provider.

        For more information, see *Verifying a domain with Amazon SES* in the
        Amazon SES documentation:
            https://docs.aws.amazon.com/ses/latest/DeveloperGuide/verify-domain-
procedure.html

        :param domain_name: The name of the domain to verify.
        :return: The token to include in the TXT record with your DNS provider.
        """
        try:
            response = self.ses_client.verify_domain_identity(Domain=domain_name)
            token = response["VerificationToken"]
            logger.info("Got domain verification token for %s.", domain_name)
        except ClientError:
            logger.exception("Couldn't verify domain %s.", domain_name)
            raise
        else:
            return token


    def verify_email_identity(self, email_address):
        """
        Starts verification of an email identity. This function causes an email
        to be sent to the specified email address from Amazon SES. To complete
        verification, follow the instructions in the email.

        :param email_address: The email address to verify.
        """
        try:
            self.ses_client.verify_email_identity(EmailAddress=email_address)
            logger.info("Started verification of %s.", email_address)
        except ClientError:
            logger.exception("Couldn't start verification of %s.", email_address)
            raise


    def wait_until_identity_exists(self, identity):
        """
        Waits until an identity exists. The waiter polls Amazon SES until the
```

```
        identity has been successfully verified or until it exceeds its maximum
time.

        :param identity: The identity to wait for.
        """
        try:
            waiter = self.ses_client.get_waiter("identity_exists")
            logger.info("Waiting until %s exists.", identity)
            waiter.wait(Identities=[identity])
        except WaiterError:
            logger.error("Waiting for identity %s failed or timed out.",
identity)
            raise


    def get_identity_status(self, identity):
        """
        Gets the status of an identity. This can be used to discover whether
        an identity has been successfully verified.

        :param identity: The identity to query.
        :return: The status of the identity.
        """
        try:
            response = self.ses_client.get_identity_verification_attributes(
                Identities=[identity]
            )
            status = response["VerificationAttributes"].get(
                identity, {"VerificationStatus": "NotFound"}
            )["VerificationStatus"]
            logger.info("Got status of %s for %s.", status, identity)
        except ClientError:
            logger.exception("Couldn't get status for %s.", identity)
            raise
        else:
            return status


    def delete_identity(self, identity):
        """
        Deletes an identity.

        :param identity: The identity to remove.
        """
```

```
        try:
            self.ses_client.delete_identity(Identity=identity)
            logger.info("Deleted identity %s.", identity)
        except ClientError:
            logger.exception("Couldn't delete identity %s.", identity)
            raise


    def list_identities(self, identity_type, max_items):
        """
        Gets the identities of the specified type for the current account.

        :param identity_type: The type of identity to retrieve, such as
 EmailAddress.
        :param max_items: The maximum number of identities to retrieve.
        :return: The list of retrieved identities.
        """
        try:
            response = self.ses_client.list_identities(
                IdentityType=identity_type, MaxItems=max_items
            )
            identities = response["Identities"]
            logger.info("Got %s identities for the current account.",
 len(identities))
        except ClientError:
            logger.exception("Couldn't list identities for the current account.")
            raise
        else:
            return identities
```

Create functions to wrap Amazon SES template actions.

```
class SesTemplate:
    """Encapsulates Amazon SES template functions."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client
```

```
        self.template = None
        self.template_tags = set()

    def _extract_tags(self, subject, text, html):
        """
        Extracts tags from a template as a set of unique values.

        :param subject: The subject of the email.
        :param text: The text version of the email.
        :param html: The html version of the email.
        """
        self.template_tags = set(re.findall(TEMPLATE_REGEX, subject + text +
html))
        logger.info("Extracted template tags: %s", self.template_tags)


    def create_template(self, name, subject, text, html):
        """
        Creates an email template.

        :param name: The name of the template.
        :param subject: The subject of the email.
        :param text: The plain text version of the email.
        :param html: The HTML version of the email.
        """
        try:
            template = {
                "TemplateName": name,
                "SubjectPart": subject,
                "TextPart": text,
                "HtmlPart": html,
            }
            self.ses_client.create_template(Template=template)
            logger.info("Created template %s.", name)
            self.template = template
            self._extract_tags(subject, text, html)
        except ClientError:
            logger.exception("Couldn't create template %s.", name)
            raise


    def delete_template(self):
        """
        Deletes an email template.
```

```
        """
        try:

self.ses_client.delete_template(TemplateName=self.template["TemplateName"])
            logger.info("Deleted template %s.", self.template["TemplateName"])
            self.template = None
            self.template_tags = None
        except ClientError:
            logger.exception(
                "Couldn't delete template %s.", self.template["TemplateName"]
            )
            raise


    def get_template(self, name):
        """
        Gets a previously created email template.

        :param name: The name of the template to retrieve.
        :return: The retrieved email template.
        """
        try:
            response = self.ses_client.get_template(TemplateName=name)
            self.template = response["Template"]
            logger.info("Got template %s.", name)
            self._extract_tags(
                self.template["SubjectPart"],
                self.template["TextPart"],
                self.template["HtmlPart"],
            )
        except ClientError:
            logger.exception("Couldn't get template %s.", name)
            raise
        else:
            return self.template


    def list_templates(self):
        """
        Gets a list of all email templates for the current account.

        :return: The list of retrieved email templates.
        """
        try:
```

```
            response = self.ses_client.list_templates()
            templates = response["TemplatesMetadata"]
            logger.info("Got %s templates.", len(templates))
        except ClientError:
            logger.exception("Couldn't get templates.")
            raise
        else:
            return templates


    def update_template(self, name, subject, text, html):
        """
        Updates a previously created email template.

        :param name: The name of the template.
        :param subject: The subject of the email.
        :param text: The plain text version of the email.
        :param html: The HTML version of the email.
        """
        try:
            template = {
                "TemplateName": name,
                "SubjectPart": subject,
                "TextPart": text,
                "HtmlPart": html,
            }
            self.ses_client.update_template(Template=template)
            logger.info("Updated template %s.", name)
            self.template = template
            self._extract_tags(subject, text, html)
        except ClientError:
            logger.exception("Couldn't update template %s.", name)
            raise
```

Create functions to wrap Amazon SES email actions.

```
class SesDestination:
    """Contains data about an email destination."""

    def __init__(self, tos, ccs=None, bccs=None):
```

```
        """
        :param tos: The list of recipients on the 'To:' line.
        :param ccs: The list of recipients on the 'CC:' line.
        :param bccs: The list of recipients on the 'BCC:' line.
        """
        self.tos = tos
        self.ccs = ccs
        self.bccs = bccs

    def to_service_format(self):
        """
        :return: The destination data in the format expected by Amazon SES.
        """
        svc_format = {"ToAddresses": self.tos}
        if self.ccs is not None:
            svc_format["CcAddresses"] = self.ccs
        if self.bccs is not None:
            svc_format["BccAddresses"] = self.bccs
        return svc_format



class SesMailSender:
    """Encapsulates functions to send emails with Amazon SES."""

    def __init__(self, ses_client):
        """
        :param ses_client: A Boto3 Amazon SES client.
        """
        self.ses_client = ses_client


    def send_email(self, source, destination, subject, text, html,
 reply_tos=None):
        """
        Sends an email.

        Note: If your account is in the Amazon SES  sandbox, the source and
        destination email accounts must both be verified.

        :param source: The source email account.
        :param destination: The destination email account.
        :param subject: The subject of the email.
        :param text: The plain text version of the body of the email.
```

```
        :param html: The HTML version of the body of the email.
        :param reply_tos: Email accounts that will receive a reply if the
recipient
                          replies to the message.
        :return: The ID of the message, assigned by Amazon SES.
        """
        send_args = {
            "Source": source,
            "Destination": destination.to_service_format(),
            "Message": {
                "Subject": {"Data": subject},
                "Body": {"Text": {"Data": text}, "Html": {"Data": html}},
            },
        }
        if reply_tos is not None:
            send_args["ReplyToAddresses"] = reply_tos
        try:
            response = self.ses_client.send_email(**send_args)
            message_id = response["MessageId"]
            logger.info(
                "Sent mail %s from %s to %s.", message_id, source,
destination.tos
            )
        except ClientError:
            logger.exception(
                "Couldn't send mail from %s to %s.", source, destination.tos
            )
            raise
        else:
            return message_id


    def send_templated_email(
        self, source, destination, template_name, template_data, reply_tos=None
    ):
        """
        Sends an email based on a template. A template contains replaceable tags
        each enclosed in two curly braces, such as {{name}}. The template data
passed
        in this function contains key-value pairs that define the values to
insert
        in place of the template tags.

        Note: If your account is in the Amazon SES  sandbox, the source and
```

```
        destination email accounts must both be verified.

        :param source: The source email account.
        :param destination: The destination email account.
        :param template_name: The name of a previously created template.
        :param template_data: JSON-formatted key-value pairs of replacement
values
                              that are inserted in the template before it is
sent.
        :return: The ID of the message, assigned by Amazon SES.
        """
        send_args = {
            "Source": source,
            "Destination": destination.to_service_format(),
            "Template": template_name,
            "TemplateData": json.dumps(template_data),
        }
        if reply_tos is not None:
            send_args["ReplyToAddresses"] = reply_tos
        try:
            response = self.ses_client.send_templated_email(**send_args)
            message_id = response["MessageId"]
            logger.info(
                "Sent templated mail %s from %s to %s.",
                message_id,
                source,
                destination.tos,
            )
        except ClientError:
            logger.exception(
                "Couldn't send templated mail from %s to %s.", source,
destination.tos
            )
            raise
        else:
            return message_id
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateTemplate
  - DeleteIdentity

- DeleteTemplate
- GetIdentityVerificationAttributes
- GetTemplate
- ListIdentities
- ListTemplates
- SendEmail
- SendTemplatedEmail
- UpdateTemplate
- VerifyDomainIdentity
- VerifyEmailIdentity

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples for Amazon SES API v2 using AWS SDKs

The following code examples show how to use Amazon SES API v2 with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon SES API v2

- Basic examples for Amazon SES API v2 using AWS SDKs
  - Hello Amazon SES API v2
  - Actions for Amazon SES API v2 using AWS SDKs
    - Use CreateContact with an AWS SDK

    - Use CreateContactList with an AWS SDK
    - Use CreateEmailIdentity with an AWS SDK
    - Use CreateEmailTemplate with an AWS SDK
    - Use DeleteContactList with an AWS SDK
    - Use DeleteEmailIdentity with an AWS SDK
    - Use DeleteEmailTemplate with an AWS SDK
    - Use GetEmailIdentity with an AWS SDK
    - Use ListContactLists with an AWS SDK
    - Use ListContacts with an AWS SDK
    - Use SendBulkEmail with an AWS SDK
    - Use SendEmail with an AWS SDK
- Scenarios for Amazon SES API v2 using AWS SDKs
    - Send emails with attachments using Amazon SES API v2
    - A complete Amazon SES API v2 Newsletter scenario using an AWS SDK

Basic examples for Amazon SES API v2 using AWS SDKs

The following code examples show how to use the basics of Amazon Simple Email Service API v2 with AWS SDKs.

Examples

- Hello Amazon SES API v2
- Actions for Amazon SES API v2 using AWS SDKs
    - Use CreateContact with an AWS SDK
    - Use CreateContactList with an AWS SDK
    - Use CreateEmailIdentity with an AWS SDK
    - Use CreateEmailTemplate with an AWS SDK
    - Use DeleteContactList with an AWS SDK
    - Use DeleteEmailIdentity with an AWS SDK
    - Use DeleteEmailTemplate with an AWS SDK
    - Use GetEmailIdentity with an AWS SDK
    - Use ListContactLists with an AWS SDK

- Use ListContacts with an AWS SDK
- Use SendBulkEmail with an AWS SDK
- Use SendEmail with an AWS SDK

Hello Amazon SES API v2

The following code example shows how to get started using Amazon SES API v2.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def hello_sesv2(sesv2_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon SESv2 client and
    list the email identities in your account. This example uses the default
    settings specified in your shared credentials and config files.

    :param sesv2_client: A Boto3 SESv2 client object.
    """
    print("Hello, Amazon SESv2. Let's list up to 5 email identities:\n")

    try:
        response = sesv2_client.list_email_identities(PageSize=5)
        identities = response["EmailIdentities"]

        if not identities:
            print(
                "No email identities found. "
                "Use CreateEmailIdentity to add one."
            )
        else:
            for identity in identities:
                print(
```

```
                    f"  Identity: {identity['IdentityName']}"
                    f"  Type: {identity['IdentityType']}"
                    f"  Status: {identity['VerificationStatus']}"
                    f"  Sending: {'Enabled' if identity['SendingEnabled'] else
'Disabled'}"
                )
            print(f"\nShowing {len(identities)} email identity(ies).")

    except ClientError as err:
        logger.error(
            "Couldn't list email identities. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- For API details, see ListEmailIdentities in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for Amazon SES API v2 using AWS SDKs

The following code examples demonstrate how to perform individual Amazon SES API v2 actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon SES API v2 API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon SES API v2 using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Simple Email Service API v2 API Reference.

Examples

- Use CreateContact with an AWS SDK
- Use CreateContactList with an AWS SDK

- Use CreateEmailIdentity with an AWS SDK
- Use CreateEmailTemplate with an AWS SDK
- Use DeleteContactList with an AWS SDK
- Use DeleteEmailIdentity with an AWS SDK
- Use DeleteEmailTemplate with an AWS SDK
- Use GetEmailIdentity with an AWS SDK
- Use ListContactLists with an AWS SDK
- Use ListContacts with an AWS SDK
- Use SendBulkEmail with an AWS SDK
- Use SendEmail with an AWS SDK

Use `CreateContact` with an AWS SDK

The following code examples show how to use `CreateContact`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Creates a contact and adds it to the specified contact list.
/// </summary>
/// <param name="emailAddress">The email address of the contact.</param>
/// <param name="contactListName">The name of the contact list.</param>
```

```
    /// <returns>The response from the CreateContact operation.</returns>
    public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
    {
        var request = new CreateContactRequest
        {
            EmailAddress = emailAddress,
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }
```

- For API details, see CreateContact in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
      try {
        // Create a new contact with the provided email address in the
        CreateContactRequest contactRequest = CreateContactRequest.builder()
            .contactListName(CONTACT_LIST_NAME)
            .emailAddress(emailAddress)
            .build();

        sesClient.createContact(contactRequest);
        contacts.add(emailAddress);

        System.out.println("Contact created: " + emailAddress);

        // Send a welcome email to the new contact
        String welcomeHtml = Files.readString(Paths.get("resources/
coupon_newsletter/welcome.html"));
        String welcomeText = Files.readString(Paths.get("resources/
coupon_newsletter/welcome.txt"));

        SendEmailRequest welcomeEmailRequest = SendEmailRequest.builder()
            .fromEmailAddress(this.verifiedEmail)
            .destination(Destination.builder().toAddresses(emailAddress).build())
            .content(EmailContent.builder()
                .simple(
                    Message.builder()
                        .subject(Content.builder().data("Welcome to the Weekly
 Coupons Newsletter").build())
                        .body(Body.builder()
                            .text(Content.builder().data(welcomeText).build())
                            .html(Content.builder().data(welcomeHtml).build())
                            .build())
                        .build())
                .build())
```

```
            .build();
        SendEmailResponse welcomeEmailResponse =
sesClient.sendEmail(welcomeEmailRequest);
        System.out.println("Welcome email sent: " +
welcomeEmailResponse.messageId());
     } catch (AlreadyExistsException e) {
        // If the contact already exists, skip this step for that contact and
proceed
        // with the next contact
        System.out.println("Contact already exists, skipping creation...");
     } catch (Exception e) {
        System.err.println("Error occurred while processing email address " +
emailAddress + ": " + e.getMessage());
        throw e;
     }
   }
```

- For API details, see CreateContact in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
```

```
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


            try:
                # Create a new contact
                self.ses_client.create_contact(
                    ContactListName=CONTACT_LIST_NAME, EmailAddress=email
                )
                print(f"Contact with email '{email}' created successfully.")

                # Send the welcome email
                self.ses_client.send_email(
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email]},
                    Content={
                        "Simple": {
                            "Subject": {
                                "Data": "Welcome to the Weekly Coupons
 Newsletter"
                            },
                            "Body": {
                                "Text": {"Data": welcome_text},
                                "Html": {"Data": welcome_html},
                            },
                        }
                    },
                )
                print(f"Welcome email sent to '{email}'.")
                if self.sleep:
                    # 1 email per second in sandbox mode, remove in production.
                    sleep(1.1)
```

```
            except ClientError as e:
                # If the contact already exists, skip and proceed
                if e.response["Error"]["Code"] == "AlreadyExistsException":
                    print(f"Contact with email '{email}' already exists.
 Skipping...")
                else:
                    raise e
```

- For API details, see CreateContact in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn add_contact(client: &Client, list: &str, email: &str) -> Result<(),
 Error> {
    client
        .create_contact()
        .contact_list_name(list)
        .email_address(email)
        .send()
        .await?;

    println!("Created contact");

    Ok(())
}
```

- For API details, see CreateContact in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_se2->createcontact(
      iv_contactlistname = iv_contact_list_name
      iv_emailaddress = iv_email_address ).
    MESSAGE 'Contact created successfully.' TYPE 'I'.
  CATCH /aws1/cx_se2alreadyexistsex.
    MESSAGE 'Contact already exists.' TYPE 'I'.
  CATCH /aws1/cx_se2badrequestex.
    MESSAGE 'Bad request.' TYPE 'E'.
  CATCH /aws1/cx_se2notfoundexception.
    MESSAGE 'Contact list not found.' TYPE 'E'.
ENDTRY.
```

- For API details, see CreateContact in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateContactList` with an AWS SDK

The following code examples show how to use `CreateContactList`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
```

```
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact
 list: {ex.Message}");
        }
        return false;
    }
```

- For API details, see CreateContactList in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    try {
      // 2. Create a contact list
      String contactListName = CONTACT_LIST_NAME;
      CreateContactListRequest createContactListRequest =
 CreateContactListRequest.builder()
          .contactListName(contactListName)
          .build();
      sesClient.createContactList(createContactListRequest);
      System.out.println("Contact list created: " + contactListName);
    } catch (AlreadyExistsException e) {
      System.out.println("Contact list already exists, skipping creation: weekly-
coupons-newsletter");
    } catch (LimitExceededException e) {
      System.err.println("Limit for contact lists has been exceeded.");
      throw e;
    } catch (SesV2Exception e) {
      System.err.println("Error creating contact list: " + e.getMessage());
```

```
            throw e;
        }
```

- For API details, see CreateContactList in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep
```

```
        try:

 self.ses_client.create_contact_list(ContactListName=CONTACT_LIST_NAME)
            print(f"Contact list '{CONTACT_LIST_NAME}' created successfully.")
        except ClientError as e:
            # If the contact list already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
                print(f"Contact list '{CONTACT_LIST_NAME}' already exists.")
            else:
                raise e
```

- For API details, see CreateContactList in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn make_list(client: &Client, contact_list: &str) -> Result<(), Error> {
    client
        .create_contact_list()
        .contact_list_name(contact_list)
        .send()
        .await?;

    println!("Created contact list.");

    Ok(())
}
```

- For API details, see CreateContactList in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_se2->createcontactlist(
      iv_contactlistname = iv_contact_list_name ).
    MESSAGE 'Contact list created successfully.' TYPE 'I'.
  CATCH /aws1/cx_se2alreadyexistsex.
    MESSAGE 'Contact list already exists.' TYPE 'I'.
  CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
    MESSAGE 'Bad request - contact list limit may be reached.' TYPE 'I'.
    " Re-raise the exception so the caller can handle it
    RAISE EXCEPTION lo_bad_request.
  CATCH /aws1/cx_se2limitexceededex INTO DATA(lo_limit_exceeded).
    MESSAGE 'Limit exceeded - contact list limit reached.' TYPE 'I'.
    " Re-raise the exception so the caller can handle it
    RAISE EXCEPTION lo_limit_exceeded.
ENDTRY.
```

- For API details, see CreateContactList in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateEmailIdentity` with an AWS SDK

The following code examples show how to use `CreateEmailIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Email Attachments Scenario

- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Creates an email identity (email address or domain) and starts the
verification process.
   /// </summary>
   /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
   /// <returns>The response from the CreateEmailIdentity operation.</returns>
   public async Task<CreateEmailIdentityResponse>
CreateEmailIdentityAsync(string emailIdentity)
   {
       var request = new CreateEmailIdentityRequest
       {
           EmailIdentity = emailIdentity
       };

       try
       {
           var response = await _sesClient.CreateEmailIdentityAsync(request);
           return response;
       }
       catch (AlreadyExistsException ex)
       {
           Console.WriteLine($"Email identity {emailIdentity} already exists.");
           Console.WriteLine(ex.Message);
           throw;
       }
       catch (ConcurrentModificationException ex)
       {
           Console.WriteLine($"The email identity {emailIdentity} is being
modified by another operation or thread.");
```

```
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email identities has been
exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email
identity: {ex.Message}");
            throw;
        }
    }
```

- For API details, see CreateEmailIdentity in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   try {
     CreateEmailIdentityRequest createEmailIdentityRequest =
CreateEmailIdentityRequest.builder()
         .emailIdentity(verifiedEmail)
         .build();
     sesClient.createEmailIdentity(createEmailIdentityRequest);
     System.out.println("Email identity created: " + verifiedEmail);
   } catch (AlreadyExistsException e) {
     System.out.println("Email identity already exists, skipping creation: " +
verifiedEmail);
   } catch (NotFoundException e) {
     System.err.println("The provided email address is not verified: " +
verifiedEmail);
     throw e;
   } catch (LimitExceededException e) {
     System.err
         .println("You have reached the limit for email identities. Please
remove some identities and try again.");
     throw e;
   } catch (SesV2Exception e) {
     System.err.println("Error creating email identity: " + e.getMessage());
     throw e;
   }
```

- For API details, see CreateEmailIdentity in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def create_email_identity(self, email_address: str) -> Dict[str, Any]:
        """
        Starts the process of verifying an email identity (email address or
 domain).

        :param email_address: The email address or domain to verify.
        :return: A dictionary with the identity type and verification status.
        :raises ClientError: If the limit is exceeded (LimitExceededException).
        """
        try:
```

```
            response = self.sesv2_client.create_email_identity(
                EmailIdentity=email_address
            )
            logger.info(
                "Started verification for email identity %s.", email_address
            )
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "LimitExceededException":
                logger.error(
                    "Couldn't create email identity %s. You have exceeded "
                    "the maximum number of email identities. "
                    "Use an existing verified identity.",
                    email_address,
                )
            else:
                logger.error(
                    "Couldn't create email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see CreateEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
```

```
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:

 self.ses_client.create_email_identity(EmailIdentity=self.verified_email)
            print(f"Email identity '{self.verified_email}' created
 successfully.")
        except ClientError as e:
            # If the email identity already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
                print(f"Email identity '{self.verified_email}' already exists.")
            else:
                raise e
```

- For API details, see CreateEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        match self
            .client
            .create_email_identity()
            .email_identity(self.verified_email.clone())
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Email identity created
successfully.")?,
            Err(e) => match e.into_service_error() {
                CreateEmailIdentityError::AlreadyExistsException(_) => {
                    writeln!(
                        self.stdout,
                        "Email identity already exists, skipping creation."
                    )?;
                }
                e => return Err(anyhow!("Error creating email identity: {}", e)),
            },
        }
```

- For API details, see CreateEmailIdentity in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_se2->createemailidentity(
      iv_emailidentity = iv_email_identity ).
    MESSAGE 'Email identity created successfully.' TYPE 'I'.
  CATCH /aws1/cx_se2alreadyexistsex.
    MESSAGE 'Email identity already exists.' TYPE 'I'.
  CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
    MESSAGE lo_bad_request TYPE 'I' DISPLAY LIKE 'E'.
  CATCH /aws1/cx_se2limitexceededex INTO DATA(lo_limit_exceeded).
    MESSAGE lo_limit_exceeded TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.
```

- For API details, see CreateEmailIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateEmailTemplate` with an AWS SDK

The following code examples show how to use `CreateEmailTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Email Attachments Scenario
- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
exists.");
            Console.WriteLine(ex.Message);
```

```
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been
 exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email
 template: {ex.Message}");
        }

        return false;
    }
```

- For API details, see CreateEmailTemplate in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    try {
      // Create an email template named "weekly-coupons"
      String newsletterHtml = loadFile("resources/coupon_newsletter/coupon-
newsletter.html");
      String newsletterText = loadFile("resources/coupon_newsletter/coupon-
newsletter.txt");
```

```
      CreateEmailTemplateRequest templateRequest =
CreateEmailTemplateRequest.builder()
          .templateName(TEMPLATE_NAME)
          .templateContent(EmailTemplateContent.builder()
              .subject("Weekly Coupons Newsletter")
              .html(newsletterHtml)
              .text(newsletterText)
              .build())
          .build();

      sesClient.createEmailTemplate(templateRequest);

      System.out.println("Email template created: " + TEMPLATE_NAME);
    } catch (AlreadyExistsException e) {
      // If the template already exists, skip this step and proceed with the next
      // operation
      System.out.println("Email template already exists, skipping creation...");
    } catch (LimitExceededException e) {
      // If the limit for email templates is exceeded, fail the workflow and
inform
      // the user
      System.err.println("You have reached the limit for email templates. Please
remove some templates and try again.");
      throw e;
    } catch (Exception e) {
      System.err.println("Error occurred while creating email template: " +
e.getMessage());
      throw e;
    }
```

- For API details, see CreateEmailTemplate in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def create_email_template(
        self,
        template_name: str,
        subject: str,
        html_body: str,
        text_body: str,
    ) -> None:
        """
        Creates an email template for use with templated and bulk email sends.

        :param template_name: The name for the new template.
        :param subject: The subject line of the template. May include
 {{placeholders}}.
        :param html_body: The HTML body of the template.
        :param text_body: The plain text body of the template.
        :raises ClientError: If the template limit is exceeded
 (LimitExceededException).
        """
        try:
            self.sesv2_client.create_email_template(
                TemplateName=template_name,
```

```
                TemplateContent={
                    "Subject": subject,
                    "Html": html_body,
                    "Text": text_body,
                },
            )
            logger.info("Created email template %s.", template_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "LimitExceededException":
                logger.error(
                    "Couldn't create email template %s. You have exceeded "
                    "the maximum number of email templates. "
                    "Delete unused templates first.",
                    template_name,
                )
            else:
                logger.error(
                    "Couldn't create email template %s. Here's why: %s: %s",
                    template_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see CreateEmailTemplate in *AWS SDK for Python (Boto3) API Reference*.

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
```

```
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:
            template_content = {
                "Subject": "Weekly Coupons Newsletter",
                "Html": load_file_content("coupon-newsletter.html"),
                "Text": load_file_content("coupon-newsletter.txt"),
            }
            self.ses_client.create_email_template(
                TemplateName=TEMPLATE_NAME, TemplateContent=template_content
            )
            print(f"Email template '{TEMPLATE_NAME}' created successfully.")
        except ClientError as e:
            # If the template already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
                print(f"Email template '{TEMPLATE_NAME}' already exists.")
            else:
                raise e
```

- For API details, see CreateEmailTemplate in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        let template_html =
            std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.html")
                .unwrap_or_else(|_| "Missing coupon-
newsletter.html".to_string());
        let template_text =
            std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.txt")
                .unwrap_or_else(|_| "Missing coupon-newsletter.txt".to_string());

        // Create the email template
        let template_content = EmailTemplateContent::builder()
            .subject("Weekly Coupons Newsletter")
            .html(template_html)
            .text(template_text)
            .build();

        match self
            .client
            .create_email_template()
            .template_name(TEMPLATE_NAME)
            .template_content(template_content)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Email template created
 successfully.")?,
            Err(e) => match e.into_service_error() {
                CreateEmailTemplateError::AlreadyExistsException(_) => {
                    writeln!(
                        self.stdout,
                        "Email template already exists, skipping creation."
```

```
                )?;
            }
            e => return Err(anyhow!("Error creating email template: {}", e)),
        },
    }
```

- For API details, see CreateEmailTemplate in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    DATA(lo_template_content) = NEW /aws1/cl_se2emailtmplcontent(
      iv_subject = iv_subject
      iv_html = iv_html
      iv_text = iv_text ).

    lo_se2->createemailtemplate(
      iv_templatename = iv_template_name
      io_templatecontent = lo_template_content ).
    MESSAGE 'Email template created successfully.' TYPE 'I'.
  CATCH /aws1/cx_se2alreadyexistsex.
    MESSAGE 'Email template already exists.' TYPE 'I'.
  CATCH /aws1/cx_se2badrequestex.
    MESSAGE 'Bad request.' TYPE 'E'.
  CATCH /aws1/cx_se2limitexceededex.
    MESSAGE 'Limit exceeded.' TYPE 'E'.
ENDTRY.
```

- For API details, see CreateEmailTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteContactList` with an AWS SDK

The following code examples show how to use `DeleteContactList`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteContactListAsync(string contactListName)
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
```

```
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being
modified by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact
list: {ex.Message}");
        }

        return false;
    }
```

- For API details, see DeleteContactList in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    try {
      // Delete the contact list
```

```
      DeleteContactListRequest deleteContactListRequest =
 DeleteContactListRequest.builder()
          .contactListName(CONTACT_LIST_NAME)
          .build();

      sesClient.deleteContactList(deleteContactListRequest);

      System.out.println("Contact list deleted: " + CONTACT_LIST_NAME);
    } catch (NotFoundException e) {
      // If the contact list does not exist, log the error and proceed
      System.out.println("Contact list not found. Skipping deletion...");
    } catch (Exception e) {
      System.err.println("Error occurred while deleting the contact list: " +
 e.getMessage());
      e.printStackTrace();
    }
```

- For API details, see DeleteContactList in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
```

```
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:

 self.ses_client.delete_contact_list(ContactListName=CONTACT_LIST_NAME)
            print(f"Contact list '{CONTACT_LIST_NAME}' deleted successfully.")
        except ClientError as e:
            # If the contact list doesn't exist, skip and proceed
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Contact list '{CONTACT_LIST_NAME}' does not exist.")
            else:
                print(e)
```

- For API details, see DeleteContactList in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        match self
            .client
```

```
            .delete_contact_list()
            .contact_list_name(CONTACT_LIST_NAME)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Contact list deleted
successfully.")?,
            Err(e) => return Err(anyhow!("Error deleting contact list: {e}")),
        }
```

- For API details, see DeleteContactList in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_se2->deletecontactlist(
          iv_contactlistname = iv_contact_list_name ).
        MESSAGE 'Contact list deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_se2notfoundexception.
        MESSAGE 'Contact list not found.' TYPE 'I'.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE 'Bad request.' TYPE 'I'.
        RAISE EXCEPTION lo_bad_request.
    ENDTRY.
```

- For API details, see DeleteContactList in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteEmailIdentity` with an AWS SDK

The following code examples show how to use `DeleteEmailIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Email Attachments Scenario
- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
```

```
            Console.WriteLine($"The email identity {emailIdentity} is being
modified by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email
identity: {ex.Message}");
        }

        return false;
    }
```

- For API details, see DeleteEmailIdentity in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
     try {
       // Delete the email identity
       DeleteEmailIdentityRequest deleteIdentityRequest =
DeleteEmailIdentityRequest.builder()
```

```
            .emailIdentity(this.verifiedEmail)
            .build();

        sesClient.deleteEmailIdentity(deleteIdentityRequest);

        System.out.println("Email identity deleted: " + this.verifiedEmail);
      } catch (NotFoundException e) {
        // If the email identity does not exist, log the error and proceed
        System.out.println("Email identity not found. Skipping deletion...");
      } catch (Exception e) {
        System.err.println("Error occurred while deleting the email identity: " +
 e.getMessage());
        e.printStackTrace();
      }
    } else {
      System.out.println("Skipping email identity deletion.");
    }
```

- For API details, see DeleteEmailIdentity in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client
```

```
    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def delete_email_identity(self, email_address: str) -> None:
        """
        Deletes an email identity.

        :param email_address: The email address or domain to delete.
        :raises ClientError: If the identity is not found (NotFoundException).
        """
        try:
            self.sesv2_client.delete_email_identity(
                EmailIdentity=email_address
            )
            logger.info("Deleted email identity %s.", email_address)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email identity %s not found or already deleted.",
                    email_address,
                )
            else:
                logger.error(
                    "Couldn't delete email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

- For API details, see DeleteEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


            try:

 self.ses_client.delete_email_identity(EmailIdentity=self.verified_email)
                print(f"Email identity '{self.verified_email}' deleted
 successfully.")
            except ClientError as e:
```

```
                # If the email identity doesn't exist, skip and proceed
                if e.response["Error"]["Code"] == "NotFoundException":
                    print(f"Email identity '{self.verified_email}' does not
exist.")
                else:
                    print(e)
```

- For API details, see DeleteEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
            match self
                .client
                .delete_email_identity()
                .email_identity(self.verified_email.clone())
                .send()
                .await
            {
                Ok(_) => writeln!(self.stdout, "Email identity deleted
successfully.")?,
                Err(e) => {
                    return Err(anyhow!("Error deleting email identity: {}", e));
                }
            }
```

- For API details, see DeleteEmailIdentity in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_se2->deleteemailidentity(
      iv_emailidentity = iv_email_identity ).
    MESSAGE 'Email identity deleted successfully.' TYPE 'I'.
  CATCH /aws1/cx_se2notfoundexception.
    MESSAGE 'Email identity not found.' TYPE 'I'.
  CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
    MESSAGE 'Bad request.' TYPE 'I'.
    RAISE EXCEPTION lo_bad_request.
ENDTRY.
```

- For API details, see DeleteEmailIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteEmailTemplate` with an AWS SDK

The following code examples show how to use `DeleteEmailTemplate`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Email Attachments Scenario
- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while deleting the email
template: {ex.Message}");
        }

        return false;
    }
```

- For API details, see DeleteEmailTemplate in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    try {
      // Delete the template
      DeleteEmailTemplateRequest deleteTemplateRequest =
DeleteEmailTemplateRequest.builder()
          .templateName(TEMPLATE_NAME)
          .build();

      sesClient.deleteEmailTemplate(deleteTemplateRequest);

      System.out.println("Email template deleted: " + TEMPLATE_NAME);
    } catch (NotFoundException e) {
      // If the email template does not exist, log the error and proceed
      System.out.println("Email template not found. Skipping deletion...");
    } catch (Exception e) {
      System.err.println("Error occurred while deleting the email template: " +
e.getMessage());
      e.printStackTrace();
    }
```

- For API details, see DeleteEmailTemplate in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def delete_email_template(self, template_name: str) -> None:
        """
        Deletes an email template.

