

---

# Amazon Simple Notification Service

## Developer Guide



## Amazon Simple Notification Service: Developer Guide

Copyright © 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.

## Table of Contents

What is Amazon SNS? .....	1
Features and capabilities .....	3
Related services .....	4
Accessing Amazon SNS .....	4
Pricing for Amazon SNS .....	4
Common Amazon SNS scenarios .....	5
Application integration .....	5
Application alerts .....	5
User notifications .....	6
Mobile push notifications .....	6
Using Amazon SNS with an AWS SDK .....	6
Amazon SNS event sources and destinations .....	7
Event sources .....	7
Analytics .....	7
Application integration .....	7
Billing and cost management .....	8
Business applications .....	8
Compute .....	8
Containers .....	9
Customer engagement .....	9
Database .....	10
Developer tools .....	10
Front-end web & mobile .....	11
Game development .....	11
Internet of Things .....	12
Machine learning .....	12
Management & governance .....	13
Media .....	14
Migration & transfer .....	14
Networking & content delivery .....	14
Security, identity, & compliance .....	15
Serverless .....	16
Storage .....	16
Additional event sources .....	17
Event destinations .....	17
A2A destinations .....	17
A2P destinations .....	18
Setting up .....	20
Create account and an IAM administrator user .....	20
Create an IAM user and get credentials .....	20
Next steps .....	21
Getting started .....	22
Prerequisites .....	22
Step 1: Create a topic .....	22
Step 2: Create a subscription to the topic .....	22
Step 3: Publish a message to the topic .....	23
Step 4: Delete the subscription and topic .....	23
Next steps .....	23
Configuring Amazon SNS .....	24
Creating a topic .....	24
AWS Management Console .....	24
AWS SDKs .....	26
Subscribing to a topic .....	31
To subscribe an endpoint to an Amazon SNS topic .....	31

Deleting a subscription and topic .....	31
AWS Management Console .....	32
AWS SDKs .....	32
Configuring tags .....	36
AWS Management Console .....	36
AWS SDKs .....	36
Message ordering and deduplication (FIFO topics) .....	38
FIFO topics use case .....	38
Message ordering details .....	40
Message grouping .....	45
Message delivery .....	46
Message filtering .....	47
Message deduplication .....	49
Message security .....	51
Message durability .....	51
Code examples .....	53
FIFO example (AWS SDKs) .....	53
FIFO example (AWS CloudFormation) .....	55
Message publishing .....	58
AWS Management Console .....	58
AWS SDKs .....	59
Large message payloads .....	65
Prerequisites .....	65
Example: Publishing messages to Amazon SNS with payload stored in Amazon S3 .....	65
Other endpoint protocols .....	67
Message attributes .....	68
Message attribute items and validation .....	68
Data types .....	69
Reserved message attributes for mobile push notifications .....	69
Message filtering .....	71
Subscription filter policies .....	71
Example filter policies .....	72
Filter policy constraints .....	73
Attribute string value matching .....	74
Attribute numeric value matching .....	76
Attribute key matching .....	77
AND/OR logic .....	78
Applying a subscription filter policy .....	79
AWS Management Console .....	79
AWS CLI .....	79
AWS SDKs .....	80
Amazon SNS API .....	81
AWS CloudFormation .....	81
Removing a subscription filter policy .....	82
AWS Management Console .....	82
AWS CLI .....	82
Amazon SNS API .....	82
Message delivery .....	83
Raw message delivery .....	83
Enabling raw message delivery using the AWS Management Console .....	83
Message format examples .....	83
Cross-account delivery .....	84
Queue owner creates subscription .....	84
A user who does not own the queue creates subscription .....	85
Cross-region delivery .....	86
Opt-in Regions .....	87
Message delivery status .....	87