        :param template_name: The name of the template to delete.
        :raises ClientError: If the template is not found (NotFoundException).
        """
        try:
            self.sesv2_client.delete_email_template(
                TemplateName=template_name
```

```
            )
            logger.info("Deleted email template %s.", template_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email template %s not found or already deleted.",
                    template_name,
                )
            else:
                logger.error(
                    "Couldn't delete email template %s. Here's why: %s: %s",
                    template_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see DeleteEmailTemplate in *AWS SDK for Python (Boto3) API Reference*.

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
```

```
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:
            self.ses_client.delete_email_template(TemplateName=TEMPLATE_NAME)
            print(f"Email template '{TEMPLATE_NAME}' deleted successfully.")
        except ClientError as e:
            # If the email template doesn't exist, skip and proceed
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Email template '{TEMPLATE_NAME}' does not exist.")
            else:
                print(e)
```

- For API details, see DeleteEmailTemplate in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        match self
            .client
            .delete_email_template()
            .template_name(TEMPLATE_NAME)
            .send()
```

```
            .await
        {
            Ok(_) => writeln!(self.stdout, "Email template deleted
successfully.")?,
            Err(e) => {
                return Err(anyhow!("Error deleting email template: {e}"));
            }
        }
```

- For API details, see DeleteEmailTemplate in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_se2->deleteemailtemplate(
          iv_templatename = iv_template_name ).
        MESSAGE 'Email template deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_se2notfoundexception.
        MESSAGE 'Email template not found.' TYPE 'I'.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE 'Bad request.' TYPE 'I'.
        RAISE EXCEPTION lo_bad_request.
    ENDTRY.
```

- For API details, see DeleteEmailTemplate in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetEmailIdentity` with an AWS SDK

The following code examples show how to use `GetEmailIdentity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Email Attachments Scenario

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)
```

```
    def get_email_identity(self, email_address: str) -> Dict[str, Any]:
        """
        Gets information about an email identity, including its verification
status.

        :param email_address: The email address or domain to look up.
        :return: A dictionary with identity information including verification
status.
        :raises ClientError: If the identity is not found (NotFoundException).
        """
        try:
            response = self.sesv2_client.get_email_identity(
                EmailIdentity=email_address
            )
            logger.info("Got email identity for %s.", email_address)
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email identity %s not found.", email_address
                )
            else:
                logger.error(
                    "Couldn't get email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see GetEmailIdentity in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Determines whether an email address has been verified.

```
async fn is_verified(client: &Client, email: &str) -> Result<(), Error> {
    let resp = client
        .get_email_identity()
        .email_identity(email)
        .send()
        .await?;

    if resp.verified_for_sending_status() {
        println!("The address is verified");
    } else {
        println!("The address is not verified");
    }

    Ok(())
}
```

- For API details, see GetEmailIdentity in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_se2->getemailidentity(
          iv_emailidentity = iv_email_identity ).
        MESSAGE |Identity type: { oo_result->get_identitytype( ) }, | &&
                |verified for sending: { oo_result-
>get_verifiedforsendingstatus( ) }| TYPE 'I'.
      CATCH /aws1/cx_se2notfoundexception.
        MESSAGE |Email identity { iv_email_identity } not found.| TYPE 'I'.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE lo_bad_request TYPE 'I' DISPLAY LIKE 'E'.
```

```
        RAISE EXCEPTION lo_bad_request.
    ENDTRY.
```

- For API details, see GetEmailIdentity in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListContactLists` with an AWS SDK

The following code example shows how to use `ListContactLists`.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_lists(client: &Client) -> Result<(), Error> {
    let resp = client.list_contact_lists().send().await?;

    println!("Contact lists:");

    for list in resp.contact_lists() {
        println!("  {}", list.contact_list_name().unwrap_or_default());
    }

    Ok(())
}
```

- For API details, see ListContactLists in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListContacts` with an AWS SDK

The following code examples show how to use `ListContacts`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Newsletter scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
```

```
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
        }

        return new List<Contact>();
    }
```

- For API details, see ListContacts in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
     ListContactsRequest contactListRequest = ListContactsRequest.builder()
         .contactListName(CONTACT_LIST_NAME)
         .build();

     List<String> contactEmails;
     try {
       ListContactsResponse contactListResponse =
sesClient.listContacts(contactListRequest);
```

```
        contactEmails = contactListResponse.contacts().stream()
            .map(Contact::emailAddress)
            .toList();
      } catch (Exception e) {
        // TODO: Remove when listContacts's GET body issue is resolved.
        contactEmails = this.contacts;
      }
```

- For API details, see ListContacts in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
```

```
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:
            contacts_response = self.ses_client.list_contacts(
                ContactListName=CONTACT_LIST_NAME
            )
        except ClientError as e:
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Contact list '{CONTACT_LIST_NAME}' does not exist.")
                return
            else:
                raise e
```

- For API details, see ListContacts in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_contacts(client: &Client, list: &str) -> Result<(), Error> {
    let resp = client
        .list_contacts()
        .contact_list_name(list)
        .send()
        .await?;

    println!("Contacts:");
```

```
    for contact in resp.contacts() {
        println!("  {}", contact.email_address().unwrap_or_default());
    }

    Ok(())
}
```

- For API details, see ListContacts in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_se2->listcontacts(
          iv_contactlistname = iv_contact_list_name ).
        DATA(lv_count) = lines( oo_result->get_contacts( ) ).
        MESSAGE |Retrieved { lv_count } contacts from list.| TYPE 'I'.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE 'Bad request.' TYPE 'I'.
        RAISE EXCEPTION lo_bad_request.
      CATCH /aws1/cx_se2notfoundexception INTO DATA(lo_not_found).
        MESSAGE 'Contact list not found.' TYPE 'I'.
        RAISE EXCEPTION lo_not_found.
    ENDTRY.
```

- For API details, see ListContacts in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendBulkEmail` with an AWS SDK

The following code examples show how to use `SendBulkEmail`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Email Attachments Scenario

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def send_bulk_email(
```

```
        self,
        from_address: str,
        template_name: str,
        default_template_data: str,
        bulk_entries: List[Dict[str, Any]],
        attachments: Optional[List[Dict[str, Any]]] = None,
    ) -> List[Dict[str, Any]]:
        """
        Sends a templated email to multiple recipients in a single API call.

        All recipients receive the same attachment(s) defined in the default
        content, while template data can be personalized per recipient.

        :param from_address: The verified sender email address.
        :param template_name: The name of an existing email template.
        :param default_template_data: Default JSON template data string.
        :param bulk_entries: A list of BulkEmailEntry dicts, each containing
            'Destination' and optionally 'ReplacementEmailContent'.
        :param attachments: An optional list of attachment dicts for all
            recipients.
        :return: A list of BulkEmailEntryResult dicts with status and MessageId.
        :raises ClientError: If the message is rejected (MessageRejected).
        """
        try:
            template_content: Dict[str, Any] = {
                "TemplateName": template_name,
                "TemplateData": default_template_data,
            }

            if attachments:
                template_content["Attachments"] = attachments

            response = self.sesv2_client.send_bulk_email(
                FromEmailAddress=from_address,
                DefaultContent={"Template": template_content},
                BulkEmailEntries=bulk_entries,
            )
            results = response.get("BulkEmailEntryResults", [])
            logger.info(
                "Sent bulk email from %s to %d recipients.",
                from_address,
                len(bulk_entries),
            )
            return results
```

```
        except ClientError as err:
            if err.response["Error"]["Code"] == "MessageRejected":
                logger.error(
                    "Bulk message was rejected. Check that the template "
                    "exists, attachment file types are supported, and "
                    "total message size is within limits. Details: %s",
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't send bulk email. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see SendBulkEmail in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    " Build the default template content used for all bulk recipients
    DATA(lo_template) = NEW /aws1/cl_se2template(
      iv_templatename = iv_template_name
      iv_templatedata = iv_template_data ).

    DATA(lo_default_content) = NEW /aws1/cl_se2bulkemailcontent(
      io_template = lo_template ).

    DATA(lo_result) = lo_se2->sendbulkemail(
      iv_fromemailaddress = iv_from_address
      io_defaultcontent   = lo_default_content
```

```
          it_bulkemailentries = it_bulk_entries ).

        ot_results = lo_result->get_bulkemailentryresults( ).
        MESSAGE |Bulk email sent to { lines( it_bulk_entries ) } recipient(s).|
TYPE 'I'.
      CATCH /aws1/cx_se2messagerejected INTO DATA(lo_rejected).
        MESSAGE lo_rejected TYPE 'I' DISPLAY LIKE 'E'.
        RAISE EXCEPTION lo_rejected.
      CATCH /aws1/cx_se2mailfrmdomnotver00 INTO DATA(lo_not_verified).
        MESSAGE lo_not_verified TYPE 'I' DISPLAY LIKE 'E'.
        RAISE EXCEPTION lo_not_verified.
      CATCH /aws1/cx_se2notfoundexception INTO DATA(lo_not_found).
        MESSAGE lo_not_found TYPE 'I' DISPLAY LIKE 'E'.
        RAISE EXCEPTION lo_not_found.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE lo_bad_request TYPE 'I' DISPLAY LIKE 'E'.
        RAISE EXCEPTION lo_bad_request.
    ENDTRY.
```

- For API details, see SendBulkEmail in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendEmail` with an AWS SDK

The following code examples show how to use `SendEmail`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Email Attachments Scenario

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Sends an email with the specified content and options.
   /// </summary>
   /// <param name="fromEmailAddress">The email address to send the email
from.</param>
   /// <param name="toEmailAddresses">The email addresses to send the email
to.</param>
   /// <param name="subject">The subject of the email.</param>
   /// <param name="htmlContent">The HTML content of the email.</param>
   /// <param name="textContent">The text content of the email.</param>
   /// <param name="templateName">The name of the email template to use
(optional).</param>
   /// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
   /// <param name="contactListName">The name of the contact list for
unsubscribe functionality (optional).</param>
   /// <returns>The MessageId response from the SendEmail operation.</returns>
   public async Task<string> SendEmailAsync(string fromEmailAddress,
List<string> toEmailAddresses, string? subject,
       string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
   {
       var request = new SendEmailRequest
       {
           FromEmailAddress = fromEmailAddress
       };

       if (toEmailAddresses.Any())
       {
           request.Destination = new Destination { ToAddresses =
toEmailAddresses };
       }
```

```
        if (!string.IsNullOrEmpty(templateName))
        {
            request.Content = new EmailContent()
            {
                Template = new Template
                {
                    TemplateName = templateName,
                    TemplateData = templateData
                }
            };
        }
        else
        {
            request.Content = new EmailContent
            {
                Simple = new Message
                {
                    Subject = new Content { Data = subject },
                    Body = new Body
                    {
                        Html = new Content { Data = htmlContent },
                        Text = new Content { Data = textContent }
                    }
                }
            };
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
```

```
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
        }
        catch (MailFromDomainNotVerifiedException ex)
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
        }

        return string.Empty;
    }
```

- For API details, see SendEmail in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sends a message.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sesv2.model.Body;
import software.amazon.awssdk.services.sesv2.model.Content;
import software.amazon.awssdk.services.sesv2.model.Destination;
import software.amazon.awssdk.services.sesv2.model.EmailContent;
import software.amazon.awssdk.services.sesv2.model.Message;
import software.amazon.awssdk.services.sesv2.model.SendEmailRequest;
import software.amazon.awssdk.services.sesv2.model.SesV2Exception;
import software.amazon.awssdk.services.sesv2.SesV2Client;

/**
 * Before running this AWS SDK for Java (v2) example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class SendEmail {
    public static void main(String[] args) {
        final String usage = """

                            Usage:
                                <sender> <recipient> <subject>\s

                            Where:
                                sender - An email address that represents the
 sender.\s
```

```
                                recipient - An email address that represents the
 recipient.\s
                                subject - The subject line.\s
                            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String sender = args[0];
        String recipient = args[1];
        String subject = args[2];

        Region region = Region.US_EAST_1;
        SesV2Client sesv2Client = SesV2Client.builder()
                .region(region)
                .build();

        // The HTML body of the email.
        String bodyHTML = "<html>" + "<head></head>" + "<body>" + "<h1>Hello!</
h1>"
                + "<p> See the list of customers.</p>" + "</body>" + "</html>";

        send(sesv2Client, sender, recipient, subject, bodyHTML);
    }

    public static void send(SesV2Client client,
                            String sender,
                            String recipient,
                            String subject,
                            String bodyHTML) {

        Destination destination = Destination.builder()
                .toAddresses(recipient)
                .build();

        Content content = Content.builder()
                .data(bodyHTML)
                .build();

        Content sub = Content.builder()
                .data(subject)
                .build();
```

```
        Body body = Body.builder()
                .html(content)
                .build();

        Message msg = Message.builder()
                .subject(sub)
                .body(body)
                .build();

        EmailContent emailContent = EmailContent.builder()
                .simple(msg)
                .build();

        SendEmailRequest emailRequest = SendEmailRequest.builder()
                .destination(destination)
                .content(emailContent)
                .fromEmailAddress(sender)
                .build();

        try {
            System.out.println("Attempting to send an email through Amazon SES "
                    + "using the AWS SDK for Java...");
            client.sendEmail(emailRequest);
            System.out.println("email was sent");

        } catch (SesV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Sends a message using a template.

```
      String coupons = Files.readString(Paths.get("resources/coupon_newsletter/
sample_coupons.json"));
      for (String emailAddress : contactEmails) {
        SendEmailRequest newsletterRequest = SendEmailRequest.builder()
            .destination(Destination.builder().toAddresses(emailAddress).build())
            .content(EmailContent.builder()
                .template(Template.builder()
```

```
                    .templateName(TEMPLATE_NAME)
                    .templateData(coupons)
                    .build())
                .build())
            .fromEmailAddress(this.verifiedEmail)
            .listManagementOptions(ListManagementOptions.builder()
                .contactListName(CONTACT_LIST_NAME)
                .build())
            .build();
        SendEmailResponse newsletterResponse =
 sesClient.sendEmail(newsletterRequest);
        System.out.println("Newsletter sent to " + emailAddress + ": " +
 newsletterResponse.messageId());
     }
```

Sends a message with header information.

```
public class SendwithHeader {

    public static void main(String[] args) {
        final String usage = """

            Usage:
                <sender> <recipient> <subject>\s

            Where:
                sender - An email address that represents the sender.\s
                recipient - An email address that represents the recipient.\s
                subject - The subject line.\s
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String sender = args[0];
        String recipient = args[1];
        String subject = args[2];
        Region region = Region.US_EAST_1;
        SesV2Client sesv2Client = SesV2Client.builder()
                .region(region)
```

```
                .build();

        String bodyHTML = """
                <html>
                    <head></head>
                    <body>
                        <h1>Hello!</h1>
                        <p>See the list of customers.</p>
                    </body>
                </html>
                """;

        sendWithHeader(sesv2Client, sender, recipient, subject, bodyHTML);
        sesv2Client.close();
    }

    /**
     * Sends an email using the AWS SES V2 client.
     *
     * @param sesv2Client the SES V2 client to use for sending the email
     * @param sender the email address of the sender
     * @param recipient the email address of the recipient
     * @param subject the subject of the email
     * @param bodyHTML the HTML content of the email body
     */
    public static void sendWithHeader(SesV2Client sesv2Client,
                                      String sender,
                                      String recipient,
                                      String subject,
                                      String bodyHTML) {
        EmailContent emailContent = EmailContent.builder()
                .simple(Message.builder()
                        .body(b -> b.html(c ->
 c.charset(UTF_8.name()).data(bodyHTML))
                                .text(c ->
 c.charset(UTF_8.name()).data(bodyHTML)))
                        .subject(c -> c.charset(UTF_8.name()).data(subject))
                        .headers(List.of(
                                MessageHeader.builder()
                                        .name("List-Unsubscribe")
                                        .value("<https://nutrition.co/?
address=x&topic=x>, <mailto:unsubscribe@nutrition.co?subject=TopicUnsubscribe>")
                                        .build(),
                                MessageHeader.builder()
```

```
                                        .name("List-Unsubscribe-Post")
                                        .value("List-Unsubscribe=One-Click")
                                        .build()))
                        .build())
                .build();

        SendEmailRequest request = SendEmailRequest.builder()
                .fromEmailAddress(sender)
                .destination(d -> d.toAddresses(recipient))
                .content(emailContent)
                .build();

        try {
            SendEmailResponse response = sesv2Client.sendEmail(request);
            System.out.println("Email sent! Message ID: " +
 response.messageId());
        } catch (SesV2Exception e) {
            System.err.println("Failed to send email: " +
 e.awsErrorDetails().errorMessage());
            throw new RuntimeException(e);
        }
    }
}
```

- For API details, see SendEmail in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sends a message with optional attachments.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""
```

```
    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def send_email(
        self,
        from_address: str,
        to_addresses: List[str],
        subject: str,
        html_body: str,
        text_body: str,
        attachments: Optional[List[Dict[str, Any]]] = None,
    ) -> str:
        """
        Sends a simple email message with optional attachments.

        SES handles MIME construction automatically when using attachments
        with the Simple content type, so developers don't need to build
        raw MIME messages.

        :param from_address: The verified sender email address.
        :param to_addresses: A list of recipient email addresses.
        :param subject: The subject line of the email.
        :param html_body: The HTML body content.
        :param text_body: The plain text body content.
        :param attachments: An optional list of attachment dictionaries. Each
            attachment should contain 'RawContent' (bytes), 'FileName' (str),
            and optionally 'ContentType', 'ContentDisposition', 'ContentId',
            'ContentDescription', and 'ContentTransferEncoding'.
```

```
        :return: The MessageId of the sent email.
        :raises ClientError: If the message is rejected (MessageRejected).
        """
        try:
            simple_message: Dict[str, Any] = {
                "Subject": {"Data": subject},
                "Body": {
                    "Html": {"Data": html_body},
                    "Text": {"Data": text_body},
                },
            }

            if attachments:
                simple_message["Attachments"] = attachments

            response = self.sesv2_client.send_email(
                FromEmailAddress=from_address,
                Destination={"ToAddresses": to_addresses},
                Content={"Simple": simple_message},
            )
            message_id = response["MessageId"]
            logger.info(
                "Sent email from %s to %s. MessageId: %s",
                from_address,
                to_addresses,
                message_id,
            )
            return message_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "MessageRejected":
                logger.error(
                    "Message was rejected. Check that attachments use "
                    "supported file types and total message size is "
                    "under 40 MB. Details: %s",
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't send email. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see SendEmail in *AWS SDK for Python (Boto3) API Reference*.

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sends a message to all members of the contact list.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


                self.ses_client.send_email(
```

```
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email]},
                    Content={
                        "Simple": {
                            "Subject": {
                                "Data": "Welcome to the Weekly Coupons
 Newsletter"
                            },
                            "Body": {
                                "Text": {"Data": welcome_text},
                                "Html": {"Data": welcome_html},
                            },
                        }
                    },
                )
                print(f"Welcome email sent to '{email}'.")
```

Sends a message to all members of the contact list using a template.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """
```

```
    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


                self.ses_client.send_email(
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email_address]},
                    Content={
                        "Template": {
                            "TemplateName": TEMPLATE_NAME,
                            "TemplateData": coupon_items,
                        }
                    },
                    ListManagementOptions={"ContactListName": CONTACT_LIST_NAME},
                )
```

- For API details, see SendEmail in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-sesv2'
require_relative 'config' # Recipient and sender email addresses.

# Set up the SESv2 client.
client = Aws::SESV2::Client.new(region: AWS_REGION)

def send_email(client, sender_email, recipient_email)
  response = client.send_email(
    {
      from_email_address: sender_email,
      destination: {
        to_addresses: [recipient_email]
```

```
      },
      content: {
        simple: {
          subject: {
            data: 'Test email subject'
          },
          body: {
            text: {
              data: 'Test email body'
            }
          }
        }
      }
    }
  )
  puts "Email sent from #{SENDER_EMAIL} to #{RECIPIENT_EMAIL} with message ID:
 #{response.message_id}"
end

send_email(client, SENDER_EMAIL, RECIPIENT_EMAIL)
```

- For API details, see SendEmail in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sends a message to all members of the contact list.

```
async fn send_message(
    client: &Client,
    list: &str,
    from: &str,
    subject: &str,
    message: &str,
```

```
) -> Result<(), Error> {
    // Get list of email addresses from contact list.
    let resp = client
        .list_contacts()
        .contact_list_name(list)
        .send()
        .await?;

    let contacts = resp.contacts();

    let cs: Vec<String> = contacts
        .iter()
        .map(|i| i.email_address().unwrap_or_default().to_string())
        .collect();

    let mut dest: Destination = Destination::builder().build();
    dest.to_addresses = Some(cs);
    let subject_content = Content::builder()
        .data(subject)
        .charset("UTF-8")
        .build()
        .expect("building Content");
    let body_content = Content::builder()
        .data(message)
        .charset("UTF-8")
        .build()
        .expect("building Content");
    let body = Body::builder().text(body_content).build();

    let msg = Message::builder()
        .subject(subject_content)
        .body(body)
        .build();

    let email_content = EmailContent::builder().simple(msg).build();

    client
        .send_email()
        .from_email_address(from)
        .destination(dest)
        .content(email_content)
        .send()
        .await?;
```

```
    println!("Email sent to list");

    Ok(())
}
```

Sends a message to all members of the contact list using a template.

```
            let coupons = std::fs::read_to_string("../resources/newsletter/
sample_coupons.json")
                .unwrap_or_else(|_| r#"{"coupons":[]}"#.to_string());
            let email_content = EmailContent::builder()
                .template(
                    Template::builder()
                        .template_name(TEMPLATE_NAME)
                        .template_data(coupons)
                        .build(),
                )
                .build();

            match self
                .client
                .send_email()
                .from_email_address(self.verified_email.clone())

 .destination(Destination::builder().to_addresses(email.clone()).build())
                .content(email_content)
                .list_management_options(
                    ListManagementOptions::builder()
                        .contact_list_name(CONTACT_LIST_NAME)
                        .build()?,
                )
                .send()
                .await
            {
                Ok(output) => {
                    if let Some(message_id) = output.message_id {
                        writeln!(
                            self.stdout,
                            "Newsletter sent to {} with message ID {}",
                            email, message_id
                        )?;
                    } else {
```

```
                        writeln!(self.stdout, "Newsletter sent to {}", email)?;
                    }
                }
                Err(e) => return Err(anyhow!("Error sending newsletter to {}:
 {}", email, e)),
            }
```

- For API details, see SendEmail in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sends a message.

```
    TRY.
        " Create destination with recipient address
        DATA lt_to_addresses TYPE /aws1/
cl_se2emailaddresslist_w=>tt_emailaddresslist.
        APPEND NEW /aws1/cl_se2emailaddresslist_w( iv_value =
 iv_to_email_address ) TO lt_to_addresses.
        DATA(lo_destination) = NEW /aws1/cl_se2destination(
          it_toaddresses = lt_to_addresses ).

        " Create message content
        DATA(lo_subject) = NEW /aws1/cl_se2content( iv_data = iv_subject ).
        DATA(lo_text_body) = NEW /aws1/cl_se2content( iv_data = iv_text_body ).
        DATA(lo_html_body) = NEW /aws1/cl_se2content( iv_data = iv_html_body ).
        DATA(lo_body) = NEW /aws1/cl_se2body(
          io_text = lo_text_body
          io_html = lo_html_body ).
        DATA(lo_message) = NEW /aws1/cl_se2message(
          io_subject = lo_subject
          io_body = lo_body ).
```

```
        DATA(lo_content) = NEW /aws1/cl_se2emailcontent(
          io_simple = lo_message ).

        " Send the email
        lo_se2->sendemail(
          iv_fromemailaddress = iv_from_email_address
          io_destination = lo_destination
          io_content = lo_content ).
        MESSAGE 'Email sent successfully.' TYPE 'I'.
      CATCH /aws1/cx_se2accountsuspendedex INTO DATA(lo_account_suspended).
        MESSAGE 'Account suspended.' TYPE 'I'.
        RAISE EXCEPTION lo_account_suspended.
      CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
        MESSAGE 'Bad request.' TYPE 'I'.
        RAISE EXCEPTION lo_bad_request.
      CATCH /aws1/cx_se2messagerejected INTO DATA(lo_message_rejected).
        MESSAGE 'Message rejected - check email verification.' TYPE 'I'.
        RAISE EXCEPTION lo_message_rejected.
    ENDTRY.
```

Sends a message using a template.

```
    TRY.
        " Create destination with recipient address
        DATA lt_to_addresses TYPE /aws1/
cl_se2emailaddresslist_w=>tt_emailaddresslist.
        APPEND NEW /aws1/cl_se2emailaddresslist_w( iv_value =
 iv_to_email_address ) TO lt_to_addresses.
        DATA(lo_destination) = NEW /aws1/cl_se2destination(
          it_toaddresses = lt_to_addresses ).

        " Create template reference
        DATA(lo_template) = NEW /aws1/cl_se2template(
          iv_templatename = iv_template_name
          iv_templatedata = iv_template_data ).

        DATA(lo_content) = NEW /aws1/cl_se2emailcontent(
          io_template = lo_template ).

        " Create list management options
        DATA(lo_list_mgmt) = NEW /aws1/cl_se2listmanagementopts(
          iv_contactlistname = iv_contact_list_name ).
```

```
      " Send the email using template
      lo_se2->sendemail(
        iv_fromemailaddress = iv_from_email_address
        io_destination = lo_destination
        io_content = lo_content
        io_listmanagementoptions = lo_list_mgmt ).
      MESSAGE 'Email sent using template successfully.' TYPE 'I'.
    CATCH /aws1/cx_se2accountsuspendedex INTO DATA(lo_account_suspended).
      MESSAGE 'Account suspended.' TYPE 'I'.
      RAISE EXCEPTION lo_account_suspended.
    CATCH /aws1/cx_se2badrequestex INTO DATA(lo_bad_request).
      MESSAGE 'Bad request.' TYPE 'I'.
      RAISE EXCEPTION lo_bad_request.
    CATCH /aws1/cx_se2messagerejected INTO DATA(lo_message_rejected).
      MESSAGE 'Message rejected - check email verification.' TYPE 'I'.
      RAISE EXCEPTION lo_message_rejected.
  ENDTRY.
```

- For API details, see SendEmail in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon SES API v2 using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon SES API v2 with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon SES API v2 or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Send emails with attachments using Amazon SES API v2
- A complete Amazon SES API v2 Newsletter scenario using an AWS SDK

Send emails with attachments using Amazon SES API v2

The following code example shows how to send emails with attachments using Amazon SES API v2.

- Create an email template for bulk sends.
- Send a simple email with a file attachment.
- Send a simple email with an inline image.
- Send bulk templated emails with attachments.
- Clean up resources.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating email attachments.

```
class SESv2EmailAttachmentsScenario:
    """
    Demonstrates sending emails with attachments using Amazon SESv2.

    This scenario demonstrates:
    1. Setting up an email identity and template.
    2. Sending a simple email with a file attachment.
    3. Sending a simple email with an inline image.
    4. Sending bulk templated emails with attachments.
    5. Cleaning up created resources.
    """

    TEMPLATE_NAME = "AttachmentDemoTemplate"

    def __init__(self, sesv2_wrapper: SESv2Wrapper) -> None:
        """
        :param sesv2_wrapper: An instance of the SESv2Wrapper class.
```

```
        """
        self.sesv2_wrapper = sesv2_wrapper
        self.sender_email = ""
        self.recipient_emails: list = []
        self.identity_was_created = False

    def run_scenario(self) -> None:
        """Runs the SESv2 email attachments scenario."""
        print("-" * 88)
        print("Welcome to the Amazon SESv2 Email Attachments Scenario.")
        print("-" * 88)
        print(
            "This scenario demonstrates how to send emails with attachments\n"
            "using SESv2 attachment support. SES handles MIME\n"
            "construction automatically, so you don't need to build raw\n"
            "MIME messages.\n"
        )

        try:
            self._setup()
            self._step1_send_email_with_attachment()
            self._step2_send_email_with_inline_image()
            self._step3_send_bulk_email_with_attachments()
        except Exception as e:
            logger.error("Scenario failed: %s", e)
            print(f"\nThe scenario encountered an error: {e}")
        finally:
            self._cleanup()

    # ---------- Setup ----------

    def _setup(self) -> None:
        """
        Prompts for configuration, verifies the sender identity, prepares a
        sample attachment, and creates an email template.
        """
        print("\n--- Setup ---\n")

        # Prompt for sender and recipient addresses.
        print(
            "Both sender and recipient addresses must be verified if your\n"
            "account is in the SES sandbox.\n"
        )
        self.sender_email = q.ask(
```

```
            "Enter a verified sender email address: "
        )
        recipient_input = q.ask(
            "Enter one or more recipient email addresses (comma-separated): "
        )
        self.recipient_emails = [
            addr.strip() for addr in recipient_input.split(",") if addr.strip()
        ]

        # Verify the sender identity.
        print(f"\nChecking identity for {self.sender_email}...")
        try:
            identity_info = self.sesv2_wrapper.get_email_identity(
                self.sender_email
            )
            verified = identity_info.get("VerifiedForSendingStatus", False)
            if verified:
                print(f"  {self.sender_email} is verified and ready to send.")
            else:
                print(
                    f"  {self.sender_email} exists but is not yet verified."
                )
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                print(
                    f"  Identity {self.sender_email} not found. "
                    "Creating it now..."
                )
                result = self.sesv2_wrapper.create_email_identity(
                    self.sender_email
                )
                self.identity_was_created = True
                print(
                    f"  Identity created. Verification status: "
                    f"{result.get('VerifiedForSendingStatus', False)}"
                )
                print(
                    "  Check your inbox and click the verification link "
                    "before continuing."
                )
                q.ask("Press Enter when you have verified the address...")
            else:
                raise
```

```
        # Create the email template for the bulk-send step.
        print("\nCreating email template for the bulk email step...")
        try:
            self.sesv2_wrapper.create_email_template(
                template_name=self.TEMPLATE_NAME,
                subject="Bulk Email with Attachment for {{name}}",
                html_body=(
                    "<h1>Hello {{name}}</h1>"
                    "<p>Please find the attached document.</p>"
                ),
                text_body=(
                    "Hello {{name}}, Please find the attached document."
                ),
            )
            print(f"  Template '{self.TEMPLATE_NAME}' created.\n")
        except ClientError as err:
            if err.response["Error"]["Code"] == "AlreadyExistsException":
                print(
                    f"  Template '{self.TEMPLATE_NAME}' already exists. "
                    "Using it.\n"
                )
            else:
                raise

    # ---------- Step 1: Simple email with file attachment ----------

    def _step1_send_email_with_attachment(self) -> None:
        """Sends a simple email with a text file attachment."""
        print("\n--- Step 1: Send a Simple Email with a File Attachment ---\n")
        print(
            "Creating a sample text file attachment and sending it with\n"
            "the Simple email content type. SES constructs the MIME message\n"
            "automatically.\n"
        )

        # Prepare a sample text file as bytes.
        sample_content = b"This is a sample report attachment."

        attachment = {
            "RawContent": sample_content,
            "FileName": "sample-report.txt",
            "ContentType": "text/plain",
            "ContentDisposition": "ATTACHMENT",
            "ContentDescription": "Sample report text file",
```

```
            "ContentTransferEncoding": "BASE64",
        }

        print(
            "Note: When using an AWS SDK, the SDK handles base64 encoding\n"
            "automatically. Direct API callers must encode content themselves.\n"
        )

        message_id = self.sesv2_wrapper.send_email(
            from_address=self.sender_email,
            to_addresses=self.recipient_emails,
            subject="SESv2 Attachment Demo - Simple Email with Attachment",
            html_body=(
                "<h1>Attachment Demo</h1>"
                "<p>Please see the attached <b>report document</b>.</p>"
            ),
            text_body="Please see the attached report document.",
            attachments=[attachment],
        )

        print(f"  Email sent. MessageId: {message_id}")
        print(
            "  SES automatically constructed the MIME message with the "
            "attachment.\n"
        )

    # ---------- Step 2: Simple email with inline image ----------

    def _step2_send_email_with_inline_image(self) -> None:
        """Sends a simple email with an inline image that renders in HTML."""
        print("\n--- Step 2: Send a Simple Email with an Inline Image ---\n")
        print(
            "This step demonstrates INLINE disposition. The image renders\n"
            "directly in the HTML body using a 'cid:' reference instead of\n"
            "appearing as a downloadable attachment.\n"
        )

        # Create a minimal 1x1 red PNG (valid PNG file).
        sample_image = (
            b"\x89PNG\r\n\x1a\n"  # PNG signature
            b"\x00\x00\x00\rIHDR\x00\x00\x00\x01\x00\x00\x00\x01"
            b"\x08\x02\x00\x00\x00\x90wS\xde"  # 1x1 RGB
            b"\x00\x00\x00\x0cIDATx\x9cc\xf8\x0f\x00\x00\x01\x01"
            b"\x00\x05\x18\xd8N"  # compressed data
```

```
            b"\x00\x00\x00\x00IEND\xaeB`\x82"  # IEND
        )

        attachment = {
            "RawContent": sample_image,
            "FileName": "logo.png",
            "ContentType": "image/png",
            "ContentDisposition": "INLINE",
            "ContentId": "logo123",
            "ContentDescription": "Company logo",
            "ContentTransferEncoding": "BASE64",
        }

        html_body = (
            "<html><body>"
            "<h1>Inline Image Demo</h1>"
            "<p>Here is our logo:</p>"
            '<img src="cid:logo123" alt="Company Logo">'
            "</body></html>"
        )

        message_id = self.sesv2_wrapper.send_email(
            from_address=self.sender_email,
            to_addresses=self.recipient_emails,
            subject="SESv2 Attachment Demo - Inline Image",
            html_body=html_body,
            text_body=(
                "Inline Image Demo - Please view this email in an "
                "HTML-capable client to see the embedded image."
            ),
            attachments=[attachment],
        )

        print(f"  Email sent. MessageId: {message_id}")
        print(
            "  The ContentId 'logo123' is referenced in the HTML body via\n"
            "  'cid:logo123', which lets the image render inline.\n"
        )

    # ---------- Step 3: Bulk templated email with attachments ----------

    def _step3_send_bulk_email_with_attachments(self) -> None:
        """Sends bulk templated emails with attachments to multiple
recipients."""
```

```
        print("\n--- Step 3: Send Bulk Templated Emails with Attachments ---\n")
        print(
            "Using SendBulkEmail to send a templated email with an attachment\n"
            "to multiple recipients in a single API call. Each recipient gets\n"
            "personalized content via template data.\n"
        )

        sample_content = b"This is a sample report attachment."

        attachment = {
            "RawContent": sample_content,
            "FileName": "sample-report.txt",
            "ContentType": "text/plain",
            "ContentDisposition": "ATTACHMENT",
            "ContentDescription": "Sample report for bulk recipients",
            "ContentTransferEncoding": "BASE64",
        }

        # Build one entry per recipient with personalized names.
        names = ["Alice", "Bob", "Charlie", "Diana", "Eve"]
        bulk_entries = []
        for i, email in enumerate(self.recipient_emails):
            name = names[i] if i < len(names) else f"Recipient{i + 1}"
            bulk_entries.append(
                {
                    "Destination": {"ToAddresses": [email]},
                    "ReplacementEmailContent": {
                        "ReplacementTemplate": {
                            "ReplacementTemplateData": json.dumps(
                                {"name": name}
                            )
                        }
                    },
                }
            )

        results = self.sesv2_wrapper.send_bulk_email(
            from_address=self.sender_email,
            template_name=self.TEMPLATE_NAME,
            default_template_data='{"name": "Valued Customer"}',
            bulk_entries=bulk_entries,
            attachments=[attachment],
        )
```

```
        print("  Bulk email results:")
        for idx, result in enumerate(results):
            status = result.get("Status", "Unknown")
            msg_id = result.get("MessageId", "N/A")
            error = result.get("Error", "")
            recipient = (
                self.recipient_emails[idx]
                if idx < len(self.recipient_emails)
                else "Unknown"
            )
            print(f"    {recipient}: Status={status}, MessageId={msg_id}")
            if error:
                print(f"      Error: {error}")

        print(
            "\n  All recipients receive the same attachment(s) defined in\n"
            "  DefaultContent. Template data is personalized per recipient.\n"
        )

    # ---------- Cleanup ----------

    def _cleanup(self) -> None:
        """Deletes the email template and optionally the email identity."""
        print("\n--- Cleanup ---\n")

        # Delete the email template.
        try:
            self.sesv2_wrapper.delete_email_template(self.TEMPLATE_NAME)
            print(f"  Template '{self.TEMPLATE_NAME}' deleted.")
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                print(
                    f"  Template '{self.TEMPLATE_NAME}' was already deleted."
                )
            else:
                logger.error("Failed to delete template: %s", err)

        # Optionally delete the email identity.
        if self.identity_was_created and self.sender_email:
            delete_identity = q.ask(
                f"Delete the email identity '{self.sender_email}'? (y/n) ",
                q.is_yesno,
            )
            if delete_identity:
```

```
                try:
                    self.sesv2_wrapper.delete_email_identity(
                        self.sender_email
                    )
                    print(
                        f"  Email identity '{self.sender_email}' deleted."
                    )
                except ClientError as err:
                    if err.response["Error"]["Code"] == "NotFoundException":
                        print(
                            f"  Identity '{self.sender_email}' was "
                            "already deleted."
                        )
                    else:
                        logger.error(
                            "Failed to delete identity: %s", err
                        )
            else:
                print(
                    f"  Skipping identity deletion for {self.sender_email}."
                )
        else:
            print(
                "  Sender identity was pre-existing. Skipping deletion."
            )

        print("\nAll resources have been cleaned up.")
        print("-" * 88)
```

Create an SESv2 wrapper class to manage operations.

```
class SESv2Wrapper:
    """Encapsulates Amazon SESv2 email sending actions."""

    def __init__(self, sesv2_client: Any) -> None:
        """
        Initializes the SESv2Wrapper with an SESv2 client.

        :param sesv2_client: A Boto3 SESv2 client.
        """
```

```
        self.sesv2_client = sesv2_client

    @classmethod
    def from_client(cls) -> "SESv2Wrapper":
        """
        Creates an SESv2Wrapper instance with a default Boto3 SESv2 client.

        :return: A new SESv2Wrapper instance.
        """
        sesv2_client = boto3.client("sesv2")
        return cls(sesv2_client)


    def get_email_identity(self, email_address: str) -> Dict[str, Any]:
        """
        Gets information about an email identity, including its verification
status.

        :param email_address: The email address or domain to look up.
        :return: A dictionary with identity information including verification
status.
        :raises ClientError: If the identity is not found (NotFoundException).
        """
        try:
            response = self.sesv2_client.get_email_identity(
                EmailIdentity=email_address
            )
            logger.info("Got email identity for %s.", email_address)
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email identity %s not found.", email_address
                )
            else:
                logger.error(
                    "Couldn't get email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

```
    def create_email_identity(self, email_address: str) -> Dict[str, Any]:
        """
        Starts the process of verifying an email identity (email address or
domain).

        :param email_address: The email address or domain to verify.
        :return: A dictionary with the identity type and verification status.
        :raises ClientError: If the limit is exceeded (LimitExceededException).
        """
        try:
            response = self.sesv2_client.create_email_identity(
                EmailIdentity=email_address
            )
            logger.info(
                "Started verification for email identity %s.", email_address
            )
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "LimitExceededException":
                logger.error(
                    "Couldn't create email identity %s. You have exceeded "
                    "the maximum number of email identities. "
                    "Use an existing verified identity.",
                    email_address,
                )
            else:
                logger.error(
                    "Couldn't create email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def create_email_template(
        self,
        template_name: str,
        subject: str,
        html_body: str,
        text_body: str,
    ) -> None:
        """
        Creates an email template for use with templated and bulk email sends.
```

```
        :param template_name: The name for the new template.
        :param subject: The subject line of the template. May include
{{placeholders}}.
        :param html_body: The HTML body of the template.
        :param text_body: The plain text body of the template.
        :raises ClientError: If the template limit is exceeded
(LimitExceededException).
        """
        try:
            self.sesv2_client.create_email_template(
                TemplateName=template_name,
                TemplateContent={
                    "Subject": subject,
                    "Html": html_body,
                    "Text": text_body,
                },
            )
            logger.info("Created email template %s.", template_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "LimitExceededException":
                logger.error(
                    "Couldn't create email template %s. You have exceeded "
                    "the maximum number of email templates. "
                    "Delete unused templates first.",
                    template_name,
                )
            else:
                logger.error(
                    "Couldn't create email template %s. Here's why: %s: %s",
                    template_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def send_email(
        self,
        from_address: str,
        to_addresses: List[str],
        subject: str,
        html_body: str,
        text_body: str,
```

```
        attachments: Optional[List[Dict[str, Any]]] = None,
    ) -> str:
        """
        Sends a simple email message with optional attachments.

        SES handles MIME construction automatically when using attachments
        with the Simple content type, so developers don't need to build
        raw MIME messages.

        :param from_address: The verified sender email address.
        :param to_addresses: A list of recipient email addresses.
        :param subject: The subject line of the email.
        :param html_body: The HTML body content.
        :param text_body: The plain text body content.
        :param attachments: An optional list of attachment dictionaries. Each
            attachment should contain 'RawContent' (bytes), 'FileName' (str),
            and optionally 'ContentType', 'ContentDisposition', 'ContentId',
            'ContentDescription', and 'ContentTransferEncoding'.
        :return: The MessageId of the sent email.
        :raises ClientError: If the message is rejected (MessageRejected).
        """
        try:
            simple_message: Dict[str, Any] = {
                "Subject": {"Data": subject},
                "Body": {
                    "Html": {"Data": html_body},
                    "Text": {"Data": text_body},
                },
            }

            if attachments:
                simple_message["Attachments"] = attachments

            response = self.sesv2_client.send_email(
                FromEmailAddress=from_address,
                Destination={"ToAddresses": to_addresses},
                Content={"Simple": simple_message},
            )
            message_id = response["MessageId"]
            logger.info(
                "Sent email from %s to %s. MessageId: %s",
                from_address,
                to_addresses,
                message_id,
```

```
            )
            return message_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "MessageRejected":
                logger.error(
                    "Message was rejected. Check that attachments use "
                    "supported file types and total message size is "
                    "under 40 MB. Details: %s",
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't send email. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def send_bulk_email(
        self,
        from_address: str,
        template_name: str,
        default_template_data: str,
        bulk_entries: List[Dict[str, Any]],
        attachments: Optional[List[Dict[str, Any]]] = None,
    ) -> List[Dict[str, Any]]:
        """
        Sends a templated email to multiple recipients in a single API call.

        All recipients receive the same attachment(s) defined in the default
        content, while template data can be personalized per recipient.

        :param from_address: The verified sender email address.
        :param template_name: The name of an existing email template.
        :param default_template_data: Default JSON template data string.
        :param bulk_entries: A list of BulkEmailEntry dicts, each containing
            'Destination' and optionally 'ReplacementEmailContent'.
        :param attachments: An optional list of attachment dicts for all
            recipients.
        :return: A list of BulkEmailEntryResult dicts with status and MessageId.
        :raises ClientError: If the message is rejected (MessageRejected).
        """
        try:
```

```
            template_content: Dict[str, Any] = {
                "TemplateName": template_name,
                "TemplateData": default_template_data,
            }

            if attachments:
                template_content["Attachments"] = attachments

            response = self.sesv2_client.send_bulk_email(
                FromEmailAddress=from_address,
                DefaultContent={"Template": template_content},
                BulkEmailEntries=bulk_entries,
            )
            results = response.get("BulkEmailEntryResults", [])
            logger.info(
                "Sent bulk email from %s to %d recipients.",
                from_address,
                len(bulk_entries),
            )
            return results
        except ClientError as err:
            if err.response["Error"]["Code"] == "MessageRejected":
                logger.error(
                    "Bulk message was rejected. Check that the template "
                    "exists, attachment file types are supported, and "
                    "total message size is within limits. Details: %s",
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't send bulk email. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def delete_email_template(self, template_name: str) -> None:
        """
        Deletes an email template.

        :param template_name: The name of the template to delete.
        :raises ClientError: If the template is not found (NotFoundException).
        """
```

```
        try:
            self.sesv2_client.delete_email_template(
                TemplateName=template_name
            )
            logger.info("Deleted email template %s.", template_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email template %s not found or already deleted.",
                    template_name,
                )
            else:
                logger.error(
                    "Couldn't delete email template %s. Here's why: %s: %s",
                    template_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def delete_email_identity(self, email_address: str) -> None:
        """
        Deletes an email identity.

        :param email_address: The email address or domain to delete.
        :raises ClientError: If the identity is not found (NotFoundException).
        """
        try:
            self.sesv2_client.delete_email_identity(
                EmailIdentity=email_address
            )
            logger.info("Deleted email identity %s.", email_address)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotFoundException":
                logger.info(
                    "Email identity %s not found or already deleted.",
                    email_address,
                )
            else:
                logger.error(
                    "Couldn't delete email identity %s. Here's why: %s: %s",
                    email_address,
                    err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
        raise
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteEmailIdentity
  - DeleteEmailTemplate
  - GetEmailIdentity
  - SendBulkEmail
  - SendEmail

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

A complete Amazon SES API v2 Newsletter scenario using an AWS SDK

The following code examples show how to run the Amazon SES API v2 newsletter scenario.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the scenario.

```
using System.Diagnostics;
```

```
using System.Text.RegularExpressions;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace Sesv2Scenario;

public static class NewsletterWorkflow
{
    /*
      This scenario demonstrates how to use the Amazon Simple Email Service (SES)
 v2 to send a coupon newsletter to a list of subscribers.
      The scenario performs the following tasks:

      1. Prepare the application:
         - Create a verified email identity for sending and replying to emails.
         - Create a contact list to store the subscribers' email addresses.
         - Create an email template for the coupon newsletter.

      2. Gather subscriber email addresses:
         - Prompt the user for a base email address.
         - Create 3 variants of the email address using subaddress extensions
 (e.g., user+ses-weekly-newsletter-1@example.com).
         - Add each variant as a contact to the contact list.
         - Send a welcome email to each new contact.

      3. Send the coupon newsletter:
         - Retrieve the list of contacts from the contact list.
         - Send the coupon newsletter using the email template to each contact.

      4. Monitor and review:
         - Provide instructions for the user to review the sending activity and
 metrics in the AWS console.

      5. Clean up resources:
         - Delete the contact list (which also deletes all contacts within it).
         - Delete the email template.
         - Optionally delete the verified email identity.

    */
```

```
    public static SESv2Wrapper _sesv2Wrapper;
    public static string? _baseEmailAddress = null;
    public static string? _verifiedEmail = null;
    private static string _contactListName = "weekly-coupons-newsletter";
    private static string _templateName = "weekly-coupons";
    private static string _subject = "Weekly Coupons Newsletter";
    private static string _htmlContentFile = "coupon-newsletter.html";
    private static string _textContentFile = "coupon-newsletter.txt";
    private static string _htmlWelcomeFile = "welcome.html";
    private static string _textWelcomeFile = "welcome.txt";
    private static string _couponsDataFile = "sample_coupons.json";

    // Relative location of the resources folder.
    private static string _resourcesFilePathLocation = "../../../../resources/";

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSimpleEmailServiceV2>()
                    .AddTransient<SESv2Wrapper>()
            )
            .Build();

        ServicesSetup(host);

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Amazon SES v2 Coupon Newsletter
Scenario.");
            Console.WriteLine("This scenario demonstrates how to use the Amazon
Simple Email Service (SES) v2 " +
                              "\r\nto send a coupon newsletter to a list of
subscribers.");
```

```
            // Prepare the application.
            var emailIdentity = await PrepareApplication();

            // Gather subscriber email addresses.
            await GatherSubscriberEmailAddresses(emailIdentity);

            // Send the coupon newsletter.
            await SendCouponNewsletter(emailIdentity);

            // Monitor and review.
            MonitorAndReview(true);

            // Clean up resources.
            await Cleanup(emailIdentity, true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon SES v2 Coupon Newsletter scenario is
complete.");
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _sesv2Wrapper = host.Services.GetRequiredService<SESv2Wrapper>();
    }

    /// <summary>
    /// Set up the resources for the scenario.
    /// </summary>
    /// <returns>The email address of the verified identity.</returns>
    public static async Task<string?> PrepareApplication()
    {
```

```
        var htmlContent = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _htmlContentFile);
        var textContent = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _textContentFile);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("1. In this step, we will prepare the application:" +
                          "\r\n  - Create a verified email identity for sending
and replying to emails." +
                          "\r\n  - Create a contact list to store the
subscribers' email addresses." +
                          "\r\n  - Create an email template for the coupon
newsletter.\r\n");

        // Prompt the user for a verified email address.
        while (!IsEmail(_verifiedEmail))
        {
            Console.Write("Enter a verified email address or an email to verify:
");
            _verifiedEmail = Console.ReadLine();
        }

        try
        {
            // Create an email identity and start the verification process.
            await _sesv2Wrapper.CreateEmailIdentityAsync(_verifiedEmail);
            Console.WriteLine($"Identity {_verifiedEmail} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Identity {_verifiedEmail} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating email identity: {ex.Message}");
        }

        // Create a contact list.
        try
        {
            await _sesv2Wrapper.CreateContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} created.");
        }
        catch (AlreadyExistsException)
```

```
        {
            Console.WriteLine($"Contact list {_contactListName} already
exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating contact list: {ex.Message}");
        }

        // Create an email template.
        try
        {
            await _sesv2Wrapper.CreateEmailTemplateAsync(_templateName, _subject,
htmlContent, textContent);
            Console.WriteLine($"Email template {_templateName} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Email template {_templateName} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating email template: {ex.Message}");
        }

        return _verifiedEmail;
    }

    /// <summary>
    /// Generate subscriber addresses and send welcome emails.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> GatherSubscriberEmailAddresses(string
fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("2. In Step 2, we will gather subscriber email
addresses:" +
                          "\r\n  - Prompt the user for a base email address." +
                          "\r\n  - Create 3 variants of the email address using
subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com)." +
```

```
                            "\r\n  - Add each variant as a contact to the contact
 list." +
                            "\r\n  - Send a welcome email to each new contact.\r
\n");

        // Prompt the user for a base email address.
        while (!IsEmail(_baseEmailAddress))
        {
            Console.Write("Enter a base email address (e.g., user@example.com):
 ");
            _baseEmailAddress = Console.ReadLine();
        }

        // Create 3 variants of the email address using +ses-weekly-newsletter-1,
 +ses-weekly-newsletter-2, etc.
        var baseEmailAddressParts = _baseEmailAddress!.Split("@");
        for (int i = 1; i <= 3; i++)
        {
            string emailAddress = $"{baseEmailAddressParts[0]}+ses-weekly-
newsletter-{i}@{baseEmailAddressParts[1]}";

            try
            {
                // Create a contact with the email address in the contact list.
                await _sesv2Wrapper.CreateContactAsync(emailAddress,
 _contactListName);
                Console.WriteLine($"Contact {emailAddress} added to the
 {_contactListName} contact list.");
            }
            catch (AlreadyExistsException)
            {
                Console.WriteLine($"Contact {emailAddress} already exists in the
 {_contactListName} contact list.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error creating contact {emailAddress}:
 {ex.Message}");
                return false;
            }

            // Send a welcome email to the new contact.
            try
            {
```

```
                string subject = "Welcome to the Weekly Coupons Newsletter";
                string htmlContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _htmlWelcomeFile);
                string textContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _textWelcomeFile);

                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress, new
List<string> { emailAddress }, subject, htmlContent, textContent);
                Console.WriteLine($"Welcome email sent to {emailAddress}.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error sending welcome email to
{emailAddress}: {ex.Message}");
                return false;
            }

            // Wait 2 seconds before sending the next email (if the account is in
the SES Sandbox).
            await Task.Delay(2000);
        }

        return true;
    }

    /// <summary>
    ///  Send the coupon newsletter to the subscribers in the contact list.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> SendCouponNewsletter(string fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("3. In this step, we will send the coupon newsletter:"
+
                          "\r\n  - Retrieve the list of contacts from the contact
list." +
                          "\r\n  - Send the coupon newsletter using the email
template to each contact.\r\n");


        // Retrieve the list of contacts from the contact list.
        var contacts = await _sesv2Wrapper.ListContactsAsync(_contactListName);
```

```
        if (!contacts.Any())
        {
            Console.WriteLine($"No contacts found in the {_contactListName}
contact list.");
            return false;
        }

        // Load the coupon data from the sample_coupons.json file.
        string couponsData = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _couponsDataFile);

        // Send the coupon newsletter to each contact using the email template.
        try
        {
            foreach (var contact in contacts)
            {
                // To use the Contact List for list management, send to only one
address at a time.
                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress,
                    new List<string> { contact.EmailAddress },
                    null, null, null, _templateName, couponsData,
_contactListName);
            }

            Console.WriteLine($"Coupon newsletter sent to contact list
{_contactListName}.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error sending coupon newsletter to contact list
{_contactListName}: {ex.Message}");
            return false;
        }

        return true;
    }

    /// <summary>
    /// Provide instructions for monitoring sending activity and metrics.
    /// </summary>
    /// <param name="interactive">True to run in interactive mode.</param>
    /// <returns>True if successful.</returns>
    public static bool MonitorAndReview(bool interactive)
    {
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. In step 4, we will monitor and review:" +
                          "\r\n  - Provide instructions for the user to review
the sending activity and metrics in the AWS console.\r\n");

        Console.WriteLine("Review your sending activity using the SES Homepage in
the AWS console.");
        Console.WriteLine("Press Enter to open the SES Homepage in your default
browser...");
        if (interactive)
        {
            Console.ReadLine();
            try
            {
                // Open the SES Homepage in the default browser.
                Process.Start(new ProcessStartInfo
                {
                    FileName = "https://console.aws.amazon.com/ses/home",
                    UseShellExecute = true
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error opening the SES Homepage:
{ex.Message}");
                return false;
            }
        }

        Console.WriteLine("Review the sending activity and email metrics, then
press Enter to continue...");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Clean up the resources used in the scenario.
    /// </summary>
    /// <param name="verifiedEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <param name="interactive">True if interactive.</param>
    /// <returns>Async task.</returns>
```

```
    public static async Task<bool> Cleanup(string verifiedEmailAddress, bool
 interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Finally, we clean up resources:" +
                          "\r\n  - Delete the contact list (which also deletes
 all contacts within it)." +
                          "\r\n  - Delete the email template." +
                          "\r\n  - Optionally delete the verified email identity.
\r\n");

        Console.WriteLine("Cleaning up resources...");

        // Delete the contact list (this also deletes all contacts in the list).
        try
        {
            await _sesv2Wrapper.DeleteContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Contact list {_contactListName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting contact list {_contactListName}:
 {ex.Message}");
            return false;
        }

        // Delete the email template.
        try
        {
            await _sesv2Wrapper.DeleteEmailTemplateAsync(_templateName);
            Console.WriteLine($"Email template {_templateName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Email template {_templateName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting email template {_templateName}:
 {ex.Message}");
```

```
            return false;
        }

        // Ask the user if they want to delete the email identity.
        var deleteIdentity = !interactive ||
            GetYesNoResponse(
                $"Do you want to delete the email identity
 {verifiedEmailAddress}? (y/n) ");
        if (deleteIdentity)
        {
            try
            {
                await
 _sesv2Wrapper.DeleteEmailIdentityAsync(verifiedEmailAddress);
                Console.WriteLine($"Email identity {verifiedEmailAddress}
 deleted.");
            }
            catch (NotFoundException)
            {
                Console.WriteLine(
                    $"Email identity {verifiedEmailAddress} not found.");
            }
            catch (Exception ex)
            {
                Console.WriteLine(
                    $"Error deleting email identity {verifiedEmailAddress}:
 {ex.Message}");
                return false;
            }
        }
        else
        {
            Console.WriteLine(
                $"Skipping deletion of email identity {verifiedEmailAddress}.");
        }

        return true;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
```

```
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Simple check to verify a string is an email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email.</returns>
    private static bool IsEmail(string? email)
    {
        if (string.IsNullOrEmpty(email))
            return false;
        return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$",
 RegexOptions.IgnoreCase);
    }
}
```

Wrapper for service operations.