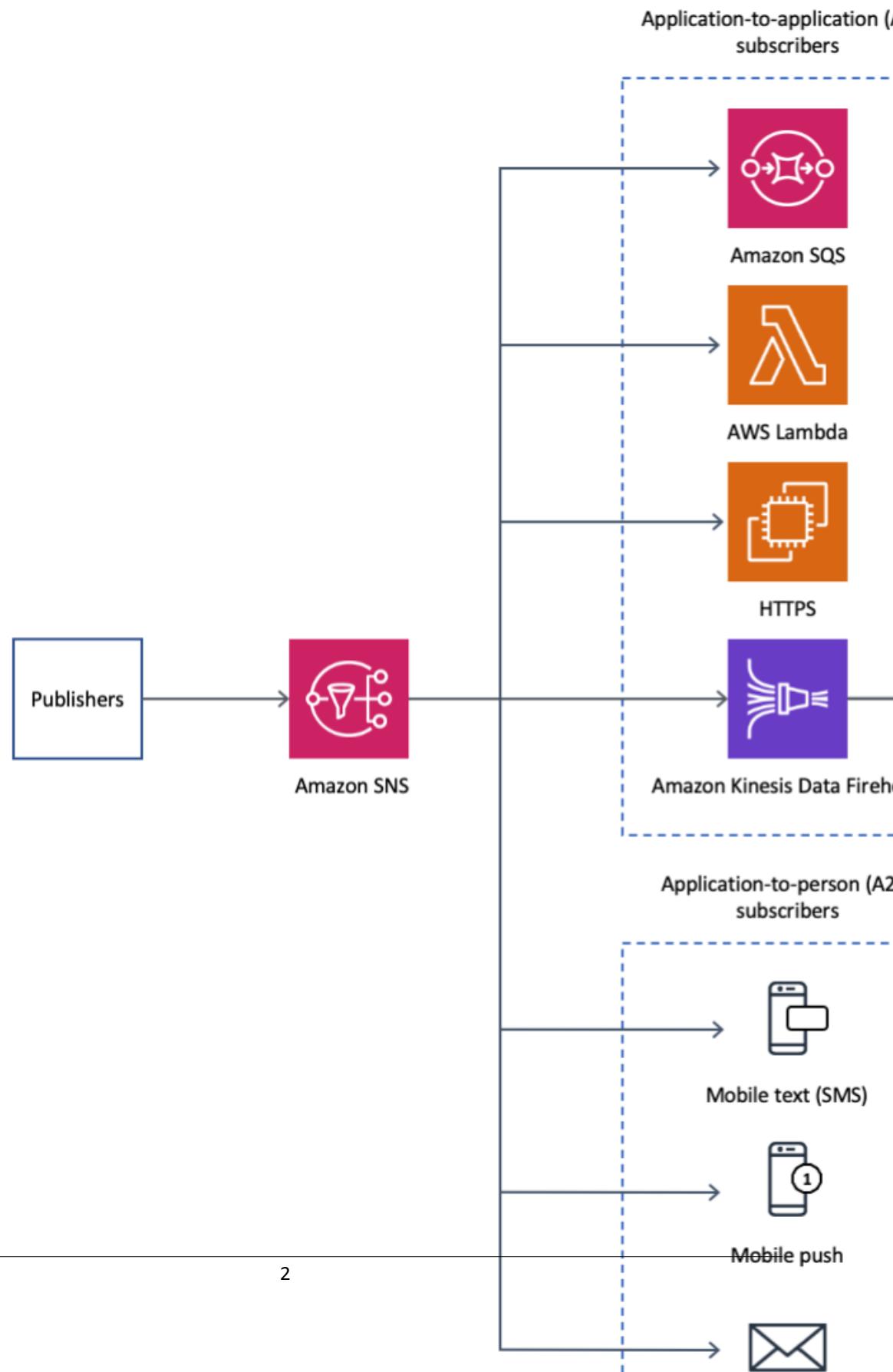
Configuring delivery status logging using the AWS Management Console .....	88
Configuring message delivery status attributes for topics subscribed to Amazon SNS endpoints using the AWS SDKs .....	88
Message delivery retries .....	92
Delivery protocols and policies .....	92
Delivery policy stages .....	93
Creating an HTTP/S delivery policy .....	94
Dead-letter queues (DLQs) .....	96
Why do message deliveries fail? .....	97
How do dead-letter queues work? .....	97
How are messages moved into a dead-letter queue? .....	97
How can I move messages out of a dead-letter queue? .....	98
How can I monitor and log dead-letter queues? .....	98
Configuring a dead-letter queue .....	98
Message archiving and analytics .....	102
Application-to-application (A2A) messaging .....	103
Fanout to Kinesis Data Firehose delivery streams .....	103
Prerequisites .....	103
Subscribing a delivery stream to a topic .....	105
Delivery stream destinations .....	105
Example use case .....	114
Fanout to Lambda functions .....	123
Prerequisites .....	123
Subscribing a function to a topic .....	123
Fanout to Amazon SQS queues .....	124
Subscribing a queue to a topic .....	124
Example (AWS CloudFormation) .....	129
Fanout to HTTP/S endpoints .....	134
Subscribing an endpoint to a topic .....	135
Verifying message signatures .....	140
Parsing message formats .....	142
Fanout to AWS Event Fork Pipelines .....	148
How AWS Event Fork Pipelines works .....	149
Deploying AWS Event Fork Pipelines .....	152
Deploying and testing AWS Event Fork Pipelines .....	153
Subscribing an event pipeline to a topic .....	159
Application-to-person (A2P) messaging .....	166
Mobile text messaging (SMS) .....	166
SMS sandbox .....	167
Origination identities .....	169
Requesting SMS support .....	181
Setting SMS preferences .....	189
Sending SMS messages .....	192
Monitoring SMS activity .....	202
Managing SMS subscriptions .....	207
Supported Regions and countries .....	219
SMS best practices .....	228
SMS requirements for US destinations .....	231
SMS requirements for India .....	231
Mobile push notifications .....	234
How user notifications work .....	235
User notification process overview .....	235
Setting up a mobile app .....	236
Sending mobile push notifications .....	244
Mobile app attributes .....	247
Mobile app events .....	249
Mobile push API actions .....	251

Mobile push API errors .....	252
Mobile push TTL .....	258
Supported Regions .....	260
Email notifications .....	260
AWS Management Console .....	261
AWS SDKs .....	261
Code examples .....	267
Cross-service examples .....	268
Build an app to submit data to a DynamoDB table .....	268
Building an Amazon SNS web application .....	269
Create an Amazon Textract explorer application .....	269
Detect people and objects in a video .....	270
Use API Gateway to invoke a Lambda function .....	271
Use scheduled events to invoke a Lambda function .....	272
Scenario examples .....	272
Create a platform endpoint for push notifications .....	273
Create and publish to a FIFO topic .....	275
Publish SMS messages to a topic .....	277
Publish a large message .....	278
API examples .....	280
Add tags to a topic .....	281
Check whether a phone number is opted out .....	281
Confirm an endpoint owner wants to receive messages .....	284
Create a topic .....	286
Delete a subscription .....	291
Delete a topic .....	294
Get the properties of a topic .....	297
Get the settings for sending SMS messages .....	301
List opted out phone numbers .....	304
List the subscribers of a topic .....	305
List topics .....	310
Publish an SMS text message .....	315
Publish to a topic .....	318
Set a dead-letter queue for a subscription .....	324
Set a filter policy .....	324
Set the default settings for sending SMS messages .....	326
Set topic attributes .....	328
Subscribe a Lambda function to a topic .....	331
Subscribe a mobile application to a topic .....	334
Subscribe an HTTP endpoint to a topic .....	336
Subscribe an email address to a topic .....	337
Security .....	343
Data protection .....	343
Data encryption .....	344
Internetwork traffic privacy .....	352
Identity and access management .....	364
Authentication .....	364
Access control .....	365
Overview .....	366
Using identity-based policies .....	380
Using temporary credentials .....	385
API permissions reference .....	385
Logging and monitoring .....	387
Logging API calls using CloudTrail .....	387
Monitoring topics using CloudWatch .....	390
Compliance validation .....	395
Resilience .....	395

Infrastructure security .....	395
Best practices .....	396
Preventative best practices .....	396
Troubleshooting .....	399
Troubleshooting topics using X-Ray .....	399
Documentation history .....	400
AWS glossary .....	402

# What is Amazon SNS?

Amazon Simple Notification Service (Amazon SNS) is a managed service that provides message delivery from publishers to subscribers (also known as *producers* and *consumers*). Publishers communicate asynchronously with subscribers by sending messages to a *topic*, which is a logical access point and communication channel. Clients can subscribe to the SNS topic and receive published messages using a supported endpoint type, such as Amazon Kinesis Data Firehose, Amazon SQS, AWS Lambda, HTTP, email, mobile push notifications, and mobile text messages (SMS).



## Topics

- [Features and capabilities \(p. 3\)](#)
- [Related services \(p. 4\)](#)
- [Accessing Amazon SNS \(p. 4\)](#)
- [Pricing for Amazon SNS \(p. 4\)](#)
- [Common Amazon SNS scenarios \(p. 5\)](#)
- [Using Amazon SNS with an AWS SDK \(p. 6\)](#)

# Features and capabilities

Amazon SNS provides the following features and capabilities:

- **Application-to-application messaging**

Application-to-application messaging supports subscribers such as Amazon Kinesis Data Firehose delivery streams, Lambda functions, Amazon SQS queues, HTTP/S endpoints, and AWS Event Fork Pipelines. For more information, see [Application-to-application \(A2A\) messaging \(p. 103\)](#).

- **Application-to-person notifications**

Application-to-person notifications provide user notifications to subscribers such as mobile applications, mobile phone numbers, and email addresses. For more information, see [Application-to-person \(A2P\) messaging \(p. 166\)](#).

- **Standard and FIFO topics**

Use a FIFO topic to ensure strict message ordering, to define message groups, and to prevent message duplication. Only Amazon SQS FIFO queues can subscribe to a FIFO topic. For more information, see [Message ordering and deduplication \(FIFO topics\) \(p. 38\)](#).

Use a standard topic when message delivery order and possible message duplication are not critical. All of the supported delivery protocols can subscribe to a standard topic.

- **Message durability**

Amazon SNS uses a number of strategies that work together to provide message durability:

- Published messages are stored across multiple, geographically separated servers and data centers.
- If a subscribed endpoint isn't available, Amazon SNS runs a [delivery retry policy \(p. 92\)](#).
- To preserve any messages that aren't delivered before the delivery retry policy ends, you can create a [dead-letter queue \(p. 96\)](#).

- **Message archiving and analytics**

You can subscribe [Kinesis Data Firehose delivery streams to SNS topics \(p. 103\)](#), which allow you to send notifications to additional archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, and more.

- **Message attributes**

Message attributes let you provide any arbitrary metadata about the message. [the section called "Message attributes" \(p. 68\)](#).

- **Message filtering**

By default, each subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a filter policy to the topic subscription. When the incoming message attributes match the filter policy attributes, the message is delivered to the subscribed endpoint. Otherwise, the message is filtered out. For more information, see [Message filtering \(p. 71\)](#).

- **Message security**

Server-side encryption protects the contents of messages that are stored in Amazon SNS topics, using encryption keys provided by AWS KMS. For more information, see [the section called "Encryption at rest" \(p. 344\)](#).

You can also establish a private connection between Amazon SNS and your virtual private cloud (VPC). For more information, see [the section called "Internetwork traffic privacy" \(p. 352\)](#).

## Related services

You can use the following services with Amazon SNS:

- **Amazon SQS** offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components. Amazon SQS is related to Amazon SNS in the following ways:
  - Amazon SNS provides [dead-letter queues \(p. 96\)](#) powered by Amazon SQS for undeliverable messages.
  - You can [subscribe an Amazon SQS queue to an SNS topic \(p. 124\)](#).
  - You can subscribe an Amazon SQS FIFO queue to an [Amazon SNS FIFO topic \(p. 38\)](#) to receive messages in order and with no duplicates.
- **AWS Lambda** enables you to build applications that respond quickly to new information. Run your application code in Lambda functions on highly available compute infrastructure. For more information, see the [AWS Lambda Developer Guide](#). You can [subscribe a Lambda function to an SNS topic \(p. 123\)](#).
- **AWS Identity and Access Management (IAM)** helps you securely control access to AWS resources for your users. Use IAM to control who can use your Amazon SNS topics (authentication), what topics they can use, and how they can use them (authorization). For more information, see [Using identity-based policies with Amazon SNS \(p. 380\)](#).
- **AWS CloudFormation** enables you to model and set up your AWS resources. Create a template that describes the AWS resources that you want, including Amazon SNS topics and subscriptions. AWS CloudFormation takes care of provisioning and configuring those resources for you. For more information, see the [AWS CloudFormation User Guide](#).

## Accessing Amazon SNS

You can configure and manage SNS topics and subscriptions using the Amazon SNS console, command line tools, or AWS SDKs.

- The [Amazon SNS console](#) provides a convenient user interface for creating topics and subscriptions, sending and receiving messages, and monitoring events and logs.
- The AWS Command Line Interface (AWS CLI) gives you direct access to the Amazon SNS API for advanced configuration and automation use cases. For more information, see [Using Amazon SNS with the AWS CLI](#).
- AWS provides SDKs in various languages. For more information, see [SDKs and Toolkits](#).

## Pricing for Amazon SNS

Amazon SNS has no upfront costs. You pay based on the number of messages that you publish, the number of notifications that you deliver, and any additional API calls for managing topics and

subscriptions. Delivery pricing varies by endpoint type. You can get started for free with the Amazon SNS free tier.

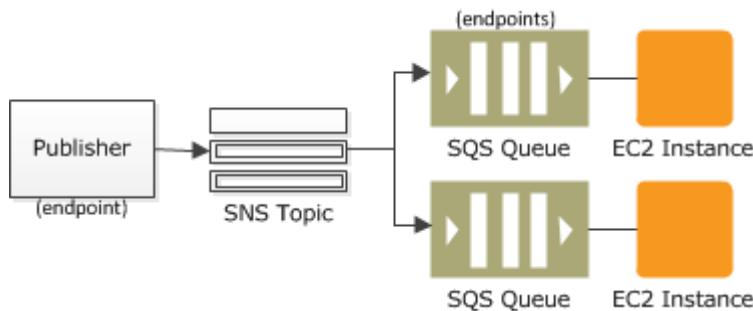
For information, see [Amazon SNS pricing](#).

## Common Amazon SNS scenarios

### Application integration

The *Fanout* scenario is when a message published to an SNS topic is replicated and pushed to multiple endpoints, such as Kinesis Data Firehose delivery streams, Amazon SQS queues, HTTP(S) endpoints, and Lambda functions. This allows for parallel asynchronous processing.

For example, you can develop an application that publishes a message to an SNS topic whenever an order is placed for a product. Then, SQS queues that are subscribed to the SNS topic receive identical notifications for the new order. An Amazon Elastic Compute Cloud (Amazon EC2) server instance attached to one of the SQS queues can handle the processing or fulfillment of the order. And you can attach another Amazon EC2 server instance to a data warehouse for analysis of all orders received.



You can also use fanout to replicate data sent to your production environment with your test environment. Expanding upon the previous example, you can subscribe another SQS queue to the same SNS topic for new incoming orders. Then, by attaching this new SQS queue to your test environment, you can continue to improve and test your application using data received from your production environment.

**Important**

Make sure that you consider data privacy and security before you send any production data to your test environment.