```
using System.Net;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;

namespace Sesv2Scenario;

/// <summary>
/// Wrapper class for Amazon Simple Email Service (SES) v2 operations.
/// </summary>
public class SESv2Wrapper
{

    private readonly IAmazonSimpleEmailServiceV2 _sesClient;

    /// <summary>
    /// Constructor for the SESv2Wrapper.
```

```
    /// </summary>
    /// <param name="sesClient">The injected SES v2 client.</param>
    public SESv2Wrapper(IAmazonSimpleEmailServiceV2 sesClient)
    {
        _sesClient = sesClient;
    }

    /// <summary>
    /// Creates a contact and adds it to the specified contact list.
    /// </summary>
    /// <param name="emailAddress">The email address of the contact.</param>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The response from the CreateContact operation.</returns>
    public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
    {
        var request = new CreateContactRequest
        {
            EmailAddress = emailAddress,
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
```

```
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact
list: {ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates an email identity (email address or domain) and starts the
verification process.
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
    /// <returns>The response from the CreateEmailIdentity operation.</returns>
    public async Task<CreateEmailIdentityResponse>
CreateEmailIdentityAsync(string emailIdentity)
    {
        var request = new CreateEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.CreateEmailIdentityAsync(request);
            return response;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email identity {emailIdentity} already exists.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being
modified by another operation or thread.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (LimitExceededException ex)
        {
```

```
            Console.WriteLine("The limit for email identities has been
exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email
identity: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
```

```
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
 exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been
 exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email
 template: {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteContactListAsync(string contactListName)
```

```
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being
 modified by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact
 list: {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</
param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being
 modified by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email
 identity: {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</
param>
```

```
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email
 template: {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
```

```
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
        }

        return new List<Contact>();
    }

    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email
from.</param>
    /// <param name="toEmailAddresses">The email addresses to send the email
to.</param>
    /// <param name="subject">The subject of the email.</param>
    /// <param name="htmlContent">The HTML content of the email.</param>
    /// <param name="textContent">The text content of the email.</param>
    /// <param name="templateName">The name of the email template to use
(optional).</param>
    /// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
```

```
   /// <param name="contactListName">The name of the contact list for
unsubscribe functionality (optional).</param>
   /// <returns>The MessageId response from the SendEmail operation.</returns>
   public async Task<string> SendEmailAsync(string fromEmailAddress,
List<string> toEmailAddresses, string? subject,
       string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
   {
       var request = new SendEmailRequest
       {
           FromEmailAddress = fromEmailAddress
       };

       if (toEmailAddresses.Any())
       {
           request.Destination = new Destination { ToAddresses =
toEmailAddresses };
       }

       if (!string.IsNullOrEmpty(templateName))
       {
           request.Content = new EmailContent()
           {
               Template = new Template
               {
                   TemplateName = templateName,
                   TemplateData = templateData
               }
           };
       }
       else
       {
           request.Content = new EmailContent
           {
               Simple = new Message
               {
                   Subject = new Content { Data = subject },
                   Body = new Body
                   {
                       Html = new Content { Data = htmlContent },
                       Text = new Content { Data = textContent }
                   }
               }
           };
```

```
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
        }
        catch (MailFromDomainNotVerifiedException ex)
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"An error occurred while sending the email:
 {ex.Message}");
        }

        return string.Empty;
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList
  - DeleteEmailIdentity
  - DeleteEmailTemplate
  - ListContacts
  - SendEmail.simple
  - SendEmail.template

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    try {
      // 2. Create a contact list
      String contactListName = CONTACT_LIST_NAME;
      CreateContactListRequest createContactListRequest =
CreateContactListRequest.builder()
```

```
            .contactListName(contactListName)
            .build();
        sesClient.createContactList(createContactListRequest);
        System.out.println("Contact list created: " + contactListName);
      } catch (AlreadyExistsException e) {
        System.out.println("Contact list already exists, skipping creation: weekly-
coupons-newsletter");
      } catch (LimitExceededException e) {
        System.err.println("Limit for contact lists has been exceeded.");
        throw e;
      } catch (SesV2Exception e) {
        System.err.println("Error creating contact list: " + e.getMessage());
        throw e;
      }

        try {
          // Create a new contact with the provided email address in the
          CreateContactRequest contactRequest = CreateContactRequest.builder()
              .contactListName(CONTACT_LIST_NAME)
              .emailAddress(emailAddress)
              .build();

          sesClient.createContact(contactRequest);
          contacts.add(emailAddress);

          System.out.println("Contact created: " + emailAddress);

          // Send a welcome email to the new contact
          String welcomeHtml = Files.readString(Paths.get("resources/
coupon_newsletter/welcome.html"));
          String welcomeText = Files.readString(Paths.get("resources/
coupon_newsletter/welcome.txt"));

          SendEmailRequest welcomeEmailRequest = SendEmailRequest.builder()
              .fromEmailAddress(this.verifiedEmail)
              .destination(Destination.builder().toAddresses(emailAddress).build())
              .content(EmailContent.builder()
                  .simple(
                      Message.builder()
                          .subject(Content.builder().data("Welcome to the Weekly
 Coupons Newsletter").build())
                          .body(Body.builder()
                              .text(Content.builder().data(welcomeText).build())
                              .html(Content.builder().data(welcomeHtml).build())
```

```
                            .build())
                        .build())
                .build())
            .build();
        SendEmailResponse welcomeEmailResponse =
 sesClient.sendEmail(welcomeEmailRequest);
        System.out.println("Welcome email sent: " +
 welcomeEmailResponse.messageId());
      } catch (AlreadyExistsException e) {
        // If the contact already exists, skip this step for that contact and
 proceed
        // with the next contact
        System.out.println("Contact already exists, skipping creation...");
      } catch (Exception e) {
        System.err.println("Error occurred while processing email address " +
 emailAddress + ": " + e.getMessage());
        throw e;
      }
    }

      ListContactsRequest contactListRequest = ListContactsRequest.builder()
          .contactListName(CONTACT_LIST_NAME)
          .build();

      List<String> contactEmails;
      try {
        ListContactsResponse contactListResponse =
 sesClient.listContacts(contactListRequest);

        contactEmails = contactListResponse.contacts().stream()
            .map(Contact::emailAddress)
            .toList();
      } catch (Exception e) {
        // TODO: Remove when listContacts's GET body issue is resolved.
        contactEmails = this.contacts;
      }


      String coupons = Files.readString(Paths.get("resources/coupon_newsletter/
sample_coupons.json"));
      for (String emailAddress : contactEmails) {
        SendEmailRequest newsletterRequest = SendEmailRequest.builder()
            .destination(Destination.builder().toAddresses(emailAddress).build())
            .content(EmailContent.builder()
```

```
                .template(Template.builder()
                    .templateName(TEMPLATE_NAME)
                    .templateData(coupons)
                    .build())
                .build())
            .fromEmailAddress(this.verifiedEmail)
            .listManagementOptions(ListManagementOptions.builder()
                .contactListName(CONTACT_LIST_NAME)
                .build())
            .build();
        SendEmailResponse newsletterResponse =
 sesClient.sendEmail(newsletterRequest);
        System.out.println("Newsletter sent to " + emailAddress + ": " +
 newsletterResponse.messageId());
     }

   try {
     CreateEmailIdentityRequest createEmailIdentityRequest =
 CreateEmailIdentityRequest.builder()
         .emailIdentity(verifiedEmail)
         .build();
     sesClient.createEmailIdentity(createEmailIdentityRequest);
     System.out.println("Email identity created: " + verifiedEmail);
   } catch (AlreadyExistsException e) {
     System.out.println("Email identity already exists, skipping creation: " +
 verifiedEmail);
   } catch (NotFoundException e) {
     System.err.println("The provided email address is not verified: " +
 verifiedEmail);
     throw e;
   } catch (LimitExceededException e) {
     System.err
         .println("You have reached the limit for email identities. Please
 remove some identities and try again.");
     throw e;
   } catch (SesV2Exception e) {
     System.err.println("Error creating email identity: " + e.getMessage());
     throw e;
   }

   try {
     // Create an email template named "weekly-coupons"
     String newsletterHtml = loadFile("resources/coupon_newsletter/coupon-
newsletter.html");
```

```
      String newsletterText = loadFile("resources/coupon_newsletter/coupon-
newsletter.txt");

      CreateEmailTemplateRequest templateRequest =
 CreateEmailTemplateRequest.builder()
          .templateName(TEMPLATE_NAME)
          .templateContent(EmailTemplateContent.builder()
              .subject("Weekly Coupons Newsletter")
              .html(newsletterHtml)
              .text(newsletterText)
              .build())
          .build();

      sesClient.createEmailTemplate(templateRequest);

      System.out.println("Email template created: " + TEMPLATE_NAME);
    } catch (AlreadyExistsException e) {
      // If the template already exists, skip this step and proceed with the next
      // operation
      System.out.println("Email template already exists, skipping creation...");
    } catch (LimitExceededException e) {
      // If the limit for email templates is exceeded, fail the workflow and
 inform
      // the user
      System.err.println("You have reached the limit for email templates. Please
 remove some templates and try again.");
      throw e;
    } catch (Exception e) {
      System.err.println("Error occurred while creating email template: " +
e.getMessage());
      throw e;
    }

    try {
      // Delete the contact list
      DeleteContactListRequest deleteContactListRequest =
 DeleteContactListRequest.builder()
          .contactListName(CONTACT_LIST_NAME)
          .build();

      sesClient.deleteContactList(deleteContactListRequest);

      System.out.println("Contact list deleted: " + CONTACT_LIST_NAME);
    } catch (NotFoundException e) {
```

```
      // If the contact list does not exist, log the error and proceed
      System.out.println("Contact list not found. Skipping deletion...");
    } catch (Exception e) {
      System.err.println("Error occurred while deleting the contact list: " +
e.getMessage());
      e.printStackTrace();
    }

      try {
        // Delete the email identity
        DeleteEmailIdentityRequest deleteIdentityRequest =
DeleteEmailIdentityRequest.builder()
            .emailIdentity(this.verifiedEmail)
            .build();

        sesClient.deleteEmailIdentity(deleteIdentityRequest);

        System.out.println("Email identity deleted: " + this.verifiedEmail);
      } catch (NotFoundException e) {
        // If the email identity does not exist, log the error and proceed
        System.out.println("Email identity not found. Skipping deletion...");
      } catch (Exception e) {
        System.err.println("Error occurred while deleting the email identity: " +
e.getMessage());
        e.printStackTrace();
      }
    } else {
      System.out.println("Skipping email identity deletion.");
    }

    try {
      // Delete the template
      DeleteEmailTemplateRequest deleteTemplateRequest =
DeleteEmailTemplateRequest.builder()
          .templateName(TEMPLATE_NAME)
          .build();

      sesClient.deleteEmailTemplate(deleteTemplateRequest);

      System.out.println("Email template deleted: " + TEMPLATE_NAME);
    } catch (NotFoundException e) {
      // If the email template does not exist, log the error and proceed
      System.out.println("Email template not found. Skipping deletion...");
    } catch (Exception e) {
```

```
      System.err.println("Error occurred while deleting the email template: " +
 e.getMessage());
      e.printStackTrace();
    }
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList
  - DeleteEmailIdentity
  - DeleteEmailTemplate
  - ListContacts
  - SendEmail.simple
  - SendEmail.template

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def main():
    """
    The main function that orchestrates the execution of the workflow.
    """
    print(INTRO)
    ses_client = boto3.client("sesv2")
    workflow = SESv2Workflow(ses_client)
    try:
        workflow.prepare_application()
```

```
        workflow.gather_subscriber_email_addresses()
        workflow.send_coupon_newsletter()
        workflow.monitor_and_review()
    except ClientError as e:
        print_error(e)
    workflow.clean_up()



class SESv2Workflow:
    """
    A class to manage the SES v2 Coupon Newsletter Workflow.
    """

    def __init__(self, ses_client, sleep=True):
        self.ses_client = ses_client
        self.sleep = sleep


        try:

 self.ses_client.create_contact_list(ContactListName=CONTACT_LIST_NAME)
            print(f"Contact list '{CONTACT_LIST_NAME}' created successfully.")
        except ClientError as e:
            # If the contact list already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
                print(f"Contact list '{CONTACT_LIST_NAME}' already exists.")
            else:
                raise e

            try:
                # Create a new contact
                self.ses_client.create_contact(
                    ContactListName=CONTACT_LIST_NAME, EmailAddress=email
                )
                print(f"Contact with email '{email}' created successfully.")

                # Send the welcome email
                self.ses_client.send_email(
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email]},
                    Content={
                        "Simple": {
                            "Subject": {
```

```
                                "Data": "Welcome to the Weekly Coupons
Newsletter"
                            },
                            "Body": {
                                "Text": {"Data": welcome_text},
                                "Html": {"Data": welcome_html},
                            },
                        }
                    },
                )
                print(f"Welcome email sent to '{email}'.")
                if self.sleep:
                    # 1 email per second in sandbox mode, remove in production.
                    sleep(1.1)
            except ClientError as e:
                # If the contact already exists, skip and proceed
                if e.response["Error"]["Code"] == "AlreadyExistsException":
                    print(f"Contact with email '{email}' already exists.
Skipping...")
                else:
                    raise e

        try:
            contacts_response = self.ses_client.list_contacts(
                ContactListName=CONTACT_LIST_NAME
            )
        except ClientError as e:
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Contact list '{CONTACT_LIST_NAME}' does not exist.")
                return
            else:
                raise e

                self.ses_client.send_email(
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email]},
                    Content={
                        "Simple": {
                            "Subject": {
                                "Data": "Welcome to the Weekly Coupons
Newsletter"
                            },
                            "Body": {
                                "Text": {"Data": welcome_text},
```

```
                                "Html": {"Data": welcome_html},
                            },
                        }
                    },
                )
                print(f"Welcome email sent to '{email}'.")

                self.ses_client.send_email(
                    FromEmailAddress=self.verified_email,
                    Destination={"ToAddresses": [email_address]},
                    Content={
                        "Template": {
                            "TemplateName": TEMPLATE_NAME,
                            "TemplateData": coupon_items,
                        }
                    },
                    ListManagementOptions={"ContactListName": CONTACT_LIST_NAME},
                )

        try:

self.ses_client.create_email_identity(EmailIdentity=self.verified_email)
            print(f"Email identity '{self.verified_email}' created
successfully.")
        except ClientError as e:
            # If the email identity already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
                print(f"Email identity '{self.verified_email}' already exists.")
            else:
                raise e

        try:
            template_content = {
                "Subject": "Weekly Coupons Newsletter",
                "Html": load_file_content("coupon-newsletter.html"),
                "Text": load_file_content("coupon-newsletter.txt"),
            }
            self.ses_client.create_email_template(
                TemplateName=TEMPLATE_NAME, TemplateContent=template_content
            )
            print(f"Email template '{TEMPLATE_NAME}' created successfully.")
        except ClientError as e:
            # If the template already exists, skip and proceed
            if e.response["Error"]["Code"] == "AlreadyExistsException":
```

```
                print(f"Email template '{TEMPLATE_NAME}' already exists.")
            else:
                raise e

        try:

self.ses_client.delete_contact_list(ContactListName=CONTACT_LIST_NAME)
            print(f"Contact list '{CONTACT_LIST_NAME}' deleted successfully.")
        except ClientError as e:
            # If the contact list doesn't exist, skip and proceed
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Contact list '{CONTACT_LIST_NAME}' does not exist.")
            else:
                print(e)

            try:

self.ses_client.delete_email_identity(EmailIdentity=self.verified_email)
                print(f"Email identity '{self.verified_email}' deleted
successfully.")
            except ClientError as e:
                # If the email identity doesn't exist, skip and proceed
                if e.response["Error"]["Code"] == "NotFoundException":
                    print(f"Email identity '{self.verified_email}' does not
exist.")
                else:
                    print(e)

        try:
            self.ses_client.delete_email_template(TemplateName=TEMPLATE_NAME)
            print(f"Email template '{TEMPLATE_NAME}' deleted successfully.")
        except ClientError as e:
            # If the email template doesn't exist, skip and proceed
            if e.response["Error"]["Code"] == "NotFoundException":
                print(f"Email template '{TEMPLATE_NAME}' does not exist.")
            else:
                print(e)
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateContact
  - CreateContactList

- CreateEmailIdentity
- CreateEmailTemplate
- DeleteContactList
- DeleteEmailIdentity
- DeleteEmailTemplate
- ListContacts
- SendEmail.simple
- SendEmail.template

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        match self
            .client
            .create_contact_list()
            .contact_list_name(CONTACT_LIST_NAME)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Contact list created
successfully.")?,
            Err(e) => match e.into_service_error() {
                CreateContactListError::AlreadyExistsException(_) => {
                    writeln!(
                        self.stdout,
                        "Contact list already exists, skipping creation."
                    )?;
                }
                e => return Err(anyhow!("Error creating contact list: {}", e)),
            },
        }
```

```
            match self
                .client
                .create_contact()
                .contact_list_name(CONTACT_LIST_NAME)
                .email_address(email.clone())
                .send()
                .await
            {
                Ok(_) => writeln!(self.stdout, "Contact created for {}", email)?,
                Err(e) => match e.into_service_error() {
                    CreateContactError::AlreadyExistsException(_) => writeln!(
                        self.stdout,
                        "Contact already exists for {}, skipping creation.",
                        email
                    )?,
                    e => return Err(anyhow!("Error creating contact for {}: {}",
 email, e)),
                },
            }

        let contacts: Vec<Contact> = match self
            .client
            .list_contacts()
            .contact_list_name(CONTACT_LIST_NAME)
            .send()
            .await
        {
            Ok(list_contacts_output) => {
                list_contacts_output.contacts.unwrap().into_iter().collect()
            }
            Err(e) => {
                return Err(anyhow!(
                    "Error retrieving contact list {}: {}",
                    CONTACT_LIST_NAME,
                    e
                ))
            }
        };

            let coupons = std::fs::read_to_string("../resources/newsletter/
sample_coupons.json")
                .unwrap_or_else(|_| r#"{"coupons":[]}"#.to_string());
            let email_content = EmailContent::builder()
```

```
                .template(
                    Template::builder()
                        .template_name(TEMPLATE_NAME)
                        .template_data(coupons)
                        .build(),
                )
                .build();

            match self
                .client
                .send_email()
                .from_email_address(self.verified_email.clone())

.destination(Destination::builder().to_addresses(email.clone()).build())
                .content(email_content)
                .list_management_options(
                    ListManagementOptions::builder()
                        .contact_list_name(CONTACT_LIST_NAME)
                        .build()?,
                )
                .send()
                .await
            {
                Ok(output) => {
                    if let Some(message_id) = output.message_id {
                        writeln!(
                            self.stdout,
                            "Newsletter sent to {} with message ID {}",
                            email, message_id
                        )?;
                    } else {
                        writeln!(self.stdout, "Newsletter sent to {}", email)?;
                    }
                }
                Err(e) => return Err(anyhow!("Error sending newsletter to {}:
{}", email, e)),
            }

        match self
            .client
            .create_email_identity()
            .email_identity(self.verified_email.clone())
            .send()
            .await
```

```
        {
            Ok(_) => writeln!(self.stdout, "Email identity created
 successfully.")?,
            Err(e) => match e.into_service_error() {
                CreateEmailIdentityError::AlreadyExistsException(_) => {
                    writeln!(
                        self.stdout,
                        "Email identity already exists, skipping creation."
                    )?;
                }
                e => return Err(anyhow!("Error creating email identity: {}", e)),
            },
        }

        let template_html =
            std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.html")
                .unwrap_or_else(|_| "Missing coupon-
newsletter.html".to_string());
        let template_text =
            std::fs::read_to_string("../resources/newsletter/coupon-
newsletter.txt")
                .unwrap_or_else(|_| "Missing coupon-newsletter.txt".to_string());

        // Create the email template
        let template_content = EmailTemplateContent::builder()
            .subject("Weekly Coupons Newsletter")
            .html(template_html)
            .text(template_text)
            .build();

        match self
            .client
            .create_email_template()
            .template_name(TEMPLATE_NAME)
            .template_content(template_content)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Email template created
 successfully.")?,
            Err(e) => match e.into_service_error() {
                CreateEmailTemplateError::AlreadyExistsException(_) => {
                    writeln!(
```

```
                        self.stdout,
                        "Email template already exists, skipping creation."
                    )?;
                }
                e => return Err(anyhow!("Error creating email template: {}", e)),
            },
        }

        match self
            .client
            .delete_contact_list()
            .contact_list_name(CONTACT_LIST_NAME)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Contact list deleted
successfully.")?,
            Err(e) => return Err(anyhow!("Error deleting contact list: {e}")),
        }

            match self
                .client
                .delete_email_identity()
                .email_identity(self.verified_email.clone())
                .send()
                .await
            {
                Ok(_) => writeln!(self.stdout, "Email identity deleted
successfully.")?,
                Err(e) => {
                    return Err(anyhow!("Error deleting email identity: {}", e));
                }
            }

        match self
            .client
            .delete_email_template()
            .template_name(TEMPLATE_NAME)
            .send()
            .await
        {
            Ok(_) => writeln!(self.stdout, "Email template deleted
successfully.")?,
            Err(e) => {
```

```
                return Err(anyhow!("Error deleting email template: {e}"));
            }
        }
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList
  - DeleteEmailIdentity
  - DeleteEmailTemplate
  - ListContacts
  - SendEmail.simple
  - SendEmail.template

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SES with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon Simple Email Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the .
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon Simple Email Service. It shows you how to configure Amazon Simple Email Service to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Simple Email Service resources.

> **Note**
>
> If you need to report abuse of AWS resources, including email spam and malware distribution, do not use the feedback link on any of the pages of this developer guide, as the form is received by the AWS Documentation team, not AWS Trust & Safety. Instead, on the How do I report abuse of AWS resources? page, follow the directions to contact the AWS Trust & Safety team to report any type of Amazon AWS abuse.

Contents

- Data protection in Amazon Simple Email Service
- Identity and access management in Amazon SES
- Logging and monitoring in Amazon SES
- Compliance validation for Amazon Simple Email Service

- Resilience in Amazon Simple Email Service
- Infrastructure security in Amazon Simple Email Service
- Setting up VPC endpoints with Amazon SES

Data protection in Amazon Simple Email Service

The AWS shared responsibility model applies to data protection in Amazon Simple Email Service. 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 Amazon Simple Email Service 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.

Contents

- Data encryption at rest for Amazon SES
- Encryption in transit
- Deleting personal data from Amazon SES

Data encryption at rest for Amazon SES

By default, Amazon SES encrypts all data at rest. Encryption by default helps reduce the operational overhead and complexity involved in protecting data. Encryption also enables you to create Mail Manager archives that meet strict encryption compliance and regulatory requirements.

SES provides the following encryption options:

- **AWS owned keys** – SES uses these by default. You can't view, manage, or use AWS owned keys, or audit their use. However, you don't have to take any action or change any programs to protect the keys that encrypt your data. For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.
- **Customer managed keys** – SES supports the use of symmetric customer managed keys that you create, own, and manage. Because you have full control of the encryption, you can perform such tasks as:
  - Establishing and maintaining key policies
  - Establishing and maintaining IAM policies and grants
  - Enabling and disabling key policies
  - Rotating key cryptographic material
  - Adding tags
  - Creating key aliases
  - Scheduling keys for deletion

  To use your own key, choose a customer managed key when you create your SES resources.

  For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> SES automatically enables encryption at rest using AWS owned keys at no charge. However, AWS KMS charges apply for using a customer managed key. For more information about pricing, see the AWS Key Management Service pricing.

Create a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs.

To create a symmetric customer managed key

Follow the steps for Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> For archiving, your key must meet the following requirements:
>
> - The key must be symmetric.
> - The key material origin must be `AWS_KMS`.
> - The key usage must be `ENCRYPT_DECRYPT`.

Key policy

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see Managing access to customer managed keys in the *AWS Key Management Service Developer Guide*.

To use your customer managed key with Mail Manager archiving, your key policy must permit the following API operations:

- kms:DescribeKey – Provides the customer managed key details that allow SES to validate the key.
- kms:GenerateDataKey – Allows SES to generate a data key for encrypting data at rest.

- kms:Decrypt – Allows SES to decrypt stored data before returning it to API clients.

The following example shows a typical key policy:

```
{
            "Sid": "Allow SES to encrypt/decrypt",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
```

For more information, see specifying permissions in a policy, in the *AWS Key Management Service Developer Guide*.

For more information about troubleshooting, see troubleshooting key access, in the *AWS Key Management Service Developer Guide*.

Specifying a customer managed key for Mail Manager

You can specify a customer managed key as an alternative to using AWS owned keys. When you create an archive or configure an ingress endpoint with mutual TLS (mTLS) authentication, you can specify the data key by entering a **KMS key ARN**. For archiving, Mail Manager uses the key to encrypt all customer data in the archive. For mTLS ingress endpoints, Mail Manager uses the key to encrypt the trust store contents at rest.

- **KMS key ARN** – A key identifier for a AWS KMS customer managed key. Enter a key ID, key ARN, alias name, or alias ARN.

Amazon SES encryption context

An encryption context is an optional set of key-value pairs that contain additional contextual information about the data.

AWS KMS uses the encryption context as additional authenticated data to support authenticated encryption. When you include an encryption context in a request to encrypt data, AWS KMS binds the encryption context to the encrypted data. To decrypt data, you include the same encryption context in the request.

> **Note**
>
> Amazon SES doesn't support encryption contexts for archive creation. Instead, you use an IAM or KMS policy. For example policies, see Archive creation policies, later in this section.

Amazon SES encryption context

SES uses the same encryption context in all AWS KMS cryptographic operations, where the key is `aws:ses:arn` and the value is the resource Amazon Resource Name (ARN).

Example

```
"encryptionContext": {
    "aws:ses:arn": "arn:aws:ses:us-west-2:111122223333:ExampleResourceName/
ExampleResourceID"
}
```

Using encryption context for monitoring

When you use a symmetric customer managed key to encrypt your SES resource, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used. The encryption context also appears in logs generated by AWS CloudTrail or Amazon CloudWatch Logs.

Using encryption context to control access to your customer managed key

You can use the encryption context in key policies and IAM policies as `conditions` to control access to your symmetric customer managed key. You can also use encryption context constraints in a grant.

SES uses an encryption context constraint in grants to control access to the customer managed key in your account or region. The grant constraint requires that the operations that the grant allows use the specified encryption context.

Example

The following are example key policy statements to grant access to a customer managed key for a specific encryption context. The condition in this policy statement requires that the grants have an encryption context constraint that specifies the encryption context.

```
{
    "Sid": "Enable DescribeKey",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ExampleReadOnlyRole"
     },
     "Action": "kms:DescribeKey",
     "Resource": "*"
},
{
     "Sid": "Enable CreateGrant",
     "Effect": "Allow",
     "Principal": {
         "AWS": "arn:aws:iam::111122223333:role/ExampleReadOnlyRole"
     },
     "Action": "kms:CreateGrant",
     "Resource": "*",
     "Condition": {
         "StringEquals": {
             "kms:EncryptionContext:aws:ses:arn": "arn:aws:ses:us-
west-2:111122223333:ExampleResourceName/ExampleResourceID"
          }
     }
}
```

Archive creation policies

The following example policies show how to enable archive creation. The policies work on all assets.

IAM policy

```
{
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "ses:CreateArchive",
            "Resource": [
```

```
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "ses.us-east-1.amazonaws.com",
                    "kms:CallerAccount": "012345678910"
                }
            }
        }
```

AWS KMS policy

```
{
            "Sid": "Allow SES to encrypt/decrypt",
            "Effect": "Allow",
            "Principal": {
                "Service": "ses.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
```

Ingress endpoint mTLS policies

The following example policies enable using a customer managed key to encrypt trust store contents for mutual TLS (mTLS) authentication on Mail Manager ingress endpoints.

To scope the example policies to a specific ingress endpoint, replace the wildcard in the condition with an exact resource ARN (for example, `arn:aws:ses:us-`

east-1:111122223333:mailmanager-ingress-point/inp-ab1c2defgh3ij4klmno5pq6rs).

IAM policy

```
{
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/rolename"
    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "kms:ViaService": "ses.us-east-1.amazonaws.com"
      },
      "StringLike": {
        "kms:EncryptionContext:aws:ses:arn": [
          "arn:aws:ses:us-east-1:111122223333:mailmanager-ingress-point/*"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/rolename"
    },
    "Action": [
      "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "kms:ViaService": "ses.us-east-1.amazonaws.com"
      }
    }
  }
```

AWS KMS policy

```
{
    "Effect": "Allow",
    "Principal": {
      "Service": "ses.amazonaws.com"
    },
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:ses:us-east-1:111122223333:mailmanager-ingress-point/*"
        ],
        "kms:EncryptionContext:aws:ses:arn": [
          "arn:aws:ses:us-east-1:111122223333:mailmanager-ingress-point/*"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "ses.amazonaws.com"
    },
    "Action": [
      "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:ses:us-east-1:111122223333:mailmanager-ingress-point/*"
        ]
      }
    }
  }
```

Monitoring your encryption keys for Amazon SES

When you use an AWS KMS customer managed key with your Amazon SES resources, you can use AWS CloudTrail or Amazon CloudWatch Logs to track requests that SES sends to AWS KMS.

The following examples are AWS CloudTrail events for `GenerateDataKey`, `Decrypt`, and `DescribeKey` to monitor KMS operations called by SES to access data encrypted by your customer managed key:

GenerateDataKey

When you enable an AWS KMS customer managed key for your resource, SES creates a unique table key. It sends a `GenerateDataKey` request to AWS KMS that specifies the AWS KMScustomer managed key for the resource.

When you enable an AWS KMS customer managed key for your Mail Manager archive resource, it will use `GenerateDataKey` when encrypting archive data at rest.

The following example event records the `GenerateDataKey` operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ses.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:07:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:ses:arn": "arn:aws:ses:us-west-2:111122223333:ExampleResourceName/
ExampleResourceID"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
```

```
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "57f5dbee-16da-413e-979f-2c4c6663475e"
}
```

Decrypt

When you access an encrypted resource, SES calls the `Decrypt` operation to use the stored encrypted data key to access the encrypted data.

The following example event records the `Decrypt` operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ses.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:10:51Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:ses:arn": "arn:aws:ses:us-west-2:111122223333:ExampleResourceName/
ExampleResourceID"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
```

```
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```

DescribeKey

SES uses the `DescribeKey` operation to verify if the AWS KMS customer managed key associated with your resource exists in the account and region.

The following example event records the `DescribeKey` operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-22T17:02:00Z"
```

```
            }
        },
        "invokedBy": "ses.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:07:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "keyId": "00dd0db0-0000-0000-ac00-b0c000SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```


Learn more

The following resources provide more information about data encryption at rest.

- For more information about AWS Key Management Service basic concepts, see the *AWS Key Management Service Developer Guide*.
- For more information about Security best practices for AWS Key Management Service, see the *AWS Key Management Service Developer Guide*.

Encryption in transit

By default, Amazon SES uses opportunistic TLS. This means that Amazon SES always attempts to make a secure connection to the receiving mail server. If it can't establish a secure connection, it sends the message unencrypted. You can change this behavior so that Amazon SES sends the message to the receiving email server only if it can establish a secure connection. For more information, see Amazon SES and security protocols.

Deleting personal data from Amazon SES

Depending on how you use it, Amazon SES might store certain data that could be considered personal. For example, in order to send email using Amazon SES, you must provide at least one verified identity (an email address or a domain). You can use the Amazon SES console or the Amazon SES API to permanently delete this personal data.

This chapter provides procedures for deleting various types of data that might be considered personal.

Contents

- Delete Email Addresses From the Account-Level Suppression List
- Delete Data About Email Sent Using Amazon SES
- Delete Data About Identities
- Delete Sender Authentication Data
- Delete Data Related to Receiving Rules
- Delete Data Related to IP Address Filters
- Delete Data in Email Templates
- Delete Data in Custom Verification Email Templates
- Delete All Personal Data by Closing Your AWS Account

Delete Email Addresses From the Account-Level Suppression List

Amazon SES includes an optional account-level suppression list. When you enable this feature, email addresses are automatically added to a suppression list when they result in a bounce or complaint. Email addresses remain on this list until you delete them. For more information about the account-level suppression list, see Using the Amazon SES account-level suppression list.

You can remove email addresses from the account-level suppression list by using the `DeleteSuppressedDestination` operation in the Amazon SES API v2. This section includes a procedure for deleting email addresses by using the AWS CLI. For more information about installing and configuring the AWS CLI, see the AWS Command Line Interface User Guide.

To remove an address from the account-level suppression list by using the AWS CLI

- At the command line, enter the following command:

  ```
  aws sesv2 delete-suppressed-destination --email-address recipient@example.com
  ```

  In the preceding command, replace *recipient@example.com* with the email address that you want to remove from the account-level suppression list.

Delete Data About Email Sent Using Amazon SES

When you use Amazon SES to send an email, you can send information about that email to other AWS services. For example, you can send information about email events (such as deliveries, opens, and clicks) to Firehose. This event data typically contains your email address and the IP address the email was sent from. It also contains the email addresses of all the recipients the email was sent to.

You can use Firehose to stream email event data to several destinations—including Amazon Simple Storage Service, Amazon OpenSearch Service, and Amazon Redshift. To remove this data, you should first stop streaming data to Firehose, and then delete the data that has already been streamed. To stop streaming Amazon SES event data to Firehose, you must delete the Firehose event destination.

To remove a Firehose event destination by using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Under **Email Sending**, choose **Configuration Sets**.
3. In the list of configuration sets, choose the configuration set that contains the Firehose event destination.
4. Next to the Firehose event destination that you want to delete, choose the **delete** ( ) button.

5. If necessary, remove the data that Firehose wrote to other services. For more information, see the section called “Remove Stored Event Data”.

You can also use the Amazon SES API to delete event destinations. The following procedure uses the AWS Command Line Interface (AWS CLI) to interact with the Amazon SES API. You can also interact with the API by using an AWS SDK, or by making HTTP requests directly.

To remove a Firehose event destination by using the AWS CLI

1. At the command line, type the following command:

```
aws sesv2 delete-configuration-set-event-destination --configuration-set-
name configSet \
--event-destination-name eventDestination
```

   In this command, replace *configSet* with the name of the configuration set that contains the Firehose event destination. Replace *eventDestination* with the name of the Firehose event destination.
2. If necessary, remove the data that Firehose wrote to other services. For more information, see the section called “Remove Stored Event Data”.

Remove Stored Event Data

For more information about deleting information from other AWS services, see the following documents:

- Delete an Object and Bucket in the *Amazon Simple Storage Service User Guide*
- Delete an OpenSearch Service Domain in the *Amazon OpenSearch Service Developer Guide*
- Deleting a Cluster in the *Amazon Redshift Cluster Management Guide*

Delete Data About Identities

Identities include the email addresses and domains that you use to send email using Amazon SES. In some jurisdictions, email addresses or domains might be considered personally identifiable data.

To delete an identity by using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.

2. Under **Identity Management**, do one of the following:
   - Choose **Domains** if you want to delete a domain.
   - Choose **Email Addresses** if you want to delete an email address.
3. Choose the identity that you want to delete, and then choose **Remove**.
4. On the confirmation dialog box, choose **Yes, Delete Identity**.

You can also use the Amazon SES API to delete identities. The following procedure uses the AWS Command Line Interface (AWS CLI) to interact with the Amazon SES API. You can also interact with the API by using an AWS SDK, or by making HTTP requests directly.

To delete an identity by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-identity --identity sender@example.com
```

  In this command, replace *sender@example.com* with the identity that you want to delete.

Delete Sender Authentication Data

Sender authentication refers to the process of configuring Amazon SES so that another user can send email on your behalf. To enable sender authorization, you must create a policy, as described in Using sending authorization with Amazon SES. These policies contain identities (which belong to you), in addition to AWS IDs (which are associated with the person or group that sends email on your behalf). You can remove this personal data by modifying or deleting the sender authentication policies. The following procedures show you how to delete these policies.

To delete a sender authentication policy by using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Under **Identity Management**, do one of the following:
   - Choose **Domains** if the sender authentication policy you want to delete is associated with a domain.
   - Choose **Email Addresses** if the sender authentication policy you want to delete is associated with an email address.