For more information, see the following resources:

- [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#)
- [Fanout to Lambda functions \(p. 123\)](#)
- [Fanout to Amazon SQS queues \(p. 124\)](#)
- [Fanout to HTTP/S endpoints \(p. 134\)](#)
- [Event-Driven Computing with Amazon SNS and AWS Compute, Storage, Database, and Networking Services](#)

### Application alerts

Application and system alerts are notifications that are triggered by predefined thresholds. Amazon SNS can send these notifications to specified users via SMS and email. For example, you can receive immediate notification when an event occurs, such as a specific change to your Amazon EC2 Auto

Scaling group, a new file uploaded to an Amazon S3 bucket, or a metric threshold breached in Amazon CloudWatch. For more information, see [Setting up Amazon SNS notifications in the Amazon CloudWatch User Guide](#).

## User notifications

Amazon SNS can send push email messages and text messages (SMS messages) to individuals or groups. For example, you could send e-commerce order confirmations as user notifications. For more information about using Amazon SNS to send SMS messages, see [Mobile text messaging \(SMS\) \(p. 166\)](#).

## Mobile push notifications

Mobile push notifications enable you to send messages directly to mobile apps. For example, you can use Amazon SNS to send update notifications to an app. The notification message can include a link to download and install the update. For more information about using Amazon SNS to send push notification messages, see [Mobile push notifications \(p. 234\)](#).

# Using Amazon SNS 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.

SDK documentation	Code examples
<a href="#">AWS SDK for C++</a>	<a href="#">AWS SDK for C++ code examples</a>
<a href="#">AWS SDK for Go</a>	<a href="#">AWS SDK for Go code examples</a>
<a href="#">AWS SDK for Java</a>	<a href="#">AWS SDK for Java code examples</a>
<a href="#">AWS SDK for JavaScript</a>	<a href="#">AWS SDK for JavaScript code examples</a>
<a href="#">AWS SDK for .NET</a>	<a href="#">AWS SDK for .NET code examples</a>
<a href="#">AWS SDK for PHP</a>	<a href="#">AWS SDK for PHP code examples</a>
<a href="#">AWS SDK for Python (Boto3)</a>	<a href="#">AWS SDK for Python (Boto3) code examples</a>
<a href="#">AWS SDK for Ruby</a>	<a href="#">AWS SDK for Ruby code examples</a>

For examples specific to Amazon SNS, see [Code examples for Amazon SNS \(p. 267\)](#).

### Example availability

Can't find what you need? Request a code example with the feedback link.

# Amazon SNS event sources and destinations

Amazon SNS can receive event-driven notifications from many AWS sources and fan out notifications to application-to-application (A2A) and application-to-person (A2P) destinations. This section lists supported event sources and destinations, and provides links for more information.

## Topics

- [Amazon SNS event sources \(p. 7\)](#)
- [Amazon SNS event destinations \(p. 17\)](#)

## Amazon SNS event sources

This page lists the AWS services that can publish events to Amazon SNS topics, grouped by their [AWS product categories](#).

### Note

Amazon SNS introduced [FIFO topics \(p. 38\)](#) in October, 2020. Currently, most AWS services support sending events to standard topics only.

## Analytics services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Athena</a> – Allows you to analyze data in Amazon S3 using standard SQL.	Receive notifications when control limits are exceeded. For more information, see <a href="#">Setting data usage control limits</a> in the <i>Amazon Athena User Guide</i> .
<a href="#">AWS Data Pipeline</a> – Helps automate the movement and transformation of data.	Receive notifications about the status of pipeline components. For more information, see <a href="#">SnsAlarm</a> in the <i>AWS Data Pipeline Developer Guide</i> .
<a href="#">Amazon Redshift</a> – Manages all of the work of setting up, operating, and scaling a data warehouse.	Receive notifications of Amazon Redshift events. For more information, see <a href="#">Amazon Redshift event notifications</a> in the <i>Amazon Redshift Cluster Management Guide</i> .

## Application integration services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon EventBridge</a> – Delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services and routes that data to targets, including Amazon	Receive notifications of EventBridge events. For more information, see <a href="#">Amazon EventBridge targets</a> in the <i>Amazon EventBridge User Guide</i> .

AWS service	Benefit of using with Amazon SNS
SNS. EventBridge was formerly called CloudWatch Events.	
<a href="#">AWS Step Functions</a> – Lets you combine AWS Lambda functions and other AWS services to build business-critical applications.	Receive notification of Step Functions events. For more information, see <a href="#">Call Amazon SNS with Step Functions</a> in the <i>AWS Step Functions Developer Guide</i> .

## Billing & cost management services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Billing and Cost Management</a> – Provides features that help you monitor your costs and pay your bill.	<p>Receive budget notifications, price change notifications, and anomaly alerts. For more information, see the following pages in the AWS Billing and Cost Management User Guide:</p> <ul style="list-style-type: none"> <li>• <a href="#">Creating an Amazon SNS topic for budget notifications</a></li> <li>• <a href="#">Setting up notifications</a></li> <li>• <a href="#">Detecting unusual spend with AWS Cost Anomaly Detection</a></li> </ul>

## Business applications services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Chime</a> – Lets you meet, chat, and place business calls inside and outside of your organization.	Receive important meeting event notifications. For more information, see <a href="#">Amazon Chime SDK event notifications</a> in the <i>Amazon Chime Developer Guide</i> .

## Compute services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon EC2 Auto Scaling</a> – Helps you have the correct number of Amazon Elastic Compute Cloud (Amazon EC2) instances available for handling your application's load.	Receive notifications when Auto Scaling launches or terminates Amazon EC2 instances in your Auto Scaling group. For more information, see <a href="#">Getting Amazon SNS notifications when your Auto Scaling group scales</a> in the <i>Amazon EC2 Auto Scaling User Guide</i> .
<a href="#">EC2 Image Builder</a> – Helps automate the creation, management, and deployment of customized, secure, and up-to-date server images that are pre-installed and pre-configured with software and settings to meet specific IT standards.	Receive notifications when builds are complete. For more information, see <a href="#">Tracking the latest server images in EC2 Image Builder pipelines</a> on the <i>AWS Compute Blog</i> .

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Elastic Beanstalk</a> – Handles the details of capacity provisioning, load balancing, and scaling for your application, and provides application health monitoring.	Receive notifications of important events that affect your application. For more information, see <a href="#">Elastic Beanstalk environment notifications with Amazon SNS</a> in the <i>AWS Elastic Beanstalk Developer Guide</i> .
<a href="#">AWS Lambda</a> – Lets you run code without provisioning or managing servers.	Receive function output data by setting an SNS topic as a Lambda dead-letter queue or a Lambda destination. For more information, see <a href="#">Asynchronous invocation</a> in the <i>AWS Lambda Developer Guide</i> .
<a href="#">Amazon Lightsail</a> – Helps developers get started using AWS to build websites or web applications.	Receive notifications when a metric for one of your instances, databases, or load balancers crosses a specified threshold. For more information, see <a href="#">Adding notification contacts in Amazon Lightsail</a> in the <i>Amazon Lightsail Developer Guide</i> .

## Containers services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon EKS Distro</a> – Lets you create reliable and secure clusters wherever your applications are deployed.	Track updates and security patches for clusters created with Amazon EKS Distro. For more information, see <a href="#">Introducing Amazon EKS Distro - an open source Kubernetes distribution used by Amazon EKS</a> .
<a href="#">Amazon Elastic Container Service (Amazon ECS)</a> – Enables you to run, stop, and manage containers on a cluster.	Receive notifications when a new Amazon ECS-optimized AMI is available. For more information, see <a href="#">Subscribing to Amazon ECS-optimized AMI update notifications</a> in the <i>Amazon Elastic Container Service Developer Guide</i> .

## Customer engagement services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Connect</a> – Lets you set up an omnichannel cloud contact center to engage with your customers.	Receive alerts and validations. For more information, see <a href="#">The power of AWS with Amazon Connect</a> in the <i>Amazon Connect Administrator Guide</i> .
<a href="#">Amazon Pinpoint</a> – Helps you engage your customers by sending them email, SMS and voice messages, and push notifications.	Configure two-way SMS, which allows you to receive messages from your customers. For more information, see <a href="#">Using two-way SMS messaging in Amazon Pinpoint</a> in the <i>Amazon Pinpoint User Guide</i> .

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Simple Email Service (Amazon SES)</a> – Provides cost-effective way for you to send and receive email using your own email addresses and domains.	Receive notifications of bounces, complaints, and deliveries. For more information, see <a href="#">Configuring Amazon SNS notifications for Amazon SES</a> in the <i>Amazon Simple Email Service Developer Guide</i> .

## Database services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Database Migration Service</a> – Migrates data from on-premises databases into the AWS Cloud.	Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see <a href="#">Working with events and notifications in AWS Database Migration Service</a> in the <i>AWS Database Migration Service User Guide</i> .
<a href="#">Amazon DynamoDB</a> – Provides fast and predictable performance with seamless scalability in this fully managed NoSQL database service.	Receive notifications when maintenance events occur. For more information, see <a href="#">Customizing DAX cluster settings</a> in the <i>Amazon DynamoDB Developer Guide</i> .
<a href="#">Amazon ElastiCache</a> – Provides a high performance, resizable, and cost-effective in-memory cache, while removing complexity associated with deploying and managing a distributed cache environment.	Receive notifications when significant events occur. For more information, see <a href="#">Event notifications and Amazon SNS</a> in the <i>Amazon ElastiCache for Memcached User Guide</i> .
<a href="#">Amazon Neptune</a> – Enables you to build and run applications that work with highly connected datasets.	Receive notifications when a Neptune event occurs. For more information, see <a href="#">Using Neptune event notification</a> in the <i>Neptune User Guide</i> .
<a href="#">Amazon Redshift</a> – Manages all of the work of setting up, operating, and scaling a data warehouse.	Receive notifications of Amazon Redshift events. For more information, see <a href="#">Amazon Redshift event notifications</a> in the <i>Amazon Redshift Cluster Management Guide</i> .
<a href="#">Amazon Relational Database Service</a> – Makes it easier to set up, operate, and scale a relational database in the AWS Cloud.	Receive notifications of Amazon RDS events. For more information, see <a href="#">Using Amazon RDS event notification</a> in the <i>Amazon RDS User Guide</i> .

## Developer tools services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS CodeBuild</a> – Compiles your source code, runs unit tests, and produces artifacts that are ready to deploy.	Receive notifications when builds succeed, fail, or move from one build phase to another. For more information, see <a href="#">Build notifications sample for CodeBuild</a> in the <i>AWS CodeBuild User Guide</i> .
<a href="#">AWS CodeCommit</a> – Provides version control for privately storing and managing assets in the cloud.	Receive notifications about CodeCommit repository events. For more information, see <a href="#">Example: Create an AWS CodeCommit trigger for</a>

AWS service	Benefit of using with Amazon SNS
	an <a href="#">Amazon SNS topic</a> in the <a href="#">AWS CodeCommit User Guide</a> .
<a href="#">AWS CodeDeploy</a> – Automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.	Receive notifications for CodeDeploy deployments or instance events. For more information, see <a href="#">Create a trigger for a CodeDeploy event</a> in the <a href="#">AWS CodeDeploy User Guide</a> .
<a href="#">Amazon CodeGuru</a> – Collects runtime performance data from your live applications, and provides recommendations that can help you fine-tune your application performance.	Receive notifications when anomalies occur. For more information, see <a href="#">Working with anomalies and recommendation reports</a> in the <a href="#">Amazon CodeGuru User Guide</a> .
<a href="#">AWS CodePipeline</a> – Automates the steps required to release software changes continuously.	Receive notifications about approval actions. For more information, see <a href="#">Manage approval actions</a> in <a href="#">CodePipeline</a> in the <a href="#">AWS CodePipeline User Guide</a> .
<a href="#">AWS CodeStar</a> – Create, manage, and work with software development projects on AWS.	Receive notifications about events that occur in the resources that you use. For more information, see <a href="#">Configure Amazon SNS topics for notifications</a> in the <a href="#">Developer Tools Console User Guide</a> .

## Front-end web & mobile services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Pinpoint</a> – Helps you engage your customers by sending them email, SMS and voice messages, and push notifications.	Configure two-way SMS, which allows you to receive messages from your customers. For more information, see <a href="#">Using two-way SMS messaging in Amazon Pinpoint</a> in the <a href="#">Amazon Pinpoint User Guide</a> .