3. Under **Identity Policies**, choose the policy you want to delete, and then choose **Remove Policy**.

You can also use the Amazon SES API to delete sender authentication policies. The following procedure uses the AWS Command Line Interface (AWS CLI) to interact with the Amazon SES API. You can also interact with the API by using an AWS SDK, or by making HTTP requests directly.

To delete a sender authentication policy by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-identity-policy --identity example.com --policy-name samplePolicy
```

  In this command, replace *example.com* with the identity that contains the sender authentication policy. Replace *samplePolicy* with the name of the sender authentication policy.

Delete Data Related to Receiving Rules

If you use Amazon SES to receive incoming email, you can create receipt rules that are applied to one or more identities (email addresses or domains). These rules determine what Amazon SES does with incoming mail sent to the specified identities.

To delete a receipt rule by using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Under **Email Receiving**, choose **Rule Sets**.
3. If the receipt rule is part of the active rule set, choose **View Active Rule Set**. Otherwise, choose the rule set that contains the receipt rule that you want to delete.
4. In the list of receipt rules, choose the rule that you want to delete.
5. On the **Actions** menu, choose **Delete**.
6. On the confirmation dialog box, choose **Delete**.

You can also use the Amazon SES API to delete receipt rules. The following procedure uses the AWS Command Line Interface (AWS CLI) to interact with the Amazon SES API. You can also interact with the API by using an AWS SDK, or by making HTTP requests directly.

To delete a receipt rule by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-receipt-rule --rule-set myRuleSet --rule-name myReceiptRule
```

  In this command, replace *myRuleSet* with the name of the receipt rule set that contains the receipt rule. Replace *myReceiptRule* with the name of the receipt rule that you want to delete.

Delete Data Related to IP Address Filters

If you use Amazon SES to receive incoming email, you can create filters to explicitly accept or block messages that are sent from specific IP addresses.

To delete an IP address filter by using the Amazon SES console

1. Open the Amazon SES console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ses/.
2. Under **Email Receiving**, choose **IP Address Filters**.
3. In the list of IP address filters, choose the filter that you want to remove, and then choose **Delete**.

You can also use the Amazon SES API to delete IP address filters. The following procedure uses the AWS Command Line Interface (AWS CLI) to interact with the Amazon SES API. You can also interact with the API by using an AWS SDK, or by making HTTP requests directly.

To delete an IP address filter by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-receipt-filter --filter-name IPfilter
```

  In this command, replace *IPfilter* with the name of the IP address filter you want to delete.

Delete Data in Email Templates

If you use email templates for sending email, it's possible that those templates might contain personal data, depending on how you configured them. For example, you might have added an email address to the template that recipients could contact for more information.

You can only delete email templates by using the Amazon SES API.

To delete an email template by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-template --template-name sampleTemplate
```

  In this command, replace *sampleTemplate* with the name of the email template that you want to delete.

Delete Data in Custom Verification Email Templates

If you use customized templates for verifying new email sending addresses, it's possible that those templates might contain personal data, depending on how you configured them. For example, you might have added an email address to the verification email template that recipients could contact for more information.

You can only delete custom verification email templates by using the Amazon SES API.

To delete a custom verification email template by using the AWS CLI

- At the command line, type the following command:

```
aws ses delete-custom-verification-email-template --template-
name verificationEmailTemplate
```

  In this command, replace *verificationEmailTemplate* with the name of the custom verification email template that you want to delete.

Delete All Personal Data by Closing Your AWS Account

It's also possible to delete all personal data that's stored in Amazon SES by closing your AWS account. However, this action also deletes all other data—personal or non-personal—that you have stored in every other AWS service.

When you close your AWS account, the data in your AWS account is retained for 90 days. After that retention period, it's deleted permanently and irreversibly.

To close your AWS account

Complete instructions on how to close your AWS account is covered in Close an AWS account.

Identity and access management in Amazon SES

You can use AWS Identity and Access Management (IAM) with Amazon Simple Email Service (Amazon SES) to specify which SES API actions an user, group, or role can perform. (In this topic we refer to these entities collectively as *user*.) You can also control which email addresses the user can use for the "From", recipient, and "Return-Path" addresses of emails.

For example, you can create an IAM policy that allows users in your organization to send email, but not perform administrative actions such as checking sending statistics. As another example, you can write a policy that allows a user to send emails through SES from your account, but only if they use a specific "From" address.

To use IAM, you define an IAM policy, which is a document that explicitly defines permissions, and attach the policy to a user. To learn how to create IAM policies, see the IAM User Guide. Other than applying the restrictions you set in your policy, there are no changes to how users interact with SES or in how SES carries out requests.

Note

- If your account is in the SES sandbox, its restrictions well prevent the implementation of some of these polices - see Request production access.
- You can also control access to SES by using sending authorization policies. Whereas IAM policies constrain what individual users can do, sending authorization policies constrain how individual verified identities can be used. Further, only sending authorization policies

can grant cross-account access. For more information about sending authorization, see Using sending authorization with Amazon SES.

If you are looking for information about how to generate SES SMTP credentials for an existing user, see Obtaining Amazon SES SMTP credentials.

Creating IAM Policies for Access to SES

This section explains how you can use IAM policies specifically with SES. To learn how to create IAM policies in general, see the IAM User Guide.

There are three reasons you might use IAM with SES:

- To restrict the email-sending action.
- To restrict the "From", recipient, and "Return-Path" addresses of the emails that the user sends.
- To control general aspects of API usage such as the time period during which a user is permitted to call the APIs that they are authorized to use.

Restricting the Action

To control which SES actions a user can perform, you use the `Action` element of an IAM policy. You can set the `Action` element to any SES API action by prefixing the API name with the lowercase string `ses:`. For example, you can set the `Action` to `ses:SendEmail`, `ses:GetSendStatistics`, or `ses:*` (for all actions).

Then, depending on the `Action`, specify the `Resource` element as follows:

If the `Action` element only permits access to email-sending APIs (that is, `ses:SendEmail` and/or `ses:SendRawEmail`):

- To allow the user to send from any identity in your AWS account, set `Resource` to *
- To restrict the identities that a user is allowed to send from, set `Resource` to the ARNs of the identities that you are permitting the user to use.

If the `Action` element permits access to all APIs:

- If you don't want to restrict the identities that the user can send from, set `Resource` to *

- If you want to restrict the identities that a user is allowed to send from, you need to create two policies (or two statements within one policy):
  - One with `Action` set to an explicit list of the permitted non-email-sending APIs and `Resource` set to *
  - One with `Action` set to one of the email-sending APIs (`ses:SendEmail` and/or `ses:SendRawEmail`), and `Resource` set to the ARN(s) of the identities you are permitting the user to use.

For a list of available SES actions, see the Amazon Simple Email Service API Reference. If the user will be using the SMTP interface, you must allow access to `ses:SendRawEmail` at a minimum.

Restricting Email Addresses

If you want to restrict the user to specific email addresses, you can use a `Condition` block. In the `Condition` block, you specify conditions by using condition keys as described in the IAM User Guide. By using condition keys, you can control the following email addresses:

> **Note**
>
> These email address condition keys apply only to the APIs noted in the following table.

Condition Key	Description	API
`ses:Recipients`	Restricts the recipient addresses, which include the To:, "CC", and "BCC" addresses.	`SendEmail` , `SendRawEmail`
`ses:FromAddress`	Restricts the "From" address.	`SendEmail` , `SendRawEmail` , `SendBounce`
`ses:FromDisplayName`	Restricts the "From" address that is used as the display name.	`SendEmail` , `SendRawEmail`
`ses:FeedbackAddress`	Restricts the "Return-Path" address, which is the address	`SendEmail` , `SendRawEmail`

Condition Key	Description	API
	where bounces and complaints can be sent to you by email feedback forwarding. For information about email feedback forwarding, see Receiving Amazon SES notifications through email.	
`ses:MultiRegionEndpointId`	Allows you to control what endpoint ID is used when sending email	`SendEmail` , `SendBulkEmail`

Restricting by SES API version

By using the `ses:ApiVersion` key in conditions, you can restrict access to SES based on the version of the SES API.

> **Note**
>
> The SES SMTP interface uses SES API version 2 of `ses:SendRawEmail`.

Restricting General API Usage

By using AWS-wide keys in conditions, you can restrict access to SES based on aspects such as the date and time that user is permitted access to APIs. SES implements only the following AWS-wide policy keys:

- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:SecureTransport`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`

- `aws:VpcSourceIp`

For more information about these keys, see the IAM User Guide.

Example IAM Policies for SES

This topic provides examples of policies that permit a user access to SES, but only under certain conditions.

Policy examples in this section:

- Allowing Full Access to All SES Actions
- Allowing Access to only SES API version 2
- Allowing Access to Email-Sending Actions Only
- Restricting the Time Period of Sending
- Restricting the Recipient Addresses
- Restricting the "From" Address
- Restricting the Display Name of the Email Sender
- Restricting the Destination of Bounce and Complaint Feedback

Allowing Full Access to All SES Actions

The following policy allows a user to call any SES action.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:*"
      ],
      "Resource":"*"
    }
  ]
}
```

Allowing Access to only SES API version 2

The following policy allows a user to call only the SES actions of API version 2.

JSON

```
{
                    "Version":"2012-10-17",
                    "Statement":[
                        {
                            "Effect":"Allow",
                            "Action":[
                            "ses:*"
                            ],
                            "Resource":"*",
                            "Condition": {
                                "StringEquals" : {
                                "ses:ApiVersion" : "2"
                                }
                            }
                        }
                    ]
                }
```

Allowing Access to Email-Sending Actions Only

The following policy permits a user to send email using SES, but does not permit the user to perform administrative actions such as accessing SES sending statistics.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
```

```
      "Resource":"*"
    }
  ]
}
```

Restricting the Time Period of Sending

The following policy permits a user to call SES email-sending APIs only during the month of September 2018.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource":"*",
      "Condition":{
        "DateGreaterThan":{
          "aws:CurrentTime":"2018-08-31T12:00Z"
        },
        "DateLessThan":{
          "aws:CurrentTime":"2018-10-01T12:00Z"
        }
      }
    }
  ]
}
```

Restricting the Recipient Addresses

The following policy permits a user to call the SES email-sending APIs, but only to recipient addresses in domain *example.com* (`StringLike` *is case sensitive*).

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource":"*",
      "Condition":{
        "ForAllValues:StringLike":{
          "ses:Recipients":[
            "*@example.com"
          ]
        }
      }
    }
  ]
}
```

Restricting the "From" Address

The following policy permits a user to call the SES email-sending APIs, but only if the "From" address is *marketing@example.com*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource":"*",
```

```
      "Condition":{
        "StringEquals":{
          "ses:FromAddress":"marketing@example.com"
        }
      }
    }
  ]
}
```

The following policy permits a user to call the SendBounce API, but only if the "From" address is *bounce@example.com*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendBounce"
      ],
      "Resource":"*",
      "Condition":{
        "StringEquals":{
          "ses:FromAddress":"bounce@example.com"
        }
      }
    }
  ]
}
```

Restricting the Display Name of the Email Sender

The following policy permits a user to call the SES email-sending APIs, but only if the display name of the "From" address includes *Marketing* (`StringLike` *is case sensitive*).

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource":"*",
      "Condition":{
        "StringLike":{
          "ses:FromDisplayName":"Marketing"
        }
      }
    }
  ]
}
```

Restricting the Destination of Bounce and Complaint Feedback

The following policy permits a user to call the SES email-sending APIs, but only if the "Return-Path" of the email is set to *feedback@example.com*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource":"*",
      "Condition":{
        "StringEquals":{
```

```
          "ses:FeedbackAddress":"feedback@example.com"
        }
      }
    }
  ]
}
```

AWS managed policies for Amazon Simple Email Service

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AmazonSESFullAccess

You can attach the `AmazonSESFullAccess` policy to your IAM identities. Provides full access to Amazon SES.

To view the permissions for this policy, see AmazonSESFullAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AmazonSESReadOnlyAccess

You can attach the `AmazonSESReadOnlyAccess` policy to your IAM identities. Provides read only access to Amazon SES.

To view the permissions for this policy, see AmazonSESReadOnlyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AmazonSESServiceRolePolicy

You can't attach the `AmazonSESServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon SES to perform actions on your behalf. For more information, see Service-linked role permissions for Amazon SES.

To view the permissions for this policy, see AmazonSESServiceRolePolicy in the *AWS Managed Policy Reference*.

Amazon Simple Email Service updates to AWS managed policies

View details and about updates to AWS managed policies for Amazon Simple Email Service since this service began tracking these changes.

Change	Description	Date
Amazon Simple Email Service added a new managed policy	Amazon Simple Email Service added `AmazonSESServiceRolePolicy` to the service-linked role `AWSServiceRoleForAmazonSES` that allows SES to perform actions on your behalf	May 13, 2024
Amazon Simple Email Service updated a policy definition	*Amazon Simple Email Service clarified the previous entry in this table (row below) to be:* Amazon Simple Email Service added `ses:BatchGetMetricData` to AmazonSESReadOnlyAccess	Apr 30, 2024

Change	Description	Date
	managed policy—this will give access to the SES API BatchGetMetricData	
Amazon Simple Email Service updated a policy definition	Amazon Simple Email Service added `ses:BatchGet*` to AmazonSESReadOnlyAccess managed policy—this will give access to the SES API BatchGetMetricData	Feb 16, 2024
Amazon Simple Email Service changed two policy definitio ns	Amazon Simple Email Service removed "via the AWS Management Console" from the end of the AmazonSES FullAccess and AmazonSES ReadOnlyAccess definitions	May 3, 2023
Amazon Simple Email Service started tracking changes	Amazon Simple Email Service started tracking changes to its AWS managed policies	April 5, 2023

Using service-linked roles for Amazon SES

Amazon Simple Email Service (SES) uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon SES. Service-linked roles are predefined by SES and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up SES easier because you don’t have to manually add the necessary permissions. SES defines the permissions of its service-linked roles, and unless defined otherwise, only SES can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your SES resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon SES

SES uses the service-linked role named **AWSServiceRoleForAmazonSES** – Allows SES to publish Amazon CloudWatch basic monitoring metrics on behalf of your SES resources.

The AWSServiceRoleForAmazonSES service-linked role trusts the following service to assume the role:

- `ses.amazonaws.com`

The role permissions policy named AmazonSESServiceRolePolicy is an AWS managed policy that allows SES to complete the following actions on the specified resources:

- Action: `cloudwatch:PutMetricData` in the `AWS/SES` CloudWatch namespace. This action grants permission for SES to put metric data into the CloudWatch `AWS/SES` namespace. For more information about SES metrics available in CloudWatch, see Logging and monitoring in Amazon SES.
- Action: `cloudwatch:PutMetricData` in the `AWS/SES/MailManager` CloudWatch namespace. This action grants permission for SES to put metric data into the CloudWatch `AWS/SES/MailManager` namespace. For more information about SES metrics available in CloudWatch, see Logging and monitoring in Amazon SES.
- Action: `cloudwatch:PutMetricData` in the `AWS/SES/Addons` CloudWatch namespace. This action grants permission for SES to put metric data into the CloudWatch `AWS/SES/Addons` namespace. For more information about SES metrics available in CloudWatch, see Logging and monitoring in Amazon SES.

You must configure permissions to allow your users, groups, or roles to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon SES

You don't need to manually create a service-linked role. When you create SES resources in the AWS Management Console, the AWS CLI, or the AWS API, SES creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create SES resources, SES creates the service-linked role for you again.

Editing a service-linked role for Amazon SES

SES does not allow you to edit the AWSServiceRoleForAmazonSES service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM.

Deleting a service-linked role for SES

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning Up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete all SES resources.

> **Note**
>
> If the SES service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonSES service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for Amazon SES service-linked roles

SES does not support using service-linked roles in every Region where the service is available. You can use the AWSServiceRoleForAmazonSES role in the following Regions.

Region name	Region identity	Support in SES
US East (N. Virginia)	eusc-de-east-1	Yes

Region name	Region identity	Support in SES
	eusc-de-east-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes
Asia Pacific (Tokyo)	ap-northeast-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes

Logging and monitoring in Amazon SES

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SES and your AWS solutions. AWS provides tools to help you monitor Amazon SES and respond to potential incidents.

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For more information, see Retrieving Amazon SES event data from CloudWatch and Creating reputation monitoring alarms using CloudWatch.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see Logging Amazon SES API calls with AWS CloudTrail.
- Amazon SES *email sending events* can help you fine-tune your email sending strategy. Amazon SES captures detailed information, including the numbers of sends, deliveries, opens, clicks, bounces, complaints, and rejections. For more information, see Monitoring sending activity.
- Amazon SES *reputation metrics* tracks the bounce and complaint rates for your account. For more information, see Monitoring sender reputation.

Logging Amazon SES API calls with AWS CloudTrail

Amazon SES is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in SES. CloudTrail captures API calls for SES as events. The calls captured include calls from the SES console and code calls to the SES API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for SES. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to SES, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

SES information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in SES, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for SES, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

SES data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource. These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn't log data events. The CloudTrail event history doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail pricing.

> **Note**
>
> Email sending activity via SES SMTP Interface is not logged to CloudTrail events. For comprehensive activity logging, use the latest SES APIs in the SES API Reference and SES API v2 Reference.

The following table lists the SES resource types for which you can log data events. The *Data event type (console)* column shows the value to choose from the **Data event type** list on the CloudTrail console. The *resources.type value* column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the column shows the AWS CLI or CloudTrail APIs. The *Data APIs logged to CloudTrail* column shows the API calls logged to CloudTrail for the resource type.

SES resource types for data events

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
SES identity	AWS::SES::EmailIdentity	**SES:**
SES configuration set	AWS::SES::ConfigurationSet	SendEmail

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
		SendRawEmail SendTemplatedEmail SendBulkTemplatedEmail **SES v2:** SendEmail SendBulkEmail
SES template	AWS::SES::Template	**SES:** SendTemplatedEmail SendBulkTemplatedEmail **SES v2:** SendEmail SendBulkEmail

The following example shows how to log all data events for all SES email identities by using the `--advanced-event-selectors` parameter:

```
aws cloudtrail put-event-selectors \
--region Region \
--trail-name TrailName \
--advanced-event-selectors
'[
    {
        "Name": "Log SES data plane actions for all email identities",
        "FieldSelectors": [
        { "Field": "eventCategory", "Equals": ["Data"] },
        { "Field": "resources.type", "Equals": ["AWS::SES::EmailIdentity"] }
       ]
    }
```

```
]'
```

You can further refine the advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector in the *AWS CloudTrail API Reference*. For more examples on how to log data events see Logging data events for trails.

CloudTrail log delivery scenarios for SES logging

CloudTrail delivers logs based on such factors as account and resource ownership, identity type, and region. The following matrix explains to whom and where the logs would be delivered to based on specific combinations of these factors.

Scenario type	Account roles	Resources	Request flow	Log delivery
Single cross-account	Account A: *resource owner* Account B: *request*	Email identity	B → A's email identity	Logs delivered to both A and B
		Feedback forwarding email	B → A's feedback email	Logs delivered to both A and B
Multiple cross-account	Account A: *feedbac* Account B: *email id* Account C: *request*	Feedback email (A) Email identity (B)	C → A's feedback email + B's email identity	Logs delivered to A, B, and C
Global endpoint (Single account)	Account A: *owner &*	Global endpoint (primary: *eu#west#* & secondary : *us#west#2*)	A → Global endpoint	Logs delivered to A in region that processed the request (either *eu#west#1* or *us#west#2*)
Global endpoint (Cross-account)	Account A: *email id* Account B: *request*	Email identity (A)	B → A's email identity via Global endpoint	Logs delivered to both A and B in region that

Scenario type	Account roles	Resources	Request flow	Log delivery
		Global endpoint (B) (*eu#west#1* & *us#west#2*)		processed the request (either *eu#west#1* or *us#west#2*)

Note

- CloudTrail always delivers logs to the requester account.
- Resource owners receive logs even if they didn't perform the operation.
- For Global endpoints, both accounts need CloudTrail subscriptions in all configured regions.
- During regional impairments, all logs appear in the healthy region.

SES management events in CloudTrail

SES delivers management events to CloudTrail. Management events include actions that are related to creating and managing resources within your AWS account. In Amazon SES, management events include actions such as creating and deleting identities or receipt rules. For more information about SES API operations, see the SES API Reference and SES API v2 Reference.

CloudTrail log file entries for SES

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following examples demonstrate CloudTrail logs of these event types:

Event types

- DeleteIdentity
- VerifyEmailIdentity

- SendEmail with simple content
- SendEmail with templated content

DeleteIdentity


```
{
  "Records":[
    {
        "eventVersion": "1.11",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "AROA4DO2KAWIPZEXAMPLE:myUserName",
            "arn": "arn:aws:sts::111122223333:assumed-role/users/myUserName",
            "accountId": "111122223333",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "AROA4DO2KAWIPZEXAMPLE",
                    "arn": "arn:aws:iam::111122223333:role/admin-role",
                    "accountId": "111122223333",
                    "userName": "myUserName"
                },
                "attributes": {
                    "creationDate": "2025-02-27T09:53:35Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2025-02-27T09:54:31Z",
        "eventSource": "ses.amazonaws.com",
        "eventName": "DeleteIdentity",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "aws-cli/2.23.4",
        "requestParameters": {
            "identity": "sender@example.com"
        },
        "responseElements": null,
        "requestID": "50b87bfe-ab23-11e4-9106-5b36376f9d12",
        "eventID": "0ffa308d-1467-4259-8be3-c749753be325",
        "readOnly": false,
```

```
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management",
        "tlsDetails": {
            "tlsVersion": "TLSv1.3",
            "cipherSuite": "TLS_AES_128_GCM_SHA256",
            "clientProvidedHostHeader": "email.us-east-1.amazonaws.com"
        }
    }
  ]
}
```

VerifyEmailIdentity

```
{
  "Records":[
    {
        "eventVersion": "1.11",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "AROA4DO2KAWIPZEXAMPLE:myUserName",
            "arn": "arn:aws:sts::111122223333:assumed-role/users/myUserName",
            "accountId": "111122223333",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "AROA4DO2KAWIPZEXAMPLE",
                    "arn": "arn:aws:iam::111122223333:role/admin-role",
                    "accountId": "111122223333",
                    "userName": "myUserName"
                },
                "attributes": {
                    "creationDate": "2025-02-27T09:53:35Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2025-02-27T09:56:20Z",
        "eventSource": "ses.amazonaws.com",
        "eventName": "VerifyEmailIdentity",
        "awsRegion": "us-east-1",
```

```
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "aws-cli/2.23.4",
        "requestParameters": {
            "emailAddress": "sender@example.com"
        },
        "responseElements": null,
        "requestID": "eb2ff803-ac09-11e4-8ff5-a56a3119e253",
        "eventID": "5613b0ff-d6c6-4526-9b53-a603a9231725",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management",
        "tlsDetails": {
            "tlsVersion": "TLSv1.3",
            "cipherSuite": "TLS_AES_128_GCM_SHA256",
            "clientProvidedHostHeader": "email.us-east-1.amazonaws.com"
        }
    }
  ]
}
```

SendEmail with simple content

```
{
  "Records":[{
       "eventTime": "2025-01-24T11:43:00Z",
        "eventSource": "ses.amazonaws.com",
        "eventName": "SendEmail",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "aws-cli/2.23.4 md/awscrt#0.23.4",
        "requestParameters": {
            "destination": {
                "bccAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"],
                "toAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"],
                "ccAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"]
            },
            "message": {
                "subject": {
                    "charset": "UTF-8",
                    "data": "HIDDEN_DUE_TO_SECURITY_REASONS"
                },
```

```
                "body": {
                    "html": {
                        "charset": "UTF-8",
                        "data": "HIDDEN_DUE_TO_SECURITY_REASONS"
                    },
                    "text": {
                        "charset": "UTF-8",
                        "data": "HIDDEN_DUE_TO_SECURITY_REASONS"
                    }
                }
            },
            "source": "sender@example.com"
        },
        "responseElements": null,
        "additionalEventData": {
            "sesMessageId": "01000100a11a11aa-00aa0a00-00a0-48a8-aaa7-
a174a83b456a-000000"
        },
        "requestID": "ab2cc803-ac09-11d7-8bb8-a56a3119e476",
        "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
        "readOnly": true,
        "resources": [{
            "accountId": "111122223333",
            "type": "AWS::SES::EmailIdentity",
            "ARN": "arn:aws:ses:us-east-1:111122223333:identity/sender@example.com"
        }],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data",
        "tlsDetails": {
            "tlsVersion": "TLSv1.3",
            "cipherSuite": "TLS_AES_128_GCM_SHA256",
            "clientProvidedHostHeader": "email.us-east-1.amazonaws.com"
        }
    }
  ]
}
```


SendEmail with templated content


```
{
        "eventVersion": "1.11",
```

```
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "AROA4DO2KAWIPZEXAMPLE:myUserName",
            "arn": "arn:aws:sts::111122223333:assumed-role/users/myUserName",
            "accountId": "111122223333",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "AROA4DO2KAWIPZEXAMPLE",
                    "arn": "arn:aws:iam::111122223333:role/admin-role",
                    "accountId": "111122223333",
                    "userName": "admin-role"
                },
                "attributes": {
                    "creationDate": "2025-03-05T18:51:06Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2025-03-05T19:16:29Z",
        "eventSource": "ses.amazonaws.com",
        "eventName": "SendEmail",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "aws-cli/2.23.4",
        "requestParameters": {
            "fromEmailAddress": "sender@example.com",
            "destination": {
                "toAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"],
                "bccAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"],
                "ccAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"]
            },
            "emailTags": [{
                "value": "test",
                "name": "campaign"
            }, {
                "value": "cli-test",
                "name": "sender"
            }],
            "replyToAddresses": ["HIDDEN_DUE_TO_SECURITY_REASONS"],
            "content": {
                "template": {
                    "templateData": "HIDDEN_DUE_TO_SECURITY_REASONS",
```

```
                    "templateName": "TestTemplate"
                }
            }
        },
        "responseElements": null,
        "additionalEventData": {
            "sesMessageId": "01000100a11a11aa-00aa0a00-00a0-48a8-aaa7-
a174a83b456a-000000"
        },
        "requestID": "50b87bfe-ab23-11e4-9106-5b36376f9d12",
        "eventID": "0ffa308d-1467-4259-8be3-c749753be325",
        "readOnly": true,
        "resources": [{
            "accountId": "111122223333",
            "type": "AWS::SES::EmailIdentity",
            "ARN": "arn:aws:ses:us-east-1:111122223333:identity/sender@example.com"
        }, {
            "accountId": "111122223333",
            "type": "AWS::SES::Template",
            "ARN": "arn:aws:ses:us-east-1:111122223333:template/TestTemplate"
        }],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data",
        "tlsDetails": {
            "tlsVersion": "TLSv1.3",
            "cipherSuite": "TLS_AES_128_GCM_SHA256",
            "clientProvidedHostHeader": "email.us-east-1.amazonaws.com"
        }
    }
```


Compliance validation for Amazon Simple Email Service

Third-party auditors assess the security and compliance of Amazon Simple Email Service as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see . For general information, see .

Your compliance responsibility when using Amazon Simple Email Service is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

Resilience in Amazon Simple Email Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Amazon Simple Email Service

As a managed service, Amazon Simple Email Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Simple Email Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Setting up VPC endpoints with Amazon SES

Many Amazon SES customers have corporate policies in place that limit the ability of their internal systems to connect to the public internet. These policies prevent the use of the public Amazon SES endpoints.

If you have similar policies, you can work within these restrictions by using Amazon Virtual Private Cloud. With Amazon VPC, you can deploy AWS resources into a virtual network that exists in an isolated area of the AWS Cloud. For more information about Amazon VPC, see the Amazon VPC User Guide.

You can connect directly from Amazon VPC to SES through a VPC Endpoint in a secure and scalable manner. When you use an interface VPC endpoint, it provides a better security posture as you don't need to open outbound traffic firewalls as well as providing other benefits of using Amazon VPC endpoints.

When using a VPC Endpoint, traffic to SES does not transmit over the internet and never leaves the Amazon network in order to securely connect your VPC to SES without availability risks or bandwidth constraints on your network traffic. You can centralize SES across your multi-account infrastructure and provide it as a service to your accounts without the need to utilize an internet gateway.

Limitations

- SES does not support SMTP VPC endpoints in the following Availability Zones: `use1-az2`, `use1-az3`, `use1-az5`, `usw1-az2`, `usw2-az4`, `apne2-az4`, `cac1-az3`, and `cac1-az4`.
- The SMTP endpoint used within the VPC is restricted to the AWS Region currently being used for your account.

You can also use VPC endpoints with Mail Manager ingress endpoints for secure, private email ingestion within your private network infrastructure. See the section called “VPC endpoint configuration” in the Mail Manager chapter.

Walkthrough example of setting up SES in Amazon VPC

Prerequisites

Before you complete the procedure in this section, you have to complete the following steps:

- Have an existing virtual private cloud (VPC) or create a new VPC. For procedures, see Get started with Amazon VPC.
- Launch an Amazon EC2 instance in your VPC for testing connectivity to the VPC endpoint created in a later step. For more information, see Default VPCs.

> **Note**
>
> While VPC endpoints for SES can be used with any resource, for ease of test method, this example will have you use an EC2 instance as the resource. Because Amazon EC2 restricts email traffic over port 25 by default, for SMTP endpoints you'll have to use a different port other than TCP 25, such as TCP 465, 587, 2465, or 2587—for more information, see Restriction on email sent using port 25. For API endpoints use port 443.

Setting up SES in Amazon VPC

The process of setting up a VPC endpoint to use with SES consists of a few separate steps. First, you have to create a security group that allows the instance to communicate with the chosen port(s), then create a VPC endpoint for Amazon SES, and finally, test the connection to the VPC endpoint to ensure that it's configured properly.

Step 1: Create the security group

In this step, you create a security group that lets Amazon EC2 instances communicate with VPC interface endpoint you'll be creating.

To create the security group

1. In the navigation pane of the Amazon EC2 console, under **Network & Security**, choose **Security Groups**.
2. Choose **Create security group**.
3. Under **Basic details**, do the following:

- For **Security group name**, enter a unique name that identifies the security group.
- For **Description**, enter some text that describes the purpose of the security group.
- For **VPC**, choose the VPC that you want to use Amazon SES in.

4. Under **Inbound rules**, choose **Add rule**.
5. For the new **Inbound rule**, do the following:

   - For **Type**, choose **Custom TCP**.
   - For **Port range**, enter the port number that you want to use to send email. For SMTP endpoints, you can use any of the following port numbers: **465**, **587**, **2465**, or **2587**. For API endpoints, use port 443.
   - For **Source type**, choose **Custom**.
   - For **Source**, enter the private IP CIDR range or other Security Group IDs that contain the resources that will use the VPC endpoint to communicate with the SES service.
   - (Repeat steps 4 - 5 for each CIDR range or Security Group you wish to allow access from.)

6. When you finish, choose **Create security group**.

Step 2: Create the VPC endpoint

In Amazon VPC, a *VPC endpoint* lets you connect your VPC to supported AWS services. In this example, you configure Amazon VPC so that your Amazon EC2 security group can connect to Amazon SES.

To create the VPC endpoint

1. Open the Amazon VPC console at https://eusc-de-east-1.console.amazonaws-eusc.eu/vpc/.
2. Under **PrivateLink and Lattice**, choose **Endpoints**.
3. Choose **Create Endpoint** to open the **Create Endpoint** page.
4. (Optional) In the **Endpoint settings** panel, create a tag in the **Name tag** field.
5. For **Service category**, select **AWS services**.
6. In the **Services** panel, for SMTP endpoints, filter on *smtp* in the search bar, then select its radio button. For API endpoints, filter on *email* in the search bar. You can also use a FIPS endpoint by searching for *email-fips*.
7. In the **VPC** panel, click inside the search bar and select a VPC from the list box (see the section called “Prerequisites”).