## Game development services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon GameLift</a> – Provides solutions for hosting session-based multiplayer game servers in the cloud, including a fully managed service for deploying, operating, and scaling game servers.	<p>Receive matchmaking and queue event notifications. For more information, see the following pages:</p> <ul style="list-style-type: none"> <li>• For matchmaking notifications, see <a href="#">Set up FlexMatch event notification</a> in the <a href="#">Amazon GameLift FlexMatch Developer Guide</a>.</li> <li>• For queue notifications, see <a href="#">Set up event notification for game session placement</a> in the <a href="#">Amazon GameLift Developer Guide</a>.</li> </ul>

## Internet of Things services

<b>AWS service</b>	<b>Benefit of using with Amazon SNS</b>
<a href="#">AWS IoT Core</a> – Provides the cloud services that connect your IoT devices to other devices and AWS Cloud services.	Receive notifications of AWS IoT Core events. For more information, see <a href="#">Creating an Amazon SNS rule</a> in the <i>AWS IoT Developer Guide</i> .
<a href="#">AWS IoT Device Defender</a> – Allows you to audit the configuration of your devices, monitor connected devices to detect abnormal behavior, and mitigate security risks.	Receive alarms when a device violates a behavior. For more information, see <a href="#">How to use AWS IoT Device Defender detect</a> in the <i>AWS IoT Developer Guide</i> .
<a href="#">AWS IoT Events</a> – Lets you monitor your equipment or device fleets for failures or changes in operation, and trigger actions when such events occur.	Receive notifications of AWS IoT Events events. For more information, see <a href="#">Amazon Simple Notification Service</a> in the <i>AWS IoT Events Developer Guide</i> .
<a href="#">AWS IoT Greengrass</a> – Extends AWS onto physical devices so they can act locally on the data they generate, while still using the cloud for management, analytics, and durable storage.	Receive notifications of AWS IoT Greengrass events. For more information, see <a href="#">SNS connector</a> in the <i>AWS IoT Greengrass Version 1 Developer Guide</i> .

## Machine learning services

<b>AWS service</b>	<b>Benefit of using with Amazon SNS</b>
<a href="#">Amazon CodeGuru</a> – Collects runtime performance data from your live applications, and provides recommendations that can help you fine-tune your application performance.	Receive notifications when anomalies occur. For more information, see <a href="#">Working with anomalies and recommendation reports</a> in the <i>Amazon CodeGuru User Guide</i> .
<a href="#">Amazon DevOps Guru</a> – Generates operational insights using machine learning to help you improve the performance of your operational applications.	Forward insights and confirmations. For more information, see <a href="#">Deliver ML-powered operational insights to your on-call teams via PagerDuty with Amazon DevOps Guru</a> on the <i>AWS Management &amp; Governance Blog</i> .
<a href="#">Amazon Lookout for Metrics</a> – Finds anomalies in your data, determines their root causes, and enables you to quickly take action.	Receive notifications of anomalies. For more information, see <a href="#">Using Amazon SNS with Lookout for Metrics</a> in the <i>Amazon Lookout for Metrics Developer Guide</i> .
<a href="#">Amazon Rekognition</a> – Lets you add image and video analysis to your applications	Receive notifications of request results. For more information, see <a href="#">Reference: Video analysis results notification</a> in the <i>Amazon Rekognition Developer Guide</i> .
<a href="#">Amazon SageMaker</a> – Enables data scientists and developers to build and train machine learning models, and then directly deploy them into a production-ready hosted environment.	Receive notifications when a data object is labeled. For more information, see <a href="#">Creating a streaming labeling job</a> in the <i>Amazon SageMaker Developer Guide</i> .

## Management & governance services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Chatbot</a> – Enables DevOps and software development teams to use Amazon Chime and Slack chat rooms to monitor and respond to operational events in the AWS Cloud.	Deliver notifications to chat rooms. For more information, see <a href="#">Setting up AWS Chatbot</a> in the <a href="#">AWS Chatbot Administrator Guide</a> .
<a href="#">AWS CloudFormation</a> – Enables you to create and provision AWS infrastructure deployments predictably and repeatedly.	Receive notifications when stacks are created and updated. For more information, see <a href="#">Setting AWS CloudFormation stack options</a> in the <a href="#">AWS CloudFormation User Guide</a> .
<a href="#">AWS CloudTrail</a> – Provides event history of your AWS account activity.	Receive notifications when CloudTrail publishes new log files to your Amazon S3 bucket. For more information, see <a href="#">Configuring Amazon SNS notifications for CloudTrail</a> in the <a href="#">AWS CloudTrail User Guide</a> .
<a href="#">Amazon CloudWatch</a> – Monitors your AWS resources and the applications you run on AWS in real time.	Receive notifications when alarms change state. For more information, see <a href="#">Using Amazon CloudWatch alarms</a> in the <a href="#">Amazon CloudWatch User Guide</a> .
<a href="#">AWS Config</a> – Provides a detailed view of the configuration of AWS resources in your AWS account.	Receive notifications when resources are updated, or when AWS Config evaluates custom or managed rules against your resources. For more information, see <a href="#">Notifications that AWS Config sends to an SNS topic</a> and <a href="#">Example configuration item change notifications</a> in the <a href="#">AWS Config Developer Guide</a> .
<a href="#">AWS Control Tower</a> – Enables you to set up and govern a secure, compliant, multi-account AWS environment.	Use alerts to help you prevent drift within your landing zone, and receive compliance notifications. For more information, see <a href="#">Tracking alerts through Amazon Simple Notification Service</a> in the <a href="#">AWS Control Tower User Guide</a> .
<a href="#">AWS License Manager</a> – Helps you manage your software licenses from software vendors centrally across AWS and your on-premises environments.	Receive License Manager notifications and alerts. For more information, see <a href="#">Settings in License Manager</a> in the <a href="#">License Manager User Guide</a> and <a href="#">Creating ServiceNow incidents for AWS License Manager notifications</a> on the <a href="#">AWS Management &amp; Governance Blog</a> .
<a href="#">AWS Service Catalog</a> – Enables IT administrators to create, manage, and distribute portfolios of approved products to end users, who can then access the products they need in a personalized portal.	Receive notifications about stack events. For more information, see <a href="#">AWS Service Catalog notification constraints</a> in the <a href="#">AWS Service Catalog Administrator Guide</a> .
<a href="#">AWS Systems Manager</a> – Lets you view and control your infrastructure on AWS.	Receive notifications about the status of commands. For more information, see <a href="#">Monitoring Systems Manager status changes using Amazon SNS notifications</a> in the <a href="#">AWS Systems Manager User Guide</a> .

## Media services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon Elastic Transcoder</a> – Lets you convert media files that you stored in Amazon S3 into media files in the formats required by consumer playback devices.	Receive notifications when jobs change status. For more information, see <a href="#">Notifications of job status</a> in the <i>Amazon Elastic Transcoder Developer Guide</i> .