8. In the **Subnets** panel, select *Availability Zones* and *Subnet IDs*.

> **Note**
>
> Amazon SES doesn't support SMTP VPC endpoints in the following *Availability Zones*: `use1-az2`, `use1-az3`, `use1-az5`, `usw1-az2`, `usw2-az4`, `apne2-az4`, `cac1-az3`, and `cac1-az4`.

9. In the **Security groups** panel, select the security group you created earlier.
10. (Optional) In the **Tags** panel, you can create one or more tags.
11. Choose **Create endpoint**. Wait approximately 5 minutes while Amazon VPC creates the endpoint. When the endpoint is ready to use, the value in the **Status** column changes to *Available*.

(Optional) Step 3: Test the connection to the VPC endpoint

When you complete the process of configuring the VPC endpoint, you can test the connection to ensure that the VPC endpoint is configured properly. You can test the connection by using command-line tools that are included with most operating systems.

To test the connection to the VPC endpoint

1. Launch an Amazon EC2 instance in the same VPC where you just created the email-smtp VPC endpoint.

   For information about connecting to Linux instances, see Connect to your Linux instance in the *Amazon EC2 User Guide*.

   For information about connecting to Windows instances, see the Get started tutorial in the *Amazon EC2 User Guide*.
2. Send a test email. For the SMTP endpoint, use the SES SMTP interface. For the API endpoint, use the SES CLI or API.

Note

You have to verify an email address or domain before you can send email through Amazon SES. For more information about verifying identities, see Creating and verifying identities in Amazon SES.

Troubleshooting Amazon SES issues

This section contains the following topics that may help you when you encounter problems:

- For information about domain verification problems that you might encounter, see Domain and Email address verification problems.
- For solutions to DKIM-related issues, see Troubleshooting DKIM problems in Amazon SES.
- For a list of common delivery problems that you might encounter when you send email, along with corrective actions that you can take, see Amazon SES Delivery problems.
- For a description of issues recipients may see when they receive an email that was sent through Amazon SES, see Problems with emails received from Amazon SES.
- For solutions to problems with bounce, complaint, and delivery notifications, see Amazon SES notification problems.
- For a list of errors that can occur when you send an email with Amazon SES, see Amazon SES email sending errors.
- For tips on how to increase your email sending speed when you make multiple calls to Amazon SES using either the API or the SMTP interface, see Increasing throughput with Amazon SES.
- For solutions to common problems that you might encounter when you use Amazon SES through its Simple Mail Transfer Protocol (SMTP) interface, as well as a list of SMTP response codes that Amazon SES returns, see Amazon SES SMTP issues.
- For a list of common error codes that are returned by the Amazon SES API v2, see Common Errors.
- For a description of common issues related to our sending review process, and how to handle them, see Amazon SES Sending review process FAQs.
- For a discussion about how DNS-based Blackhole Lists (DNSBLs) affect your sending with Amazon SES, see DNS Blackhole List (DNSBL) FAQs.

If you are calling the Amazon SES API directly, see the Amazon Simple Email Service API Reference for the HTTP errors that you might receive.

> **Note**
>
> If you need to request technical support, do not use the feedback link on any of the pages of this developer guide, as the form is received by the AWS Documentation team, not AWS Support. Instead, on the Contact Us page, explore the different support options available.

Contents

- General Amazon SES issues
- Domain and Email address verification problems
- Troubleshooting DKIM problems in Amazon SES
- Amazon SES Delivery problems
- Problems with emails received from Amazon SES
- Amazon SES notification problems
- Amazon SES email sending errors
- Increasing throughput with Amazon SES
- Amazon SES SMTP issues

General Amazon SES issues

The information on this page will explain and help diagnose issues that you may encounter when using Amazon SES.

Changes that I make are not immediately visible

As a service that is accessed through computers in data centers around the world, Amazon SES uses a distributed computing model called eventual consistency. Any change that you make in Amazon SES (or other AWS services) takes time to become visible from all possible endpoints. Some of the delay results from the time it takes to send the data from server to server and from region to region around the world. In the majority of cases, this delay will be no more than a few minutes.

Some areas in which you may notice a delay include:

- **Creating and modifying configuration sets** – When you create or modify a configuration set (for example, if you associate a dedicated IP pool with an existing configuration set), there may be a brief delay from the time that you create or modify it to the time those changes are active.

- **Creating and modifying event destinations** – When you create or modify an event destination (for example, to tell Amazon SES to send your email sending data to another AWS service), there may be a delay between the time your created or modified the event destination and the time email sending events actually arrive at the specified destination.

Domain and Email address verification problems

To verify a domain or an email address with Amazon SES, you initiate the process using either the Amazon SES console or the Amazon SES API. This section contains information that may help resolve issues with the verification process.

> **Note**
>
> In the following procedures, the reference to DNS records could refer to either CNAME or TXT records depending on which form of DKIM you used. Easy DKIM uses CNAME records and Bring Your Own DKIM (BYODKIM) uses TXT records. Detailed verification procedures are provided for each of Easy DKIM or BYODKIM.

Common domain verification problems

If you attempt to verify a domain using the procedure in the section called "Verifying a domain identity" and you encounter problems, review the possible causes and solutions below.

- **You're attempting to verify a domain that you don't own** – You can't verify a domain that you don't own. For example, if you want to send email through Amazon SES from an address on the *gmail.com* domain, you need to verify that email address specifically. You can't verify the entire *gmail.com* domain.
- **You're attempting to verify a private domain** – You can't verify a domain if the DNS records can't be resolved over public DNS.
- **Your DNS provider doesn't allow underscores in the DNS record names** – A small number of DNS providers don't allow you to include underscores (_) in record names. However, the underscore in the DKIM record name is required. If your DNS provider doesn't allow you to enter an underscore in the record name, contact the provider's customer support team for assistance.
- **Your DNS provider appended the domain name to the end of the DNS record** – Some DNS providers automatically append the name of your domain to the attribute name of DNS record.

For example, if you create a record where the attribute name is *_domainkey.example.com*, the provider might append the domain name, resulting in *_domainkey.example.com.example.com*). To avoid duplication of the domain name, add a period to the end of the domain name when you enter the DNS record. This step tells your DNS provider that it isn't necessary to append the domain name to the record.

- **Your DNS provider modified the DNS record value** – Some providers automatically modify DNS record values to use only lowercase letters. Amazon SES only verifies your domain when it detects a verification record for which the attribute value exactly matches the value that Amazon SES provided when you started the domain verification process. If the DNS provider for your domain changes your DNS record values to use only lowercase letters, contact the DNS provider for additional assistance.
- **You want to verify the same domain multiple times** – You might need to verify your domain more than once because you're sending in different regions, or because you're using the same domain to send from multiple AWS accounts. If your DNS provider doesn't allow you to have more than one DNS record with the same attribute name, you might still be able to verify two domains. If your DNS provider allows it, you can assign multiple attribute values to the same DNS record. For example, if your DNS is managed by Amazon Route 53, you can set up multiple values for the same CNAME record by completing the following steps:

  1. In the Route 53 console, choose the CNAME record you created when you verified your domain in the first region.
  2. In the **Value** box, go to the end of the existing attribute value, and then press Enter.
  3. Add the attribute value for the additional region, and then save the record set.

  If your DNS provider doesn't let you to assign multiple values to the same DNS record, you can verify the domain once with *_domainkey* in the attribute name of the DNS record, and another time with *_domainkey* removed from the attribute name. The downside of this solution is that you can only verify the same domain two times.

Checking domain verification settings

You can check that your Amazon SES domain verification DNS record is published correctly to your DNS server by using the following procedure. This procedure uses the nslookup tool, which is available for Windows and Linux. On Linux, you can also use dig.

The commands in these instructions were executed on Windows 7, and the example domain we use is *ses-example.com* configured with Easy DKIM which uses CNAME records.

In this procedure, you first find the DNS servers that serve your domain, and then query those servers to view the CNAME records. You query the DNS servers that serve your domain because those servers contain the most up-to-date information for your domain, which can take time to propagate to other DNS servers.

To verify that your domain verification CNAME records are published to your DNS server

1. Find the name servers for your domain by taking the following steps.

   a. Go to the command line. To get to the command line on Windows 7, choose **Start** and then type **cmd**. On Linux-based operating systems, open a terminal window.

   b. At the command prompt, type the following, where *<domain>* is your domain. This will list all of the name servers that serve your domain.

   ```
   nslookup -type=NS <domain>
   ```

   If your domain was *ses-example.com*, this command would look like:

   ```
   nslookup -type=NS ses-example.com
   ```

   The command's output will list the name servers that serve your domain. You will query one of these servers in the next step.

2. Verify that the CNAME records are correctly published by taking the following steps. *Keep in mind that Amazon SES generates three CNAME records for Easy DKIM authentication, so repeat the following procedures for each of the three.*

   a. At the command prompt, type the following, where *<random string>* is the SES generated CNAME name, *<domain>* is your domain, and *<name server>* is one of the name servers you found in step 1.

   ```
   nslookup -type=CNAME <random string>._domainkey.<domain> <name server>
   ```

   In our *ses-example.com* example, if a name server that we found in step 1 was called *ns1.name-server.net*, and the *<random string>* generated by SES is `4hzwn5lmznmmjyl2pqf2agr3uzzzzxyz`, we would type the following:

```
nslookup -type=CNAME 4hzwn5lmznmmjyl2pqf2agr3uzzzzxyz_domainkey.ses-example.com
 ns1.name-server.net
```

b. In the output of the command, verify that the string that follows `canonical name =` matches the CNAME value you see when you choose the domain in the Identities list of the Amazon SES console.

   In our example, we are looking for a CNAME record under `4hzwn5lmznmmjyl2pqf2agr3uzzzzxyz_`*domainkey.ses-example.com* with a value that matches the CNAME value shown for the domain in the Identities list of the Amazon SES console. The hosted zone portion of this value is the `SigningHostedZone` returned for your identity and varies by AWS Region. If the record is correctly published, and the hosted zone for our identity is `dkim.us-west-2.amazonses.com`, we would expect the command to have the following output:

```
4hzwn5lmznmmjyl2pqf2agr3uzzzzxyz_domainkey.ses-example.com canonical name =
 "4hzwn5lmznmmjyl2pqf2agr3uzzzzxyz.dkim.us-west-2.amazonses.com"
```

Common email verification problems

- **The verification email didn't arrive** – If you complete the procedures in Verifying an email address identity but you don't receive the verification email within a few minutes, complete the following steps:
  - Check the spam or junk mail folder for the email address you're attempting to verify.
  - Confirm that the address that you're trying to verify is able to receive email. Using a separate email address (such as your personal email address), send a test email to the address that you want to verify.
  - Check the list of verified addresses in the Amazon SES console. Make sure that there aren't any errors in the email address that you're attempting to verify.

Troubleshooting DKIM problems in Amazon SES

This section lists some of the problems that you may encounter when you configure DKIM authentication in Amazon SES. If you attempt to set up DKIM and you encounter problems, review the possible causes and solutions below.

You set up DKIM successfully, but your messages aren't being DKIM-signed

If you used Easy DKIM or BYODKIM to configure DKIM for a domain, but the messages that you send aren't DKIM-signed, do the following:

- Make sure that DKIM is enabled for the appropriate identity. To enable DKIM for an identity in the Amazon SES console, choose the email domain in the **Identities** list. On the details page for the domain, expand **DKIM**, and then choose **Enable** to enable DKIM.
- Make sure that you're not sending from a verified email address on the same domain. If you set up DKIM for a domain, then all of the messages that you send from that domain are DKIM-signed, *except* for email addresses that you verified individually. Individually verified email addresses use separate settings. For example, if you configured DKIM for the domain *example.com*, and you separately verified the email address *mary@example.com* (but didn't configure DKIM for the address), then emails that you send from *mary@example.com* are sent without DKIM authentication. You can resolve this issue by deleting the email address identity from the list of identities for your account.
- If you use the same identity in more than one AWS Region, you have to configure DKIM for each region separately. Similarly, if you use the same domain with more than one AWS account, you have to configure DKIM for each account. If you remove the necessary DNS records for a specific region or account, Amazon SES disables DKIM signing in that region or account. If DKIM signing becomes disabled, Amazon SES sends you a notification by email.

Your domain's DKIM details in the Amazon SES console show *DKIM: waiting on sender verification... DKIM Verification Status: pending verification.*

If you complete the procedures in Easy DKIM or BYODKIM - Bring Your Own DKIM to configure DKIM for a domain, but the Amazon SES console still indicates that DKIM verification is pending, do the following:

- Wait up to 72 hours. In rare cases, it can take time for the DNS records to become visible to Amazon SES.
- Confirm that the CNAME record (for Easy DKIM) or the TXT record (for BYODKIM) uses the correct name. Some DNS providers automatically append the domain name to records that you create. For example, if you create a record with a Name of `example._domainkey.example.com`, your DNS provider might add the name of your domain to the end of this string, resulting in `example._domainkey.example.com.example.com`. For more information, see the documentation for your DNS provider.

You receive an email from Amazon SES that says your DKIM setup has been (or will be) revoked.

This means that Amazon SES can no longer find the required CNAME records (if you used Easy DKIM) or the required TXT record (if you used BYODKIM) records on your DNS server. The notification email will inform you of the length of time in which you must re-publish the DNS records before your DKIM setup status is revoked and DKIM signing is disabled. If your DKIM setup is revoked, you must restart the DKIM set-up procedure from the beginning.

When attempting to set up BYODKIM, the DKIM verification process fails.

Make sure that your private key uses the right format. The private key has to be in either PKCS #1 or PKCS #8 format and use either 1024 or 2048 bit RSA encryption. Additionally, the private key has to be base64 encoded.

While setting up BYODKIM, you receive a `BadRequestException` error when you try to specify a public key for the domain.

If you receive a `BadRequestException` error, do the following:

- Make sure that the selector that you specify for the public key contains at least 1 and less than or equal to 63 alphanumeric characters. The selector can't include periods or other symbols or punctuation.
- Make sure that you've removed the header and footer lines from the public key, and that you've removed all of the line breaks from the public key.

When using Easy DKIM, your DNS servers successfully return the Amazon SES DKIM CNAME records, but return `SERVFAIL` for the domain verification TXT record.

Your DNS provider might not be able to redirect CNAME records. Amazon SES and ISPs query for TXT records. To comply with the DKIM specification, your DNS servers have to be able to respond to TXT record queries as well as CNAME record queries. If your DNS provider isn't able to respond to TXT record queries, an alternative is to use Route 53 as your DNS hosting provider.

Your emails contain two DKIM signatures

The extra DKIM signature, which contains `d=amazonses.com`, is automatically added by Amazon SES. You can ignore it.

> **Note**
>
> Microsoft Enterprise Outlook randomly selects one DKIM signature for validation while ignoring the other. This behavior does not appear to correlate with the order

in which signatures are arranged, making procedural workarounds ineffective. SES cannot discontinue signing emails with `amazonses.com` DKIM signatures because they are required for complaint feedback loops. Microsoft has acknowledged this as a known issue, attributing it to DNS resolution problems within their infrastructure. These DNS resolution failures would explain why DKIM signature validation appears to fail randomly, given that public keys are retrieved from DNS records.

Amazon SES Delivery problems

After you make a successful request to Amazon SES, your message is often sent immediately. At other times, there might be a short delay. In any case, you can be assured that your email will be sent.

When Amazon SES sends your message, however, several factors can prevent it from being delivered successfully, and in some cases you will become aware that delivery failed only when the message you send does not arrive. Use the following process to resolve this situation.

If an email does not arrive, try the following:

- Verify that you made a `SendEmail` or `SendRawEmail` request for the email in question and that you received a successful response. If you are making these requests programmatically, check your software logs to ensure that the program made the request and received a successful response.
- Read the blog article Three places where your email could get delayed when sending through SES because the problem might actually be a delay rather than a nondelivery.
- Check the sender's email address (the "From" address) to verify that it is valid. Also check the Return-Path address, which is where bounce messages are sent. If your mail bounced, there will be an explanatory error message there.
- Check the AWS Service Health Dashboard to confirm that there is not a known problem with Amazon SES.
- Contact the email recipient or the recipient's ISP. Verify that the recipient is using the correct email address, and inquire whether there have been any known delivery problems with the recipient's ISP. Also, determine whether the email did arrive but was filtered as spam.
- If you have signed up for a paid AWS Support Plan, you can open a new technical support case. In your correspondence with us, please provide any relevant recipient addresses, along with any request IDs or message IDs returned from the `SendEmail` or `SendRawEmail` responses.

- Wait to see if the problem is actually a delay, not a permanent delivery failure. To combat spammers, some ISPs temporarily reject incoming messages from unknown sending mail servers. This process, called *greylisting*, can cause a delay in delivery. Amazon SES will retry these messages. If greylisting is the issue, the ISP might accept the email on one of these retry attempts.
- Even when you have your customers' best interests in mind, you may still encounter situations that impact the deliverability of your messages. See the section called “Maintaining a positive sender reputation” to help ensure that your email communications reach your intended audience.

Problems with emails received from Amazon SES

This section discusses some common issues that you might see when you receive emails that were sent from Amazon SES.

The email client displays "sent via amazonses.com" as the source of the email

Some email clients display the "via" domain when the sender's domain doesn't match the domain that the email was sent from (in this case, *amazonses.com*). For more information, see Extra info next to sender's name on the Gmail Support website. Alternatively, you can set up DomainKeys Identified Mail (DKIM). When you authenticate your emails using DKIM, email clients typically don't show the "via" domain because the DKIM signature shows that the email is from the domain that it claims to be from. For information about setting up DKIM, see Authenticating Email with DKIM in Amazon SES.

> **Note**
>
> If you've received spam or other unsolicited email messages from an SES user, use the spam reporting tools in your email client, and follow the steps to report SES email abuse listed under Contacting Us.

The message contains garbled or nonsense characters

If your message includes characters that aren't in the ASCII character set (such as accented Latin characters, Chinese characters, or Arabic characters), you have to encode those characters using HTML character encoding. You can use web-based tools to encode the characters in your email, such as the HTML Character Convertor on the Email On Acid website.

Alternatively, you can assemble the MIME message yourself. In the MIME message, you can specify that the message should use UTF-8 encoding. When you use UTF-8 encoding, you can use non-ASCII characters directly in your messages. When you've finished creating the MIME message, you can send it using the SendRawEmail API or the SendMail API v2.

One common cause of this issue is the Smart Quotes feature of Microsoft Word. If you often copy content from Word and paste it into your emails, you might encounter this issue. The Smart Quotes feature replaces straight quote characters ("...") with curly quote characters (“...”). Curly quote characters aren't standard ASCII characters. As a result, they might be rendered in some email clients as "??" or as a group of characters such as "â€œ". To correct this issue, you can disable the Smart Quotes feature in Word. Alternatively, you can use the SendRawEmail solution from the preceding paragraph. To learn how to disable this feature, see Smart quotes in Word on the Microsoft Office Support website.

Amazon SES notification problems

If you encounter a problem with bounce, complaint, or delivery notifications, review the possible causes and solutions below.

- **You receive bounce notifications via Amazon SNS, but you don't know which recipients the notifications correspond to**—In the future, to associate a bounce notification with a given recipient, you have the following options:
  - Since Amazon SES doesn't retain any custom message IDs that you have added, store a mapping between an identifier and the Amazon SES message ID that Amazon SES passes back to you when it accepts the email.
  - In each call to Amazon SES, send to a single recipient, rather than sending a single message to multiple recipients.
  - You can enable feedback forwarding via email, which will forward the full bounce message to you.
- **You receive complaint or delivery notifications via Amazon SNS or email feedback forwarding, but you don't know which recipients the notifications correspond to**—Some ISPs redact the complained recipient's email address before passing the complaint notification to Amazon SES. To enable you to find the recipient's email address, your best option is to store your own mapping between an identifier and the Amazon SES message ID that Amazon SES passes back to you when it accepts the email. Note that Amazon SES does not retain any custom message IDs that you add.

- **You want to set up notifications to go to an Amazon SNS topic you don't own**—The owner of that topic must configure an Amazon SNS access policy that allows your account to call the `SNS:Publish` action on their topic. For information about how to control access to your Amazon SNS topic through the use of IAM policies, see Managing Access to Your Amazon SNS Topics in the *Amazon Simple Notification Service Developer Guide*.

Amazon SES email sending errors

This topic reviews the types of email sending-specific errors that you may encounter when you send an email through Amazon SES. If you try to send an email through Amazon SES and the call to Amazon SES fails, Amazon SES returns an error message to your application and does not send the email. The way that you observe this error message depends on the way that you call Amazon SES.

- If you call the Amazon SES API directly, the Query action will return an error. The error may be `MessageRejected` or one of the errors specified in the Common Errors topic of the *Amazon Simple Email Service API Reference*.
- If you call Amazon SES using an AWS SDK that uses a programming language that supports exceptions, Amazon SES may throw an exception. The type of exception depends on the SDK and on the error. For example, the exception could be an Amazon SES `MessageRejectedException` (the actual name may vary depending on the SDK) or a general AWS exception. Regardless of the type of exception, the error type and the error message in the exception will give you more information.
- If you call Amazon SES through its SMTP interface, the way that you experience the error depends on the application. Some applications might display a specific error message, and others might not. For a list of SMTP response codes that Amazon SES returns, see SMTP response codes returned by Amazon SES.

> **Note**
>
> When your call to Amazon SES to send an email fails, you are not billed for that email.

The following are the types of Amazon SES-specific problems that can cause Amazon SES to return an error when you try to send an email. These errors are in addition to general AWS errors like

`MalformedQueryString` as specified in the Common Errors topic of the *Amazon Simple Email Service API Reference*.

- **Email address is not verified. The following identities failed the check in region *region*: *identity1*, *identity2*, *identity3***—You are trying to send email from an email address or domain that you have not verified with Amazon SES. This error could apply to the "From", "Source", "Sender", or "Return-Path" address. If your account is still in the Amazon SES sandbox, you also must verify every recipient email address except for the recipients provided by the Amazon SES mailbox simulator. If Amazon SES is not able to show all of the failed identities, the error message ends with an ellipsis.

  > **Note**
  >
  > Amazon SES has endpoints in multiple AWS Regions, and email address verification status is separate for each AWS Region. You must complete the verification process for each sender in the AWS Regions you want to use.

- **Account is paused**—Your account's ability to send email is paused. You can still access the Amazon SES console and perform most operations. However, if you try to send an email, you receive this message.

  If we pause your account's ability to send email, we automatically send a notification to the email address associated with your AWS account. For more information, see the section called “Sending review process FAQs”.

- **Throttling**—Your application may be trying to send too many messages per second, or you may have sent too much email over the last 24 hours. In these cases, the error message may be similar to the following examples:

  - **Daily message quota exceeded**—You have sent the maximum number of messages that you are permitted in a 24-hour period. If you have exceeded your daily quota, you will have to wait until the next 24-hour period before you can send more email.
  - **Maximum sending rate exceeded**—You are attempting to send more emails per second than is permitted by your maximum send rate. If you have exceeded your sending rate, you can continue to send email, but will need to reduce your send rate. For more information, see How to handle a "Throttling - Maximum sending rate exceeded" error on the AWS Messaging and Targeting Blog.
  - **Maximum SigV2 SMTP sending rate exceeded**—You are attempting to send messages using SMTP credentials created before January 10, 2019; your SMTP credentials were created using

an older version of the AWS Signature. For security purposes, you should delete credentials that you created before this date, and replace them with newer credentials. You can delete older credentials by using the IAM console. For more information, see the section called "Obtaining SMTP credentials" for creating credentials.

You should regularly monitor your sending activity to see how close you are to your sending quotas. For more information, see Monitoring your Amazon SES sending quotas. For general information about sending quotas, see Managing your Amazon SES sending limits. For information about how to increase your sending quotas, see Increasing your Amazon SES sending quotas.

> **Important**
>
> If the error text that explains the throttling error is not related to you exceeding your daily quota or maximum send rate, then there might be a system-wide problem that is causing reduced sending capabilities. For information about the service status, go to the AWS Service Health Dashboard.

- **There are no recipients specified**—No recipients were provided.
- **There are non-ASCII characters in the email address**—The email address string must be 7-bit ASCII. If you want to send to or from email addresses that contain Unicode characters in the domain part of an address, you must encode the domain using Punycode. Punycode is not permitted in the local part of the email address (the part before the @ sign) nor in the "friendly from" name. If you want to use Unicode characters in the "friendly from" name, you must encode the "friendly from" name using MIME encoded-word syntax, as described in Sending raw email using the Amazon SES API v2. For more information about Punycode, see RFC 3492.
- **Mail FROM domain is not verified**—Amazon SES could not read the MX record required to use the specified MAIL FROM domain. For information setting up custom MAIL FROM domains, see Using a custom MAIL FROM domain.
- **Configuration set does not exist**—The configuration set that you specified does not exist. A configuration set is an optional parameter that you use to publish email sending events. For more information, see Monitor email sending using Amazon SES event publishing.
- **Server errors and timeouts**—On rare occasions, a `SendEmail` or `SendRawEmail` request may time out or return a server error after the email has already been accepted for delivery. If your application retries the request, a second email with a different message ID may be sent.

To reliably correlate event notifications with your sending logic, use message tags rather than relying solely on the message ID.

Increasing throughput with Amazon SES

When you send emails, you can call Amazon SES as frequently as your maximum send rate allows. (For more information about your maximum send rate, see Managing your Amazon SES sending limits.) However, each call to Amazon SES takes time to complete.

If you are making multiple calls to Amazon SES using the Amazon SES API or the SMTP interface, you may want to consider the following tips to help you improve your throughput:

- **Measure your current performance to identify bottlenecks**—A possible performance test involves sending multiple test emails as quickly as possible within a code loop in your application. Measure the round-trip latency of each `SendEmail` request. Then, incrementally launch additional instances of the application on the same machine, and watch for any impact on network latency. You may also want to run this test on multiple machines and on different networks to help pinpoint any possible machine resource bottlenecks or network bottleneck that may exist.
- **(API only) Consider using persistent HTTP connections**—Rather than incurring the overhead of establishing a separate new HTTP connection for each API request, use persistent HTTP connections. That is, reuse the same HTTP connection for multiple API requests.
- **Consider using multiple threads**—When an application uses a single thread, the application code calls the Amazon SES API and then synchronously waits for an API response. Sending emails is typically an I/O-bound operation, and doing the work from multiple threads provides better throughput. You can send concurrently using as many threads of execution as you wish.
- **Consider using multiple processes**—Using multiple processes can help increase your throughput because you will have more concurrent active connections to Amazon SES. For example, you can segment your intended emails into multiple buckets, and then run multiple instances of your email sending script simultaneously.
- **Consider using a local mail relay**—Your application can quickly transmit messages to your local mail server, which can then help to buffer the messages and asynchronously transmit them to Amazon SES. Some mail servers support delivery concurrency, which means that even if your application is generating emails to the mail server in a single-threaded fashion, the mail server will use multiple threads when sending to Amazon SES. For more information, see Integrating Amazon SES with your existing email server.

- **Consider hosting your application closer to the Amazon SES API endpoint**—You may wish to consider hosting your application in a data center close to the Amazon SES API endpoint, or on an Amazon EC2 instance in the same AWS Region as the Amazon SES API endpoint. This can help to decrease network latency between your application and Amazon SES, and improve throughput. For a list of regions where Amazon SES is available, see Amazon Simple Email Service (Amazon SES) in the *AWS General Reference*.
- **Consider using multiple machines**—Depending on the system configuration on your host machine, there may be a limit on the number of simultaneous HTTP connections to a single IP address, which may limit the benefits of parallelism once you exceed a certain number of concurrent connections on a single machine. If this is a bottleneck, you may wish to consider making concurrent Amazon SES requests using multiple machines.
- **Consider using the Amazon SES query API instead of the SMTP endpoint**—Using the Amazon SES query API enables you to submit the email send request using a single network call, whereas interfacing with the SMTP endpoint involves an SMTP conversation which consists of multiple network requests (for example, EHLO, MAIL FROM, RCPT TO, DATA, QUIT). For more information about the Amazon SES query API, see Using the Amazon SES API to send email.
- **Use the Amazon SES mailbox simulator to test your maximum throughput**—To test any changes you may implement, you can use the mailbox simulator. The mailbox simulator can help you to determine your system's maximum throughput without using up your daily sending quota. For information about the mailbox simulator, see Using the mailbox simulator manually.

If you are accessing Amazon SES through its SMTP interface, see Amazon SES SMTP issues for specific SMTP-related issues that may affect throughput.

Amazon SES SMTP issues

This section contains solutions for several common issues related to sending email through the Amazon SES Simple Mail Transfer Protocol (SMTP) interface. It also contains a list of SMTP response codes that Amazon SES returns.

To learn more about sending email through the Amazon SES SMTP interface, see Using the Amazon SES SMTP interface to send email.

- **You can't connect to the Amazon SES SMTP endpoint.**

  Problems connecting to the Amazon SES SMTP endpoint are most commonly related to the following issues:

- **Incorrect credentials** – The credentials that you use to connect to the SMTP endpoint are different from your AWS credentials. To obtain your SMTP credentials, see Obtaining Amazon SES SMTP credentials. For more information about credentials, see Types of Amazon SES credentials.
- **Network or firewall issues** – Your network might be blocking outbound connections over the port you're trying to send email from. To determine if an issue on your local network is causing connection issues, type the following command at the command line, replacing *port* with the port you're trying to use (typically 465, 587, 2465, or 2587): `telnet email-smtp.us-west-2.amazonaws.com port`

  If you are able to connect to the SMTP server using this command, and you are trying to connect to Amazon SES using TLS Wrapper or STARTTLS, complete the procedures shown in Testing your connection to the Amazon SES SMTP interface using the command line.

  If you can't connect to the Amazon SES SMTP endpoint using `telnet` or `openssl`, it indicates that something in your network (such as a firewall) is blocking outbound connections over the port you're trying to use. Work with your network administrator to diagnose and fix the problem.

- **You're sending to Amazon SES from an Amazon EC2 instance using port 25, and you're receiving timeout errors.**

  Amazon EC2 restricts port 25 by default. To remove these restrictions, submit an Amazon EC2 Request to Remove Email Sending Limitations. You can also connect to Amazon SES using ports 465 or 587, neither of which is restricted.

- **Network errors are causing dropped emails.**

  Ensure that your application uses retry logic when it connects to the Amazon SES SMTP endpoint, and that your application can detect and retry message delivery in case of a network error. SMTP is a verbose protocol, and sending an email using this protocol requires several network round trips. Because of the nature of SMTP, the potential for network errors increases.

- **You lose connection with the SMTP endpoint.**

  Lost connections are most commonly caused by the following issues:

  - **MTU size** – If you receive a time-out error message, the Maximum Transmission Unit (MTU) of the network interface for the computer you're using to connect to the Amazon SES SMTP interface may be too large. To resolve this issue, set the MTU size on that computer to 1500 bytes.

For more information about setting the MTU size on Windows, Linux, and macOS operating systems, see Queries Appear to Hang in the Client and Do Not Reach the Cluster in the *Amazon Redshift Management Guide*.

For more information about setting the MTU size for an Amazon EC2 instance, see Network Maximum Transmission Unit (MTU) for Your EC2 Instance in the *Amazon EC2 User Guide*.

- **Long-lived connections** – The Amazon SES SMTP endpoint runs on a fleet of Amazon EC2 instances behind an Elastic Load Balancer (ELB). In order to ensure that the system is up-to-date and fault tolerant, active Amazon EC2 instances are periodically terminated and replaced with new instances. Because your application connects to an Amazon EC2 instance through the ELB, the connection becomes invalid when the Amazon EC2 instance is terminated. You should establish a new SMTP connection after you have delivered a fixed number of messages via a single SMTP connection, or if the SMTP connection has been active for some amount of time. You will need to experiment to find appropriate thresholds depending on where your application is hosted and how it submits email to Amazon SES.
- **You want to know the IP addresses of the Amazon SES SMTP mail servers so that you can allowlist the IP addresses with your network.**

  The IP addresses for the Amazon SES SMTP endpoints reside behind load balancers. As a result, these IP addresses change frequently. It's not possible to provide a definitive list of all of the IP addresses for the Amazon SES endpoints. We recommend that you allowlist the `amazonses.com` domain, rather than allowlisting individual IP addresses.

SMTP response codes returned by Amazon SES

This section contains a list of response codes that the Amazon SES SMTP interface returns.

You should retry SMTP requests that receive 400 errors. We recommend that you implement a system that retries requests with progressively longer wait times (for example, wait 5 seconds before retrying, then wait 10 seconds, and then wait 30 seconds). If the third retry doesn't succeed, wait 20 minutes, and then repeat the process. To see an example of an implementation that uses an exponential retry policy, see How to handle a "Throttling - Maximum sending rate exceeded" error on the AWS Messaging and Targeting Blog.

> **Note**
>
> AWS SDKs implement retry logic automatically, but they use the HTTPS interface instead of SMTP.

If you receive a 500 error, you have to revise your request to correct an issue before you submit the request again. For example, if your AWS authentication credentials are invalid, you have to update your application to use the correct credentials before you submit your request again.

Description	Response code	More information
Authentication successful	`235 Authentication successful`	Your SMTP client successfully connected and signed in to the SMTP server.
Successful delivery	`250 Ok` *`MessageID`*	*`MessageID`* is a unique string of characters that Amazon SES uses to identify a message.
Service unavailable	`421 Too many concurrent SMTP connections`	Amazon SES can't process the request because there are currently too many connections to the SMTP server.
Local processing error	`451 Temporary service failure`	Amazon SES couldn't process the request. There might be issues with the request that prevent it from being processed .
Timeout	`451 Timeout waiting for data from client`	Too much time elapsed between requests, so the SMTP server closed the connection.
Daily sending quota exceeded	`454 Throttling failure: Daily message quota exceeded`	You've exceeded the maximum number of emails that Amazon SES permits you to send in

Description	Response code	More information
		a 24-hour period. For more information, see Managing your Amazon SES sending limits.
Maximum send rate exceeded	`454 Throttling failure: Maximum sending rate exceeded`	You've exceeded the maximum number of emails that Amazon SES permits you to send per second. For more information, see Managing your Amazon SES sending limits.
Amazon SES issue when validating SMTP credentials	`454 Temporary authentication failure`	Issues that could cause this issue include (but aren't limited to): • There is a problem with the encryption between your email-sending application and Amazon SES. Note that you have to use an encrypted connection when you connect to Amazon SES. For more information, see Connecting to an Amazon SES SMTP endpoint. • Amazon SES could be experiencing an issue. Check the AWS Service Health Dashboard for updates.
Problem receiving the request	`454 Temporary service failure`	Amazon SES didn't successfully receive the request. As a result, the message wasn't sent.

Description	Response code	More information
Incorrect credentials	`530 Authentication required`	The application that you use to send email didn't attempt to authenticate when it connected to the Amazon SES SMTP interface.
Authentication Credentials Invalid	`535 Authentication Credentials Invalid`	The application that you use to send email didn't provide the correct SMTP credentials to Amazon SES. Note that your SMTP credentials aren't the same as your AWS credentials. For more information, see Obtaining Amazon SES SMTP credentials.
Account not subscribed to Amazon SES	`535 Account not subscribed to SES`	The AWS account that owns the SMTP credentials is not signed up for Amazon SES.
Message is too long	`552 Message is too long.`	The message that you're trying to send is larger than the maximum message size.
Account not subscribed to Amazon SES	`535 Account not subscribed to SES`	The AWS account that owns the SMTP credentials is not signed up for Amazon SES.
MAIL FROM syntax error	`553 <email-address > Invalid email address`	There is a syntax error in the MAIL FROM part of the SMTP message. Please check that you are following the correct format and don't forget to enclose the email-address in '<>'.

Description	Response code	More information
RCPT TO syntax error	`553 <email-address > address unknown`	There is a syntax error in the RCPT TO part of the SMTP message. Please check that you are following the correct format and don't forget to enclose the email-address in '<>'.
User not authorized to call the Amazon SES SMTP endpoint	`554 Access denied: User UserARN is not authorized to perform ses:SendRawEmail on resource IdentityARN`	The AWS Identity and Access Management (IAM) policy or the Amazon SES sending authorization policy of the user who owns the SMTP credentials isn't allowed to call the Amazon SES SMTP endpoint.

Description	Response code	More information
Unverified email address	`554 Message rejected: Email address is not verified. The following identities failed the check in region region: identity0 , identity1 , identity2`	You're trying to send email from an email address or domain that isn't verified to send email from your Amazon SES account. This error could apply to the "From", "Source", "Sender", or "Return-Path" addresses. If your account is still in the sandbox, you also have to verify every recipient email address (except for the recipients provided by the Amazon SES mailbox simulator). If Amazon SES isn't able to show all of the identities that failed the verification check, the error message ends with three periods (...). **Note** Amazon SES has endpoints in several AWS Regions, and email address verification status is separate for each AWS Region. You have to complete the verification process for each sender in the AWS Regions that you want to use.

Note

For SMTP issues that aren't addressed by the troubleshooting on this page, try the SES support options listed under Contacting Us.

Amazon SES frequently asked questions (FAQs)

This section contains answers to several frequently asked questions related to using Amazon SES.

This section contains FAQs for the following topics:

- Dedicated IP addresses (managed) FAQs
- Amazon SES Sending review process FAQs
- DNS Blackhole List (DNSBL) FAQs
- Amazon SES email sending metrics FAQs

Dedicated IP addresses (managed) FAQs

While Dedicated IP addresses (managed) offers numerous automated features for dedicated IP management, scaling, and warmup, there has been some misunderstanding about the extent of this automation and SES' responsibilities. It would be incorrect to assume that "managed" means SES completely handles all aspects of IP reputation and listing issues. To clarify these misconceptions, we need to emphasize that while the service automates technical aspects like scaling and warmup, you remain responsible for maintaining your sending reputation and managing any reputation related issues such as getting listed in a Reputation Block List (RBL).

These FAQs address common misconceptions about the feature's scope and clarify the shared responsibility model between you and SES. These FAQS highlight that while the "managed" aspect refers to technical infrastructure management, you must still actively monitor and maintain your sending reputation, keep bounce rates low, and handle most RBL delisting requests yourself.

Q1. Can I ask SES to remove my dedicated IP address (managed) from being listed in an RBL?

If your dedicated IP address (managed) is listed in any RBL of recipient mail providers, it is not the responsibility of SES and you must request removal yourself directly to the RBL administrator. It's important to monitor your dedicated IPs (managed) by tracking both bounce notifications and SMTP response messages to identify blocks. This monitoring helps protect your email sending reputation and allows you to quickly address any RBL incidents that may occur, ensuring consistent email deliverability.

Q2. Can I ask SES to allocate a new dedicated IP address (managed) to replace a current one that is listed in an RBL that is not offered by Spamhaus?

No. SES does not rotate dedicated IP addresses. As you are responsible for your dedicated IP addresses, managed or standard, you need to figure out why they are getting listed in the RBL and get them delisted yourself.

Q3. Can SES monitor the bounce rate of a dedicated IP address (managed) assigned to my account and rotate the address when the bounce rate becomes high?

No. SES does not rotate dedicated IP addresses when an account experiences a high bounce rate. When you lease a dedicated IP address (managed), only your account has exclusive rights to send email through that dedicated IP address, thus SES cannot monitor your account. It is your responsibility to manage your sender reputation and email components, including managing complaints and maintaining a bounce rate below 2%.

Q4. I just leased a dedicated IP address (managed) and the email sent through it is bouncing because the address is on an RBL. Does SES check the reputation of an dedicated IP address (managed) before leasing it to an account?

Yes. SES does reset any dedicated IP address (managed) (30 days) before leasing it to an SES account. The reputation typically resets to most major providers. SES makes sure that the address is not on any RBL, like Spamhaus; however, SES does not monitor all RBLs available, such as smaller, regional RBLs. If you have any concerns about an RBL due to B2B focus or a regional provider focus that uses those domains, you would need to review the reputation state of the dedicated IP address (managed) yourself.

Q5. If SES does not take action when dedicated IP addresses (managed) get listed in an RBL, other than Spamhus, why should I use them?

Dedicated IP addresses (managed) are managed in terms of auto-scaling by adding and removing IP addresses based on traffic. Also, it saves you time in terms of per-ISP warm-up management;

thus, a managed pool keeps track of how many emails can go out through every IP address based on historic sending patterns. More benefits can be found in the section called “Benefits and features”.

Q6. How can I track dedicated IP addresses (managed) leased to my account?

You can use an SES configuration set with an event publishing destination defined for either a Amazon Data Firehose or an Amazon SNS topic. SES delivery events include the tag `ses:outgoing-ip`. Thus, if an email was bounced due to the reputation of a dedicated IP address (managed), you can find the offending dedicated IP address (managed) in the bounce event's `ses:outgoing-ip` tag.

Amazon SES Sending review process FAQs

We monitor the email that's sent through Amazon SES to make sure that the service isn't being used to deliver malicious, unsolicited, or low-quality email. If we determine that a user is sending content that falls into one of these categories, we take actions on that account. We call this process our *sending review process*.

In many cases, when we detect an issue with an account, we place that account under review. In other cases, we pause the account's ability to send email. We take these actions to protect each account's sender reputation, and to prevent other SES users from experiencing service interruptions and deliverability issues.

Contents

- Account under review FAQ
- Sending pause FAQ
- Bounce FAQ
- Complaint FAQ
- Spamtrap FAQ
- Manual investigation FAQ

Account under review FAQ

Q1. I received a message stating that my account is under review. What does that mean?

We've detected an issue related to the email sent from your account, and we're giving you time to fix it. You can continue to send email as you normally would, but you should also correct the issue that caused your account to be placed under review. If you don't correct the issue before the review period is over, we might pause your ability to send additional email.

Q2. Will I always be notified if my account is placed under review?

Yes. You'll receive a notification at the email address associated with your AWS account.

Q3. Why didn't I receive a notification that my account is under review?

When your account is placed under review, we automatically send a notice to the email address associated with your AWS account. This email address is the one you specified when you created your AWS account. In some cases, this email address may be different from the one you use to send email using SES.

We recommend that you monitor your sender reputation by regularly viewing your Reputation metrics. You can also set up automated alarms in Amazon CloudWatch. These alarms can send you a notification when your reputation metrics exceed certain thresholds. You can also configure Amazon CloudWatch to contact you in other ways, such as by sending a text message to your mobile phone.

Q4. Will the fact that my SES account is under review impact my use of other AWS services?

You'll still be able to use other AWS services while your SES account is under review. However, if you request a service quota increase for another AWS service that sends outbound communications (such as Amazon SNS), that request may be denied until the review period for your SES account is lifted.

Q5. What should I do if my account is under review?

You should do the following:

- If your situation allows it, stop sending mail until you fix the problem. You can still send email while your account is under review. However, if you continue to send mail without making changes, you might inadvertently make the issue worse.
- Look at the email you received from us for a summary of the issue.
- Investigate your sending to determine what aspect of your sending specifically triggered the issue.
- After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future.
- Be sure to provide any information we specifically request. We need this information to evaluate your case.

Q6. How do I request a review?

You can request that we review our decision to place your account under review. To request a review, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf.

In your request, provide the following information:

- Information about the root cause of the event that caused your account to be placed under review.
- A list of the changes that you've made to correct the issue. Only include the steps you've already implemented, not the steps you plan to implement in the future.
- Information about how these changes prevent the same issue from occurring again in the future.

Depending on the nature of the event that led us to place your account under review, we might require additional information. See the FAQ topic associated with the issue you experienced for a list of the information you should include in your request.

Q7. What if my review request isn't accepted?

We'll respond to your request with information about why we didn't accept it. In some cases, you'll be able to submit another request if you're able to demonstrate that you resolved the issue, and that your changes prevent the issue from occurring again in the future.

Q8. Can you help me diagnose the problem?

Typically we can give you only a high-level overview of your issue (for example, that you have a problem with bounces). You'll need to investigate the root cause on your end.

Q9. How will I know if my account is no longer under review?

Reputation metrics includes information about the current status of your account. For more information, see Using reputation metrics to track bounce and complaint rates.

Q10. Do you place my account under review every time there's a problem?

No. In some situations, we might pause your account's ability to send email without first placing your account under review. For example:

- If the issue is very serious.
- If your account has been placed under review for the same issue multiple times in the past. For this reason, it's important to address the underlying problem rather than just resolve the specific incident that led to your account being placed under review. For instance, if a particular campaign caused us to place your account under review, you have to do more than simply stop that campaign. You should determine which properties of the campaign were problematic and ensure that you have processes in place so that your future campaigns don't have the same issue.

In either of these situations, we automatically send you a notification when we pause your account's ability to send email.

Q11. What if I make my fixes shortly before the review period expires?

Sign into the AWS Management Console and go to Support Center. Reply to the case we opened on your behalf. In your reply to the case, let us know that you've resolved the issue.

Q12. Can I get help from my AWS representative or Premium Support?

If you're already working with an AWS account representative, we'll automatically contact him or her when your account is placed under review. Your account representative may be able to provide additional information to help you better understand the issue. If you use Premium Support, you should also contact that team for additional help.

Sending pause FAQ

Q1. I received a message stating that my account's ability to send email is paused. What does that mean?

We paused your account's ability to send email because of a critical issue with emails you sent. Most often, we pause accounts for one of the following reasons:

- We previously placed your account under review. The issues that caused us to place your account under review weren't corrected before the end of the review period, so we paused your account's ability to send email.
- We've placed your account under review several times for the same issue.
- Your account sent email that violated the AWS Service Terms. If these violations are serious, we might pause your account's ability to send email without placing your account under review first.

Q2. Will I always be notified if my account's ability to send email is paused?

Yes. You'll receive a notification at the email address associated with your AWS account.

Q3. My account's ability to send email is paused. Why didn't I receive a notification?

When we pause an account's ability to send email, we automatically send a notification to the email address associated with that account.

> **Note**
>
> When you create your AWS account, you must provide an email address. You can change this address at any time. For more information about changing the address associated with your AWS account, see Managing an AWS Account in the *AWS Billing and Cost Management User Guide*.

You can use Amazon CloudWatch to create alarms that inform you when your bounce and complaint rates are too high. Creating an alarm is a good way to receive an early warning of factors that could cause us to pause your account's ability to send email. However, there are factors other than bounces and complaints that could cause us to pause your ability to send email. For more

information about creating alarms in CloudWatch, see Creating reputation monitoring alarms using CloudWatch.

You can also use the Account dashboard to determine the current status of your account. For example, if your account's ability to send email is currently paused, the **Account status** section of the Account dashboard displays a status of **Paused**. If your account is able to send email normally, it displays a status of **Healthy**.

Finally, you can check the AWS Health Dashboard (PHD) at https://phd.aws.amazon.com/ to determine if your account's ability to send email is currently paused. When we pause an account's ability to send email, we automatically add an **SES sending paused** event to the **Event log** section of the PHD. The **SES sending paused** event always has a Status of **Closed**, regardless of whether or not the account's ability to send email is currently paused. The event log also includes a copy of the email that we sent to the email address associated with your AWS account when the sending pause event occurred.

You can use CloudWatch to create an alarms that alert you when new events appear on your Personal Health Dashboard. For more information, see Monitoring AWS Health Events with CloudWatch Events in the *AWS Health User Guide*.

Q4. My account's ability to send email is paused. Does this impact my ability to use of other AWS services?

You can still use other AWS services while your account's ability to send email is paused. However, if you request a service quota increase for another AWS service that sends outbound communications (such as Amazon SNS), we might deny your request until your account's ability to send email is restored.

Q5. What should I do if my account's ability to send email is paused?

You should do the following:

- Look at the email you received from us for a summary of the issue.
- Investigate your sending to determine what aspect of your sending specifically triggered the issue.
- After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future.

- Be sure to provide any information we specifically request. We need this information to evaluate your case.

Q6. What's a review?

You can request that we review our decision to place your under review. See the following question for more information about requesting a review.

Q7. How do I request a review?

To request a review, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf.

In your request, provide the following information:

- Information about what caused the issue.
- A list of the changes that you've made to correct the issue. Only include the steps that you've already implemented, not the steps you plan to implement in the future.
- Information about how these changes will prevent the same issue from occurring again in the future.

Depending on the nature of the event that led us to pause your account's ability to send email, we might require additional information. See the FAQ topic associated with the issue you experienced for a list of the information you should include in your request.

Q8. What if my request isn't accepted?

We'll respond to your request with information about why we didn't accept it. In some cases, you'll be able to submit another request if you're able to demonstrate that you resolved the issue, and that your changes prevent the issue from occurring again in the future.

Q9. Can you help me diagnose the problem?

Typically we can give you only a high-level overview of your issue (for example, that you have a problem with bounces). It's your responsibility to correct the issue.

Q10. How do I know if my account's ability to send email has been restored?

Reputation metrics includes information about the current status of your account. For more information, see Using reputation metrics to track bounce and complaint rates.

Q11. Can I get help from my AWS representative or Premium Support?

If you're already working with an AWS account representative, we'll automatically contact him or her if we pause your account's ability to send email. Your account representative may be able to provide additional information to help you better understand the issue. If you use Premium Support, you should also contact that team for additional help.

Q12. How can I detect an account-level sending pause or reinstatement programmatically?

Account-level sending pauses and reinstatements are surfaced through the Account dashboard, the AWS Health Dashboard (PHD) at https://phd.aws.amazon.com/, and the AWS Support case that we open on your behalf. To react to these changes in your own automation, poll the account status rather than waiting for an event.

> **Note**
>
> SES does not emit an Amazon EventBridge event when it pauses or reinstates sending for your entire account. Automation that responds to account-level enforcement must read the account status; it cannot subscribe to an event for it. This is different from the tenant-level path, where pausing or reinstating an individual tenant does emit an EventBridge event. For more information, see the section called “Pause/Unpause” and Monitoring SES events using Amazon EventBridge.

To read the account status programmatically, call the `GetAccount` operation in the Amazon SES API v2 and check the `EnforcementStatus` field. This is the same value shown as **Account status** in the SES console. When your account's ability to send email is paused for enforcement, this status reflects the pause. When your account is reinstated, the status returns to **Healthy**.

> **Important**
>
> Do not use the value returned by `GetAccountSendingEnabled` (the `Enabled` flag) to detect an enforcement pause. That flag reflects whether you have turned account-level sending on or off yourself, and it does not change when SES pauses or reinstates your account for enforcement. Key your detection on the account status (`EnforcementStatus`) instead.

You can also detect account-level pauses by monitoring the PHD. When we pause your account's ability to send email, we add an **SES sending paused** event to the PHD, as described in Q3. My account's ability to send email is paused. Why didn't I receive a notification?. You can use Amazon CloudWatch to create an alarm that alerts you when new events appear on your PHD.

Bounce FAQ

Q1. Why do you care about my bounces?

High bounce rates are often used by entities such as email providers and anti-spam organizations to detect senders who engage in bad email-sending practices. High bounce rates can lead to email being sent to the spam folder rather than the inbox.

Q2. What should I do if I receive a notification stating that my account is under review or that my sending is paused because of my account's bounce rate?

Identify the cause of the issue, and then correct it. After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. Also include the following information:

- The method you use to track your bounces
- How you ensure that the email addresses of new recipients are valid prior to sending to them. For example, which of the recommendations are you following in Q11. What can I do to minimize bounces?

Q3. What types of bounces count toward my bounce rate?

Your bounce rate includes only hard bounces to domains you haven't verified. Hard bounces are permanent delivery failures such as "address does not exist." Temporary and intermittent failures such as "mailbox full," or bounces due to blocked IP addresses, don't count toward your bounce rate.

Q4. Do you disclose the bounce rates that could cause my account to be placed under review or that could cause my sending to be paused?

For best results, you should maintain a bounce rate below 2%. Higher bounce rates can impact the delivery of your emails.

If your bounce rate is 5% or greater, we'll place your account under review. If your bounce rate is 10% or greater, we might pause your account's ability to send additional email until you resolve the issue that resulted in the high bounce rate.

Q5. Over what period of time is my bounce rate calculated?

We don't calculate your bounce rate based on a fixed period of time, because different senders send at different rates. Instead, we look at a *representative volume*—an amount of email that represents your typical sending practices. To be fair to both high- and low-volume senders, the representative volume is different for each user and changes as the user's sending patterns change.

Q6. Can I calculate my own bounce rate by using the information from the SES console or the GetSendStatistics API?

No. The bounce rate is calculated using representative volume (see Q5. Over what period of time is my bounce rate calculated?). Depending on your sending rate, your bounce rate can stretch farther back in time than the SES console or `GetSendStatistics` can retrieve. In addition, only emails to non-verified domains are considered when calculating your bounce rate. However, if you regularly monitor your bounce rates using those methods, you should still have a good indicator that you can use to catch problems before they get to levels that cause us to place your account under review or pause your account's ability to send email.

Q7. How can I find out which email addresses bounced?

Examine the bounce notifications that SES sends you. The email address to which SES forwards the notifications depends on how you sent the original messages, as described at Receiving Amazon SES notifications through email. You can also set up bounce notifications through Amazon Simple Notification Service (Amazon SNS), as described at Setting up event notifications for Amazon SES. Note that simply removing bounced addresses from your list without any additional investigation might not solve the underlying problem. For information about what you can do to reduce bounces, see Q11. What can I do to minimize bounces?.

Q8. If I haven't been monitoring my bounces, can you give me a list of addresses that have bounced?

No, we can't provide a complete list of addresses that have bounced. You are responsible for monitoring and acting upon the bounces for your account.

Q9. How should I handle bounces?

You need to remove bounced addresses from your mailing list and stop sending mail to them immediately. If you're a small sender, it might be sufficient to simply monitor bounces through email and manually remove bounced addresses from your mailing list. If your volume is higher, you'll probably want to set up automation for this process, either by programmatically processing the mailbox where you receive bounces, or by setting up bounce notifications through Amazon SNS. For more information, see Setting up event notifications for Amazon SES.

Q10. Could my emails be bouncing because I've reached my sending quota?

No. Bounces aren't related to sending quotas. If you try to exceed your sending quota, you'll receive an error from the SES API or SMTP interface when you try to send an email.

Q11. What can I do to minimize bounces?

First, be sure that you're aware of your bounces (see Q7. How can I find out which email addresses bounced?). Then follow these guidelines:

- Don't buy, rent, or share email addresses. Send email only to recipients who explicitly requested to receive email from you.
- Remove bounced email addresses from your list.
- On web forms, ask users to enter their email addresses two times, and check to make sure both addresses match before the form can be submitted.
- Use double opt-in to sign up new users. That is, when a new users sign up, send them a confirmation email that they need to click before receiving any additional mail. This prevents people from signing up other people as well as accidental sign-ups.
- If you must send to addresses that you haven't mailed lately (and thus you can't be confident that the addresses are still valid), do so only with a small portion of your overall sending. For more information, see our blog post Never send to old addresses, but what if you have to?.
- Ensure that you're not structuring sign-ups to encourage people to use fictional addresses. For example, don't provide any added value or benefits until recipients verify their addresses.
- If you have an "email a friend" feature, use CAPTCHA or a similar mechanism to discourage automated use of the feature, and don't allow users to insert arbitrary content.
- If you're using SES for system notifications, ensure that you're sending the notifications to real addresses that can receive mail. Also consider turning off notifications that you don't need.

- If you're testing a new system, be sure you're either sending to real addresses that can receive email, or you're using the SES mailbox simulator. For more information, see Using the mailbox simulator manually.

Complaint FAQ

Q1. What's a complaint?

A complaint occurs when a recipient reports that they don't want to receive an email. They might have clicked the "Report spam" button in their email client, complained to their email provider, notified SES directly, or through some other method. This topic includes general information about complaints. If your notification contains specific information about the source of the complaints, also read the relevant topic:

- SES complaints through feedback loops FAQ
- SES complaints directly from recipients FAQ
- SES complaints through email providers FAQ

Q2. Why do you care about my complaints?

High complaint rates are often used by entities such as email providers and anti-spam organizations as indicators that a sender is sending to recipients who didn't specifically sign up to receive emails, or that the sender is sending content that is different from the type that recipients signed up for.

Q3. What should I do if I receive a notice saying that my account is under review or that my sending is paused because of an issue with complaints?

Review your list acquisition process and the content of your emails to try to understand why your recipients might not appreciate the email they're receiving from you. Identify the cause of the issue, and then correct it. After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future.

Q4. What can I do to minimize complaints?

First, be sure that you monitor the complaints that SES can notify you about, which are complaints that SES receives through feedback loops (see the SES complaints through feedback loops FAQ). Then follow these guidelines:

- Do not buy, rent, or share email addresses. Use only addresses that specifically requested your mail.
- Use double opt-in to sign up new users. That is, when users sign up, send them a confirmation email that they need to click before receiving any additional mail. This prevents people from signing up other people as well as accidental sign-ups.
- Monitor engagement with the mail you send and stop sending to recipients who don't open or click your messages.
- When new users sign up, be clear about the type of email they will receive from you, and ensure that you send only the type of mail that they signed up for. For example, if users sign up for news updates, don't send them advertisements.
- Ensure that your mail is well-formatted and looks professional.
- Ensure that your mail is clearly from you and can't be confused for something else.
- Provide users an obvious and easy way to unsubscribe from your mail.

SES complaints through feedback loops FAQ

This topic provides information about complaints that SES receives from email providers through feedback loops. For general information that applies to all types of complaints, see the Complaint FAQ.

Q1. How is this type of complaint reported?

Most email client programs provide a button labeled "Mark as Spam" or similar, which moves the message to a spam folder and forwards it to the email provider. Additionally, most email providers maintain an abuse address (such as *abuse@example.com*), where users can forward unwanted email and request that the provider take action to prevent them. If SES has a feedback loop (FBL) set up with the email provider, then they send the complaint back to SES.

Note

SES automatically sets the Feedback-ID header when you send messages, giving mailbox providers a way to aggregate delivery statistics, such as complaint and spam rates, and make them available to you. The Feedback-ID header value that SES provides is comprised as such:

- `FeedBackId:((SESInternalID):(AmazonSES))`, where:
  - *SESInternalID* is the identifier used by SES for collecting complaint information.
  - *AmazonSES* is a static tag identifying SES as the sending platform.

Optionally, in addition to the standard Feedback-ID header value that SES provides, you can also specify your own customized feedback IDs (up to two) using the `ses:feedback-id-a` and `ses:feedback-id-b` message tags, see the section called "Fine-grained feedback for email campaigns".

Q2. Are these complaints included in the complaint rate statistic shown in the SES console and returned by the GetSendStatistics API?

Yes. However, the complaint rate statistic doesn't include complaints from email providers that don't provide feedback to SES. The complaint rate from domains that provide feedback is likely to be representative of the rest of your sending as well.

Q3. How can I be notified of these complaints?

You can be notified through email or through Amazon SNS notifications. See the set-up instructions in Setting up event notifications for Amazon SES.

Q4. What should I do if I receive a complaint notification through email or through Amazon SNS?

First, you need to remove addresses that generated complaints from your mailing list and stop sending mail to them immediately. Do not even send an email that says you've received the request to unsubscribe. Consider setting up automation for this process, either by programmatically processing the mailbox where you receive complaints, or by setting up complaint notifications through Amazon SNS. For more information, see Setting up event notifications for Amazon SES.

Then, take a close look at your sending to determine why your recipients don't appreciate the mail you're sending, and address that underlying problem. For every person who complains, there are potentially dozens who didn't appreciate your mail who didn't (or weren't able to) complain. If you only remove the recipients who actually complain, you're not addressing the underlying problem.

Q5. Do you disclose the SES complaint rates that could cause my account to be placed under review or that could result in my account's ability to send email being paused?

For best results, you should maintain a complaint rate below 0.1%. Higher complaint rates can impact the delivery of your emails.

If your complaint rate is 0.1% or greater, we'll place your account under review. If your complaint rate is 0.5% or greater, we might pause your account's ability to send additional email until you resolve the issue that resulted in the high complaint rate.

Q6. Over what period of time is my complaint rate calculated?

We don't calculate your complaint rate based on a fixed period of time, because different senders send at different rates. Instead, we look at a *representative volume*—an amount of mail that represents your typical sending practices. To be fair to both high- and low-volume senders, the representative volume is different for each user and changes as the user's sending patterns change. Additionally, the complaint rate isn't calculated based on every email. Instead, it's calculated as the percentage of complaints on mail sent to domains that send complaint feedback to SES.

Q7. Can I calculate my own complaint rate by using metrics from the SES console or the GetSendStatistics API?

No. There are two primary reasons for this:

- The complaint rate is calculated using representative volume (see Q6. Over what period of time is my complaint rate calculated?). Depending on your sending rate, your complaint rate can stretch farther back in time than the SES console or `GetSendStatistics` API can retrieve. For this reason, we recommend that you regularly use these methods to monitor the complaint rate for your account. Monitoring your complaint rate in this way gives you the information you need to identify problems before they reach levels that could impact the delivery of your email.
- When calculating complaint rate, not every email counts. Complaint rate is calculated as the percentage of complaints on mail sent to domains that send complaint feedback to SES.

Q8. How can I find out which email addresses complained?

Examine the complaint notifications that SES sends you through email or through Amazon SNS (see Setting up event notifications for Amazon SES). However, different email providers provide differing amounts of information, and some providers redact the recipient's email address before passing the complaint notification to SES. To enable you to find the recipient's email address in the future, your best option is to store your own mapping between an identifier and the SES message ID that SES passes back to you when it accepts the email. Note that SES doesn't retain any custom message IDs that you add.

Q9. If I haven't been monitoring my complaints, can you give me a list of addresses that have complained?

Unfortunately, we can't give you a comprehensive list. However, you can monitor future complaints by email or through Amazon SNS.

Q10. Can I get a sample email?

We can't send you a sample email upon request, but you might find this information in the complaint notification. For more information, see Q8. How can I find out which email addresses complained?.

SES complaints directly from recipients FAQ

This topic provides information about complaints that SES receives directly from recipients. For general information that applies to all types of complaints, see the Complaint FAQ.

Q1. How is this type of complaint reported?

Multiple recipients directly contacted SES about your mail through email or some other means.

Q2. Are these complaints included in the complaint rate statistic shown in the SES console and returned by the GetSendStatistics API?

No. The complaint rate statistic you retrieve using the SES console or the `GetSendStatistics` API only includes complaints that SES receives through feedback loops. For more information about those types of complaints, see the SES complaints through feedback loops FAQ.

Q3. Why haven't I heard about these complaints through email feedback notifications or through Amazon SNS?

Email feedback forwarding and Amazon SNS notifications only include complaints that SES receives through feedback loops. You won't receive notifications for complaints that recipients filed directly with SES.

Q4. How can I find out which email addresses complained?

To protect the identities of the recipients who complained, we can't list the email addresses that complained about your email.

Rather than focus on removing individual recipients from your lists, we recommend that you determine the problem that led to the complaints being issued. We recommend that you begin by reviewing your customer acquisition process, and that you remove any customers from your lists that didn't explicitly ask to receive email from you. You should also analyze the content of your emails to try to understand why your recipients are complaining.

Q5. Can I get a sample email?

To protect the identities of the recipients who complained, we can't provide copies of the emails that caused your recipients to complain.

Q6. What should I do if I receive a notification stating that my account is under review or that my sending is paused because of direct complaints?

Immediately change your sending processes so that you're only sending messages recipients who have specifically signed up to receive them. Also, ensure that you're sending the type of content that your recipients signed up to receive. After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future.

If you don't request a review within three weeks, and we continue to receive direct recipient complaints, we might pause your account's ability to send email.

SES complaints through email providers FAQ

This topic provides information about complaints that SES receives through email providers (also called *mailbox providers*). For general information that applies to all types of complaints, see the Complaint FAQ.

Q1. How is this type of complaint reported?

An email provider reported to SES that a significant number of its customers marked your emails as spam. The report was provided to SES through a means other than the feedback loops described in the SES complaints through feedback loops FAQ.

Q2. Are these complaints included in the complaint rate statistic shown in the SES console and returned by the GetSendStatistics API?

No. The complaint rate statistic you retrieve using the SES console or the `GetSendStatistics` API only includes complaints that SES receives through feedback loops.

Q3. Why haven't I heard about these complaints through email feedback notifications or through Amazon SNS?

Email feedback forwarding and Amazon SNS notifications only include complaints that SES receives through feedback loops.

Q4. How can I find out which email addresses complained?

Email providers typically don't disclose this information. However, rather than focusing on removing individual recipients from your list, you need to focus on finding and fixing the underlying problem. Start by reviewing your list acquisition process and the content of your emails to try to understand why your recipients might not appreciate your email.

Q5. Can I get a sample email?

No. Email providers typically don't provide an example email.

Q6. What should I do if I receive a notification stating that my account is under review or that my sending is paused because of email provider complaints?

Identify the cause of the issue, and then correct it. After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. If you don't request a review within three weeks, and we continue to receive complaints from providers, we might pause your account's ability to send additional email.

Spamtrap FAQ

Q1. What are spamtraps?

A spamtrap is a special email address maintained by an Internet Service Provider (ISP), email provider, or anti-spam organization. Because that address will never legitimately be signed up to receive email, the organizations that maintain these spamtraps know that anyone who sends mail to any of these addresses is likely to be engaging in questionable email practices.

Q2. How are spamtraps set up?

Spamtrap addresses can be set up in multiple ways. They can be converted from addresses that were once valid, but have been unused (and bouncing) for an extended period of time. They can also be addresses that were set up just to be spamtraps. They can be unusual addresses that are hard to guess, and sometimes they are addresses that are close to real addresses (for example, introducing a typo into a common domain name). Often, but not always, spamtraps are "seeded" into the world by putting them on the internet in a variety of ways.

Q3. How does SES know if I am sending to spamtraps?

Certain organizations that operate spamtraps send SES notifications when their spamtraps are hit by SES senders.

Q4. How does SES use the spamtrap reports?

We review the reports. If we determine that your account is sending email to spamtraps, we place your account under review and ask you to fix the underlying problem. If you don't fix the problem before the review period is over, we might pause your account's ability to send additional email. If your spamtrap problem is very severe, we might pause your account's ability to send email immediately, without placing your account under review first.

Q5. What should I do if a receive a notice saying that my account is under review or that my sending is paused because of an issue with spamtraps?

First, you should address the issue that caused us to place your account under review or pause your ability to send email. Next, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in

the future. If we agree that the changes you've made appropriately address the issue, we'll cancel the review period or remove the sending pause from your account.

Because of the way that spamtrap hits are reported, it may take three weeks or more before we are able to determine if the changes you made solved the issue.

Q6. How many spamtrap hits can I have before you place my account under review or pause my account's ability to send email?

We don't disclose the specific number of spamtrap hits that cause us to take action on your account. However, it's important to note that even a small number of spamtrap hits can have a very negative effect on your reputation as a sender, so you should take spamtrap reports seriously.

Q7. Do you disclose the spamtrap addresses?

No. In order for spamtraps to be effective, it's essential that they remain confidential. Spamtrap organizations disclose only the occurrence of spamtrap hits, not the actual spamtrap addresses.

Q8. What can I do to avoid sending to spamtraps?

To reduce the risk of sending to spamtraps, follow these guidelines:

- Do not buy, rent, or share email addresses. Use only addresses that specifically requested your mail.
- On web forms, ask users to enter their email addresses two times, and check to make sure both addresses match before the form can be submitted.
- Use double opt-in to sign up new users. That is, when users sign up, send them a confirmation email that they need to click before receiving any additional mail.
- Ensure that you remove addresses that hard bounce from your list, so that they are removed long before they are converted to spamtraps.
- Ensure that you're monitoring engagement by your recipients, and stop sending to recipients who haven't engaged with your emails or website recently. Time frames for what an "engaged user" is depend on your use case, but generally speaking if users haven't opened or clicked your emails in several months, you should consider removing them unless you have evidence that they do want your mail.
- Be very careful with re-engagement campaigns where you intentionally contact people who haven't interacted with you recently. These efforts tend to be highly risky, and can often cause problems not only with spamtrap sending, but also with bounces and complaints.

- Send an opt-in message to your entire mailing list and keep only the recipients who click on the verification link. In addition to removing inactive recipients from your list, this procedure also helps remove spamtrap addresses. However, we don't recommend using this technique if you think that your mailing list might contain a lot of bad addresses, or if your account already has a problem with bounces, because it might cause your account's bounce rate to increase further.

Manual investigation FAQ

Q1. What should I do if I receive a notification stating that my account is under review or that my sending is paused because of a manual investigation?

An SES investigator has identified a significant problem with your sending. Typical problems include, but aren't limited to, the following:

- Your sending violates the AWS Acceptable Use Policy (AUP).
- Your emails appear to be unsolicited.
- Your content is phishing related (this includes simulated phishing).
- Your content is otherwise associated with a use case that SES doesn't support.

If we believe that the problem can be corrected, we place your account under review for a certain amount of time. While your account is under review, you should make changes to your email sending practices to correct the issue.

If we don't believe that the problem can be corrected, or if the problem is very severe, we might pause your account's ability to send email without first placing your account under review.

Q2. What issues could cause you to perform a manual review of my email sending?

There are several issues that could cause us to begin a manual review of your account. These reasons include, but aren't limited to, the following:

- Recipients contact SES to complain about email sent from your account.
- We detect unusual changes in your email sending patterns.
- Our spam filters find characteristics of your email that are typical of unsolicited or low-quality content.

When we place your account under review or pause your account's ability to send email, we send you a notification. In most cases, this notification contains information about the issue, and provides information about the next steps you can take.

Q3. What are "unsolicited" emails?

Unsolicited emails are emails that the recipient didn't explicitly ask to receive. This includes cases in which a recipient signs up for a certain type of mail (for example, notifications), and instead is sent a different type of mail (for example, advertisements).

When we place your account under review or pause your account's ability to send email, we send you a notification. If you receive a notification stating that we're taking one of these actions because of an issue with unsolicited email, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, include the following information:

- Are all the messages that you send specifically requested by the recipient, and do they comply with the AWS Acceptable Use Policy?
- Have you acquired email addresses in any way other than a customer specifically interacting with you or your website and requesting emails from it? You should explain how you acquired your mailing list.
- How do your subscribe and unsubscribe processes work? You should include your opt-in and opt-out links.

Q4. What should I do if I receive a notification stating that my account is under review or that my sending is paused because of a manual review?

Identify the cause of the issue, and then correct it. After you make changes that you believe will resolve the issue, sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf. In your message, provide detailed information about the steps you've taken to resolve the issue, and describe how these steps prevent the issue from happening again in the future. If we agree that the changes you've made appropriately address the issue, we'll cancel the review period on your account.

Q5. What types of problems do you view as "correctable?"

Generally, we believe the situation is correctable if you have a history of good sending practices, and if there are steps you can take to eliminate the problematic sending while continuing the bulk of your sending. For example, if you're sending three different types of email and only one type is

problematic, you might be able to simply stop the problematic sending and continue with the rest of your sending.

Q6. What if I can't find the source of the problem?

You can sign into the AWS Console and go to Support Center. Reply to the case we opened on your behalf and request a sample of the mail that caused the issue.

DNS Blackhole List (DNSBL) FAQs

Domain Name System-based Blackhole Lists (*DNSBLs*)—sometimes referred to as *Realtime Blackhole Lists* (*RBLs*), *deny lists*, *blocklists*, or *blacklists*—are intended to inform email providers of IP addresses that are suspected of sending unwanted email.

Different DNSBLs have different impacts on email deliverability. This topic describes how DNSBLs impact the delivery of emails you send using Amazon SES, as well as our policies for removing Amazon SES IP addresses from DNSBLs.

> **Note**
>
> This topic is about the DNSBLs that email providers use to block incoming messages. For information about how Amazon SES blocks outgoing email sent to recipients whose email addresses have previously generated bounces, see Amazon SES global suppression list.

Q1. How do DNSBLs impact email delivery?

Different DNSBLs have different impacts on the successful delivery of a message. Major email providers—including Gmail, Hotmail, AOL, and Yahoo—seem to recognize a very small number of highly regarded DNSBLs, such as those offered by Spamhaus. In our experience, other DNSBLs tend to have a low impact, although some mail systems emphasize certain DNSBLs over others.

Finally, many email providers have their own internal deny lists. Email providers guard these lists very closely, and rarely share them with the public. If an IP address is on one of these lists, it can have a major impact on your ability to send email to recipients who use that provider.

Q2. How do IP addresses end up on DNSBLs?

There are several ways that an IP address can end up on a DNSBL. IP addresses can be added to DNSBLs when they send email to a *spamtrap*. A spamtrap is an email address that doesn't belong

to a human user. Spamtraps exist solely to collect spam and identify spammers. Some DNSBLs also allow individual users to submit IP addresses. A few DNSBLs even allow users to submit entire IP address ranges. Other DNSBLs are maintained through contributions by email administrators, and can include IP addresses that administrators believe are abusing their own systems.

Q3. How does Amazon SES prevent its IP addresses from appearing on DNSBLs?

Our systems look for signs of abuse. If we detect sending patterns or other characteristics that could lead to an IP address being added to a DNSBL, we send a notification to the sender. If the situation is severe, or if the sender doesn't fix the issue after we send the notification, we'll pause the sender's ability to send email until they resolve the issue. Enforcing our sending policies in this way helps reduce the chances that our IP addresses end up on DNSBLs.

Q4. Can Amazon SES have its IP addresses removed from a DNSBL?

For SES shared IPs, we actively monitor DNSBLs that could impact delivery across the entire Amazon SES service, or that could impact the ability to send email to recipients who use major email providers, such as Gmail, Yahoo, AOL, and Hotmail. The DNSBLs offered by Spamhaus fall into this category. When one of our IP addresses appears on a list that meets either of these criteria, we take immediate action to have that address removed from the DNSBL as quickly as possible.

We don't monitor DNSBLs that are unlikely to impact delivery across the entire Amazon SES service, or that don't have a measurable impact on delivery to major email providers. The DNSBLs offered by SORBS and UCEPROTECT fall into this category. Because of the specific listing and delisting practices of the vendors who operate these lists, we are unable to have our IP addresses removed from these lists.

For dedicated IP address (managed or standard) listed in any RBL of recipient mail providers, it is not the responsibility of SES to delist the IPs and you must request removal yourself directly to the RBL administrator.

Q5. An email provider is rejecting my email because the sending IP address is listed by a DNSBL other than Spamhaus. What can I do?

First, confirm that the message was truly blocked because of a DNSBL. If your email was rejected because the sending IP address was added to a DNSBL, you'll receive a bounce notification that mentions the DNSBL provider by name, as in the following example:

```
554 5.7.1 Service unavailable; Client host [192.0.2.0] blocked using DNSBLName;
 See: http://www.example.com/query/ip/192.0.2.0
```

If you received a bounce notification, but it didn't contain information similar to the message shown in the preceding example, then the email provider most likely rejected your message for a reason unrelated to being added to a DNSBL.

If you can confirm that an email provider is blocking your email because the sending IP address is on a DNSBL, there are a few things you can do:

- Contact the postmaster of the domain that rejected your message to request an exception from their spam filtering policy. Some postmasters have support processes, and may publish a postmaster page that describes this process. If the domain you're trying to contact doesn't publish its postmaster support policies, you might be able to contact the postmaster by sending email to *postmaster@example.com*, where *example.com* is the domain in question. Domains are required by RFC 5321 to have a postmaster mailbox.

  When you contact the postmaster, provide the bounce codes you received, the headers of the email you're trying to send, a measurement of the impact the DNSBL is having on the delivery of your email, and information about why you believe that your email is being improperly blocked. The more information you can provide to the postmaster to demonstrate that you're sending legitimate email, the more likely the postmaster is to make an exception for you.
- If the email provider doesn't respond, or is unwilling to change their policies, consider using a dedicated IP address. Dedicated IP addresses are addresses that only you can use. By implementing good sending practices, you can keep your engagement rates high, and your rates of bounces, complaints, and spamtrap hits low. Good sending practices can help ensure that your addresses don't end up on DNSBLs.

Q6. Email that I send to Gmail, Yahoo, Hotmail, or another major provider is being sent to the spam folder. Is this happening because my sending IP address is on a DNSBL?

Probably not. If an IP address is listed by a DNSBL with significant impact, such as one of the DNSBLs from Spamhaus, major email providers will reject email from that IP address completely, rather than sending it to the spam folder.

When major email providers accept an email (rather than rejecting it), they usually consider *user engagement* when considering whether to place the message in the inbox or in the spam folder. *User engagement* refers to the ways in which users interacted with the messages you sent them previously.

To increase the chances that your messages reach your customers' inboxes, you should implement all of the following best practices:

- **Never** rent or purchase lists of email addresses. Renting or purchasing lists is a violation of the AWS Acceptable Use Policy (AUP) and isn't allowed on Amazon SES under any circumstances.
- Only send email to customers who explicitly asked to receive email from you. In many countries and jurisdictions around the world, it's illegal to send email to recipients who didn't explicitly agree to receive email from you.
- Stop sending email to customers who haven't opened or clicked links in messages that you've sent in the past 30–90 days. This step can help to keep your engagement rates high, which increases the chances that the messages you send in the future arrive in recipients' inboxes.
- Use consistent design elements and writing styles in each message that you send to ensure that customers can easily identify messages from you.
- Use email authentication mechanisms, such as SPF and DKIM.
- When customers use a web form to subscribe to your content, send them an email to confirm that they want to receive email from you. Don't send them any additional email until they confirm that they want to receive email from you. This process is known as *confirmed opt-in* or *double opt-in*.
- Make it easy for your customers to unsubscribe, and honor unsubscribe requests immediately.
- If you send email that contains links, check those links against the Spamhaus Domain Block List (DBL). To test your links, use the Domain Lookup Tool on the Spamhaus website.

By implementing these practices, you can improve your sender reputation, which increases the likelihood that the email you send reaches recipients' inboxes. Implementing these practices also helps keep the bounce and complaint rates low for your account, and reduces the risk of sending email to spamtraps.

Amazon SES email sending metrics FAQs

Amazon SES collects several metrics about the emails you send. These metrics enable you to analyze the effectiveness of your email program and monitor important statistics, such as your bounce and complaint rates.

This section contains FAQs on the following topics related to email sending metrics:

- General Questions
- Open Tracking
- Click Tracking

> **Note**
>
> Event tracking is dependent upon the recipient's email service provider (ESP) and how they've configured their privacy settings which are beyond the control of Amazon SES. The count of tracking events can be skewed (returning inaccurate counts) under conditions such as:
>
> - The email recipient is using an email service provider (ESP) that protects their privacy.
> - The email recipient explicitly doesn't give their ESP permission to share their data.
> - The email recipient's ESP caches images or links, SES can only count the initial open, but won't be able to count subsequent openings.

General Questions

Q1. After an email is delivered, how long does Amazon SES continue to collect open and click metrics?

Amazon SES collects open and click metrics for 60 days after each email is sent.

Q2. If a user opens an email multiple times, or clicks a link in an email multiple times, is each of those events tracked separately?

If a recipient opens an email multiple times, Amazon SES counts each open as a unique open event. Similarly, if a recipient clicks the same link multiple times, Amazon SES counts each click as a

unique click event. However, these counts can be skewed by the scenarios outlined above in the note box.

Q3. Are open and click metrics aggregated, or can they be measured down to the recipient level?

Opens and clicks are tracked at the recipient level. With open and click tracking, you can determine which recipients opened an email or clicked a link in an email.

Q4. Can I retrieve open and click metrics using the Amazon SES API?

The Amazon SES API does not provide a method for retrieving open and click metrics. However, you can retrieve open and click metrics for Amazon SES using the CloudWatch API. For example, you can use the AWS CLI to retrieve click metrics using the CloudWatch API by issuing the following command:

```
aws cloudwatch get-metric-statistics --namespace AWS/SES --metric-name Click \
  --statistics Sum --period 86400 --start-time 2017-01-01T00:00:00Z \
  --end-time 2017-12-31T23:59:59Z
```

The command shown above retrieves the total number of click events for each day in 2017. To retrieve open metrics change the value of the `metric-name` parameter to `Open`. You can also modify the `start-time` and `end-time` parameters to change the analysis period, or change the `period` parameter for more fine-grained analysis.

Open Tracking

Q1. How does open tracking work?

A 1 pixel by 1 pixel transparent GIF image is inserted in each email sent through Amazon SES and includes a unique reference to this image file; when the image is downloaded, SES can tell exactly which message was opened and by whom.

By default, this pixel is inserted at the bottom of the email; however, some email providers' applications truncate the preview of an email when it exceeds a certain size and may provide a link to view the remainder of the message. In this scenario, the SES pixel tracking image does not load and will throw off the open rates you're trying to track. To get around this, you can optionally place the pixel at the beginning of the email, or anywhere else, by inserting the `{{ses:openTracker}}`

placeholder into the email body. Once SES receives the message with the placeholder, it will be replaced with open tracking pixel image.

> **Important**
>
> Just add one `{{ses:openTracker}}` placeholder, as more than one will result in a `400 BadRequestException` error code being returned.

The addition of this tracking pixel does not change the appearance of your email.

Q2. Is open tracking enabled by default?

Open tracking is available to all Amazon SES users by default. To use open tracking, you must do the following:

1. Create a configuration set.
2. In the configuration set, create an event destination.
3. Configure the event destination to publish open event notifications to a destination.
4. In every email for which you want to track opens, specify the configuration set that you created in step 1.

For details about how to enable open tracking through a configuration set's event destination, see the section called “Create event destinations”. You can use the pixel placeholder in SMTP email in such ways as formatted, raw, and templated email.

Learn more about how to Monitor email sending using event publishing.

Q3. Can I omit the open tracking pixel from certain emails?

There are two ways to omit the open tracking pixel from your emails. The first method is to send the email without specifying a configuration set. Alternatively, you can specify a configuration set that is not configured to publish data about open events.

Q4. Do you track opens for plaintext emails?

Open tracking only works with HTML emails. Because open tracking relies on the inclusion of an image, it is not possible to collect open metrics for users who open emails using a text-only (non-HTML) email client.

Click Tracking

Q1. How does click tracking work?

To track clicks, Amazon SES modifies each link in the body of the email. When recipients open a link, they are sent to an Amazon SES server, and are immediately forwarded to the destination address. As with open tracking, each redirect link is unique. This enables Amazon SES to determine which recipient clicked the link, when they clicked it, and the email from which they arrived at the link.

> **Important**
>
> If you send a single message to multiple recipients, each recipient will save the same click tracking link. To track individual recipients' click activity, send email to one recipient per send operation.

Q2. Can I disable click tracking?

You can disable click tracking for individual links by adding an attribute, `ses:no-track`, to the anchor tags in the HTML body of your email. For example, if you link to the AWS home page, a normal anchor link resembles the following:

```
<a href="https://aws.amazon.com">Amazon Web Services</a>
```

To disable click tracking for that link, modify it to resemble the following:

```
<a ses:no-track href="aws.amazon.com">Amazon Web Services</a>
```

Because `ses:no-track` isn't a standard HTML attribute, Amazon SES automatically removes it from the version of the email that arrives in your recipients' inboxes.

You can also disable click tracking for all messages that you send using a specific configuration set. To disable click tracking, modify the configuration set event destination so that it doesn't capture click events.

For details about how to enable and disable click tracking through a configuration set's event destination, see the section called “Create event destinations”.

Learn more about how to Monitor email sending using event publishing.

Q3. How many links can be tracked in each email?

The click tracking system can track a maximum of 250 links.

Q4. Are click metrics collected for links in plain text emails?

It's only possible to track clicks in HTML emails.

Q5. Can I tag links with unique identifiers?

You can add an unlimited number of tags, as key-value pairs, to links in your email by using the `ses:tags` attribute. When you use this attribute, specify the keys and values using the same format that you would use to pass inline CSS properties: type the key, followed by a colon (:), followed by the value. If you need to pass several key-value pairs, separate each pair with a semicolon (;).

For example, assume you want to add the tags `product:book, genre:fiction, subgenre:scifi, type:newrelease` to a link. The resulting link resembles the following:

```
<a ses:tags="product:book;genre:fiction;subgenre:scifi;type:newrelease;"
    href="http://www.amazon.com/…/">New Releases in Science Fiction</a>
```

These tags are passed through to your event publishing destination so that you can perform additional analysis on the specific links that your users clicked.

> **Note**
>
> Link tags can include the numbers 0–9, the letters A–Z (both uppercase and lowercase), hyphens (-), and underscores (_).

Q6. Do tracked links use the HTTP or HTTPS protocol?

Tracking links use the same protocol as the original links in your email.

For example, if your email includes a link to `https://www.amazon.com`, the link is replaced with a tracking link that uses the HTTPS protocol. If your email includes a link to `http://www.example.com`, the link is replaced with a tracking link that uses HTTP. If your email includes

both of the previously mentioned links, the HTTPS link is replaced with a tracking link that uses the HTTPS protocol, and the HTTP link is replaced with a tracking link that uses the HTTP protocol.

Q7. A link in my email isn't being tracked. Why not?

Amazon SES expects the links in your emails to contain properly encoded URLs. Specifically, URLs in your links must comply with RFC 3986. If a link in an email isn't properly encoded, recipients will still see the link in the email, but Amazon SES won't track click events for that link.

Issues related to improper encoding typically occur in URLs that contain query strings. For example, if the URL of a link in your email contains a non-encoded space character in the query string (such as the space between "John" and "Doe" in the following example: *http://www.example.com/path/to/page?name=John Doe*), Amazon SES won't track that link. However, if the URL uses an encoded space character instead (such as "%20" in the following example: *http://www.example.com/path/to/page?name=John%20Doe*), Amazon SES tracks it as expected.

Q8. Can I customize the URL path in a click-tracking link?

Yes. Use the `ses:custom-path` attribute on an anchor tag to add a fixed path segment to the click-tracking URL:

```
<a href="https://example.com/product" ses:custom-path="myapp">Shop now</a>
```

The value must contain only letters (A–Z, a–z), digits (0–9), hyphens (-), periods (.), or underscores (_). It must be 1–32 characters long and is case-sensitive. SES removes this attribute from the email before delivery. If the value is invalid, the link falls back to the default format.

If `ses:no-track` is also present on the same link, click tracking is disabled and `ses:custom-path` has no effect.

> **Note**
>
> For instructions on using this attribute for iOS Universal Links or Android App Links, see Custom path segments for Universal Links and App Links.

Quick Find Index

The following index has been created to help you quickly find things in Amazon SES by providing two ways of searching - either by how-tos or concepts. The how-tos describe "how to" do something while concepts explain the bigger picture.

Let us know what you think

Please use the **Feedback** button in the upper-right corner to let us know...

- Was this index helpful?
- Are there any how-tos or concepts you'd like to see added to this index?
- Was there something you thought should have been categorized differently?

SES How-to & concept links

How-tos

SES how-to links are listed alphabetically and will take you to the corresponding section to show you "how to" perform the action you selected.

- Learn how to...
  - Add an SPF Record as part of setting up a custom MAIL FROM domain
  - Assign IP pools
  - Block SPAM for email receiving
  - Configure custom open/click domains
  - Configure SNS notifications
  - Connect to a SMTP endpoint
  - Create a configuration set
  - Create a domain identity
  - Create an email address identity
  - Create event destinations
  - Create IP address filters
  - Create a managed IP pool to enable dedicated IPs (managed)

- Create receipt rules
- Create reputation alarms using CloudWatch
- Create a sending authorization policy using a custom policy
- Create a sending authorization policy using the policy generator
- Create standard dedicated IP pools for dedicated IP addresses (standard)
- Delete an identity
- Delete personal data
- Edit an identity
- Enable email feedback forwarding
- Export reputation metrics
- Get out of the sandbox
- Get started with SES
- Get started with Virtual Deliverability Manager
- Give permissions for email receiving
- Increase throughput
- Increase your sending quota
- Integrate with your existing email server
- Log API calls
- Manage a configuration set
- Manage Easy DKIM & BYODKIM
- Monitor send and reputation metrics
- Monitor sending statistics
- Monitor usage statistics
- Monitor your sending quota
- Obtain DKIM records for an identity
- Obtain SMTP credentials
- Override account-level suppression with configuration set-level suppression
- Override inherited DKIM signing on an email address identity
- Pause email sending
- Publish an MX record

- Report abuse of AWS resources
- Request dedicated IP addresses
- Request technical support
- Resolve deliverability & reputation issues using the Virtual Deliverability Manager advisor
- Retrieve event data from CloudWatch
- Retrieve event data from Kinesis Data Firehose
- Retrieve event data from SNS
- Send email using an AWS SDK
- Send emails programmatically
- Send email using the SES API
- Send email using SMTP
- Send raw email with an attachment using the CLI or SES API
- Send test emails using the mailbox simulator
- Set up BYODKIM *(Bring Your Own DKIM)*
- Set up a DMARC policy
- Set up Easy DKIM
- Set up email receiving
- Set up event publishing
- Set up a MAIL FROM domain
- Set up sending authorization (identity owner tasks)
- Set up sending authorization (delegate sender tasks)
- Specify a configuration set when sending email
- Test your connection to the SMTP interface
- Track bounce and complaint rates
- Understand inherited DKIM signing properties
- Use reputation metrics
- Use software packages to send email
- Use subscription management
- Use templates to send email
- Use your account-level suppression list

- Verify a domain identity
- Verify an email address identity
- View an identity
- View high & detailed levels of your account's deliverability metrics using the Virtual Deliverability Manager dashboard
- View SNDS metrics for dedicated IPs
- Warm up dedicated IP addresses

Concepts

SES concept links are listed alphabetically and will take you to the corresponding chapter and sections to explain the concept you selected.

- Find information about...
  - Abuse of AWS resources, report
  - Account dashboard
  - Account-level suppression list
  - Action options for email receiving
  - Add header action
  - Attachment types, unsupported
  - Bounce response action, return
  - BYODKIM *(Bring Your Own DKIM)*
  - BYOIP *(Bring Your Own IP)*
  - Code examples
  - Compliance validation
  - Configuration set-level suppression
  - Configuration sets
  - Content encodings
  - Cross-account notifications legacy support
  - Custom MAIL FROM domain
  - Data protection
  - Dedicated IP addresses

- Dedicated IP addresses (managed)
- Dedicated IP addresses (standard)
- DKIM, authenticating email with
- DMARC *(Domain-based Message Authentication, Reporting and Conformance)*
- DMARC through DKIM, complying with
- DMARC through SPF, complying with
- Easy DKIM
- Email feedback forwarding destination
- Email receiving authentication
- Email receiving concepts
- Email receiving console walkthroughs
- Email receiving malware scanning
- Email receiving permissions
- Email receiving use cases
- Email receiving restrictions
- Email sending authentication methods
- Endpoints
- Event notifications
- Event notifications through email
- Event notifications through SNS
- Event publishing
- FAQs *(Frequently Asked Questions)*
- Global suppression list
- Header fields, supported
- Identities, managing
- Identity and access management
- Infrastructure security
- Integrate with Amazon WorkMail action
- IP-based control using IP address filters
- Lambda function action, invoke

- List management
- Lists and subscriptions
- Logging and monitoring
- Malware detection
- Manual DKIM signing
- Monitor email sending using event publishing
- Monitor sender reputation
- Monitoring sending activity
- Quotas
- Receipt rules
- Recipient-based control using receipt rules
- Regions
- Reputation metrics
- Reputation metrics messages
- Resilience
- S3 bucket action, deliver to
- Sandbox - getting out of
- Security
- Security protocols, supported
- Sending authorization
- Sending authorization policy anatomy
- Sending authorization policy examples
- Sending authorization process
- SNDS metrics for dedicated IPs
- SNS notification contents
- SNS notification examples
- SNS topic action, publish to
- SPF *(Sender Policy Framework)*
- Stop rule set action
- Subscription management

- Support, request technical
- Templates for custom email verification
- Troubleshooting
- Verified identities
- Virtual Deliverability Manager
- VPC endpoints

FAQs on supported protocols

The Amazon SES Mail Manager service is adding SDK support for the Smithy RPC v2 CBOR protocol in addition to the existing AWS JSON protocol. The Smithy RPC v2 CBOR protocol is more performant than AWS JSON for most workloads. AWS SDKs will prioritize the protocol that is the most performant for each language.

What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ?

A wire protocol acts as a mediator between a client and the service. The AWS SDK handles the serialization (converting an object to the wire format) and deserialization (converting the wire format back to an object) between the client and server transparently for both protocols.

AWS JSON - JSON is one of the most widely used and accepted data formats for communication between systems. This service uses AWS JSON as a medium to communicate between an AWS SDK client and the server. An HTTP request for a service API operation accepts input in the form of JSON, the operation is executed, and the response is returned to the SDK client in the form of JSON.

Smithy RPC v2 CBOR - Smithy RPC v2 CBOR (Concise Binary Object Representation) is a protocol developed by Amazon for more efficient data transmission. It is a data format designed to keep message sizes compact and to allow the format to evolve without requiring version negotiation between systems. The key difference from AWS JSON is that AWS JSON uses human-readable text (like `{"name": "John"}`), while Smithy RPC v2 CBOR uses a binary format (sequences of 1s and 0s). This binary approach creates smaller data packages that travel faster over the network and reduces serialization time since computers process binary data directly.

How do I get started with the Smithy RPC v2 CBOR protocol?

To get started, use the latest AWS SDK/CLI version to use the Smithy RPC v2 CBOR protocol for this service. The SDK will automatically use the Smithy RPC v2 CBOR protocol when available. Upgrade your AWS SDK to the specified version or any subsequent version.

Starting 2026-04-16, the following AWS SDKs will use the Smithy RPC v2 CBOR protocol, while the other SDKs will continue using the AWS JSON protocol:

- AWS SDK for C++
- AWS SDK for Go V2
- AWS SDK for Java 2.x
- AWS SDK for .NET v4
- AWS Tools for PowerShell v5
- AWS SDK for Swift
- AWS SDK for Kotlin

What are the risks of enabling the Smithy RPC v2 CBOR protocol?

The added support of Smithy RPC v2 CBOR in the SDKs is expected to be fully transparent and effortless for customers. However, if you are using a custom implementation of the AWS SDK, or a combination of custom clients and the AWS SDK, that depends on AWS JSON-specific serialization behavior, it may be incompatible with the Smithy RPC v2 CBOR protocol.

What if my solution does not support Smithy RPC v2 CBOR?

To use the AWS JSON protocol, downgrade your SDK to a version released before 2026-04-16.

What AWS Regions support the Smithy RPC v2 CBOR protocol?

The Smithy RPC v2 CBOR protocol is supported in all AWS Regions where this service is available.

What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR?

Smithy RPC v2 CBOR provides up to 13% lower latency and up to 15% reduction in payload size compared to AWS JSON. The binary encoding eliminates the overhead of JSON text parsing and string escaping, which reduces both network transfer time and serialization cost.

Will the AWS JSON protocol be deprecated?

AWS JSON protocol will continue to be supported through older versions of SDKs. Also, some language-specific SDKs will continue using AWS JSON as it is more performant than Smithy RPC

v2 CBOR due to the available JSON implementation for the language being faster than its CBOR implementation.

Where can I find more information about Smithy RPC v2 CBOR?

You can find more information about the AWS JSON protocol at AWS JSON 1.0 protocol and AWS JSON 1.1 protocol in the Smithy documentation, and about Smithy RPC v2 CBOR at Smithy RPC v2 CBOR protocol in the Smithy documentation.