## Migration & transfer services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Application Discovery Service</a> – Helps you plan your migration to the AWS Cloud by collecting usage and configuration data about your on-premises servers.	Receive notifications of events through AWS CloudTrail. For more information, see <a href="#">Logging Application Discovery Service API calls with AWS CloudTrail</a> in the <i>Application Discovery Service User Guide</i> .
<a href="#">AWS Database Migration Service</a> – Migrates data from on-premises databases into the AWS Cloud.	Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see <a href="#">Working with events and notifications in AWS Database Migration Service</a> in the <i>AWS Database Migration Service User Guide</i> .
<a href="#">AWS Snowball</a> – Uses physical storage devices to transfer large amounts of data between Amazon S3 and your onsite data storage location at faster-than-internet speeds.	Receive notifications for Snowball jobs. For more information, see the following: <ul style="list-style-type: none"> <li>• <a href="#">Snowball notifications</a> in the <i>AWS Snowball User Guide</i></li> <li>• <a href="#">Step 5: Choose your notification preferences</a> in the <i>AWS Snowball Edge Developer Guide</i></li> <li>• <a href="#">Step 5: Choose your notification preferences</a> in the <i>AWS Snowcone User Guide</i></li> </ul>

## Networking & content delivery services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon API Gateway</a> – Enables you to create and deploy your own REST and WebSocket APIs at any scale.	Receive messages posted to an API Gateway endpoint. For more information, see <a href="#">Tutorial: Build an API Gateway REST API with AWS integration</a> in the <i>API Gateway Developer Guide</i> .
<a href="#">Amazon CloudFront</a> – Speeds up distribution of your static and dynamic web content, such as .html, .css, .php, image, and media files.	Receive notifications when alarms based on specified CloudFront metrics occur. For more information, see <a href="#">Setting alarms to receive notifications</a> in the <i>Amazon CloudFront Developer Guide</i> .

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Direct Connect</a> – Links your internal network to an AWS Direct Connect location over a standard Ethernet fiber-optic cable.	Receive notifications when alarms that monitor the state of an AWS Direct Connect connection change state. For more information, see <a href="#">Creating CloudWatch alarms to monitor AWS Direct Connect connections</a> in the <i>AWS Direct Connect User Guide</i> .
<a href="#">Elastic Load Balancing</a> – Automatically distributes your incoming traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in one or more Availability Zones.	Receive notifications of alarms you've created for load balancer events. For more information, see <a href="#">Create CloudWatch alarms for your load balancer</a> in the <i>User Guide for Classic Load Balancers</i> .
<a href="#">Amazon Route 53</a> – Provides domain registration, DNS routing, and health checking.	Receive notifications when health check status is unhealthy. For more information, see <a href="#">To receive an Amazon SNS notification when a health check status is unhealthy (console)</a> in the <i>Amazon Route 53 Developer Guide</i> .
<a href="#">Amazon Virtual Private Cloud (Amazon VPC)</a> – Enables you to launch AWS resources into a virtual network that you've defined.	Receive notifications for specific events that occur on interface endpoints. For more information, see <a href="#">Create and manage a notification for an endpoint service</a> in the <i>Amazon VPC User Guide</i> .

## Security, identity, & compliance services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Directory Service</a> – Provides multiple ways to use Microsoft Active Directory (AD) with other AWS services.	Receive email or text (SMS) messages when the status of your directory changes. For more information, see <a href="#">Configure directory status notifications</a> in the <i>AWS Directory Service Administration Guide</i> .
<a href="#">Amazon GuardDuty</a> – Provides continuous security monitoring to help to identify unexpected and potentially unauthorized or malicious activity in your AWS environment.	Receive notifications about newly released finding types, updates to the existing finding types, and other functionality changes. For more information, see <a href="#">Subscribing to GuardDuty announcements SNS topic</a> in the <i>Amazon GuardDuty User Guide</i> .
<a href="#">Amazon Inspector</a> – Tests the network accessibility of your Amazon EC2 instances and the security state of your applications that run on those instances.	Receive notifications for Amazon Inspector events. For more information, see <a href="#">Setting up an SNS topic for Amazon Inspector notifications</a> in the <i>Amazon Inspector User Guide</i> .

## Serverless services

AWS service	Benefit of using with Amazon SNS
<a href="#">Amazon DynamoDB</a> – Provides fast and predictable performance with seamless scalability in this fully managed NoSQL database service.	Receive notifications when maintenance events occur. For more information, see <a href="#">Customizing DAX cluster settings</a> in the <i>Amazon DynamoDB Developer Guide</i> .
<a href="#">Amazon EventBridge</a> – Delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services and routes that data to targets, including Amazon SNS. EventBridge was formerly called CloudWatch Events.	Receive notifications of EventBridge events. For more information, see <a href="#">Amazon EventBridge targets</a> in the <i>Amazon EventBridge User Guide</i> .
<a href="#">AWS Lambda</a> – Lets you run code without provisioning or managing servers.	Receive function output data by setting an SNS topic as a Lambda dead-letter queue or a Lambda destination. For more information, see <a href="#">Asynchronous invocation</a> in the <i>AWS Lambda Developer Guide</i> .

## Storage services

AWS service	Benefit of using with Amazon SNS
<a href="#">AWS Backup</a> – Helps you centralize and automate the backup of data across AWS services in the cloud and on premises	Receive notifications of AWS Backup events. For more information, see <a href="#">Using Amazon SNS to track AWS Backup events</a> in the <i>AWS Backup Developer Guide</i> .
<a href="#">Amazon Elastic File System</a> – Provides file storage for your Amazon EC2 instances.	Receive notifications of alarms you've created for Amazon EFS events. For more information, see <a href="#">Automated monitoring tools</a> in the <i>Amazon Elastic File System User Guide</i> .
<a href="#">Amazon S3 Glacier</a> – Provides storage for infrequently used data.	Set a notification configuration on a vault so that when a job completes, a message is sent to an SNS topic. For more information, see <a href="#">Configuring vault notifications in Amazon S3 Glacier</a> in the <i>Amazon S3 Glacier Developer Guide</i> .
<a href="#">Amazon Simple Storage Service (Amazon S3)</a> – Provides object storage.	Receive notifications when changes occur to an Amazon S3 bucket or in the rare instance when objects don't replicate to their destination Region. For more information, see <a href="#">Walkthrough: Configure a bucket for notifications (SNS topic or SQS queue)</a> and <a href="#">Monitoring progress with replication metrics and Amazon S3 event notifications</a> in the <i>Amazon Simple Storage Service User Guide</i> .
<a href="#">AWS Snowball</a> – Uses physical storage devices to transfer large amounts of data between Amazon S3 and your onsite data storage location at faster-than-internet speeds.	Receive notifications for Snowball jobs. For more information, see the following:

AWS service	Benefit of using with Amazon SNS
	<ul style="list-style-type: none"> <li>• <a href="#">Snowball notifications</a> in the <i>AWS Snowball User Guide</i></li> <li>• <a href="#">Step 5: Choose your notification preferences</a> in the <i>AWS Snowball Edge Developer Guide</i></li> <li>• <a href="#">Step 5: Choose your notification preferences</a> in the <i>AWS Snowcone User Guide</i></li> </ul>

## Additional event sources

Source	Benefit of using with Amazon SNS
<a href="#">AWS IP address ranges</a>	Receive notifications of changes to AWS IP ranges. For more information, see <a href="#">AWS IP address ranges notifications</a> in the <i>Amazon Web Services General Reference</i> .

For more information on event-driven computing, see the following sources:

- [What is an Event-Driven Architecture?](#)
- [Event-Driven Computing with Amazon SNS and AWS Compute, Storage, Database, and Networking Services](#) on the *AWS Compute Blog*
- [Enriching Event-Driven Architectures with AWS Event Fork Pipelines](#) on the *AWS Compute Blog*

## Amazon SNS event destinations

This page lists all destinations that can receive information on events, grouped by [application-to-application \(A2A\) messaging \(p. 103\)](#) and [application-to-person \(A2P\) notifications \(p. 166\)](#).

**Note**

Amazon SNS introduced [FIFO topics \(p. 38\)](#) in October, 2020. Currently, most AWS services support receiving events from SNS standard topics only. Amazon SQS supports receiving events from both SNS standard and FIFO topics.

## A2A destinations

Event destination	Benefit of using with Amazon SNS
<a href="#">Amazon Kinesis Data Firehose</a>	Deliver events to delivery streams for archiving and analysis purposes. Through delivery streams, you can deliver events to AWS destinations like Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and Amazon OpenSearch Service (OpenSearch Service), or to third-party destinations such as Datadog, New Relic, MongoDB, and Splunk. For more information, see <a href="#">Fanout to Kinesis Data Firehose delivery streams (p. 103)</a> .
<a href="#">AWS Lambda</a>	Deliver events to functions for triggering the execution of custom business logic. For

Event destination	Benefit of using with Amazon SNS
	more information, see <a href="#">Fanout to Lambda functions (p. 123)</a> .
Amazon SQS	Deliver events to queues for application integration purposes. For more information, see <a href="#">Fanout to Amazon SQS queues (p. 124)</a> .
AWS Event Fork Pipelines	Deliver events to event backup and storage, event search and analytics, or event replay pipelines. For more information, see <a href="#">Fanout to AWS Event Fork Pipelines (p. 148)</a> .
HTTP/S	Deliver events to external webhooks. For more information, see <a href="#">Fanout to HTTP/S endpoints (p. 134)</a> .

## A2P destinations

Event destination	Benefit of using with Amazon SNS
SMS	Deliver events to mobile phones as text messages. For more information, see <a href="#">Mobile text messaging (SMS) (p. 166)</a> .
Email	Deliver events to inboxes as email messages. For more information, see <a href="#">Email notifications (p. 260)</a> .
Platform endpoint	Deliver events to mobile phones as native push notifications. For more information, see <a href="#">Mobile push notifications (p. 234)</a> .
AWS Chatbot	Deliver events to Amazon Chime chat rooms or Slack channels. For more information, see the following pages in the <i>AWS Chatbot Administrator Guide</i> : <ul style="list-style-type: none"> <li>• <a href="#">Setting up AWS Chatbot with Amazon Chime</a></li> <li>• <a href="#">Setting up AWS Chatbot with Slack</a></li> <li>• <a href="#">Using AWS Chatbot with other AWS services</a></li> </ul>
PagerDuty	Deliver operational insights to on-call teams. For more information, see <a href="#">Deliver ML-powered operational insights to your on-call teams via PagerDuty with Amazon DevOps Guru</a> on the <a href="#">AWS Management &amp; Governance Blog</a> .

### Note

You can deliver both native AWS events and custom events to chat apps:

- **Native AWS events** – You can use AWS Chatbot to send native AWS events, through Amazon SNS topics, to Amazon Chime and Slack. The supported set of native AWS events includes events from AWS Billing and Cost Management, AWS Health, AWS CloudFormation, Amazon

CloudWatch, and more. For more information, see [Using AWS Chatbot with other services](#) in the *AWS Chatbot Administrator Guide*.

- **Custom events** – You can also send your custom events, through Amazon SNS topics, to Amazon Chime, Slack, and Microsoft Teams. To do this, you publish custom events to an SNS topic, which delivers the events to a subscribed Lambda function. The Lambda function then uses the chat app's webhook to deliver the events to recipients. For more information, see [How do I use webhooks to publish Amazon SNS messages to Amazon Chime, Slack, or Microsoft Teams?](#)

# Setting up access for Amazon SNS

Before you can use Amazon SNS, you must complete the following steps.

## Topics

- [Step 1: Create an AWS account and an IAM administrator user \(p. 20\)](#)
- [Step 2: Create an IAM user and get your AWS credentials \(p. 20\)](#)
- [Next steps \(p. 21\)](#)

## Step 1: Create an AWS account and an IAM administrator user

To access any AWS service, you must first create an [AWS account](#). You can use your AWS account to view your activity and usage reports and to manage authentication and access.

1. Navigate to the [AWS home page](#), and then choose **Create an AWS Account**.
2. Follow the instructions.  
  
Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.
3. When you finish creating your AWS account, follow the instructions in the *IAM User Guide* to [create your first IAM administrator user and group](#).

## Step 2: Create an IAM user and get your AWS credentials

To avoid using your IAM administrator user for Amazon SNS operations, it is a best practice to create an IAM user for each person who needs administrative access to Amazon SNS.

To work with Amazon SNS, you need the `AmazonSNSFullAccess` policy and AWS credentials that are associated with your IAM user. These credentials are comprised of an access key ID and a secret access key. For more information, see [What Is IAM?](#) in the *IAM User Guide* and [AWS Security Credentials](#) in the *AWS General Reference*.

1. Sign in to the [AWS Identity and Access Management console](#).
2. Choose **Users, Add user**.
3. Type a **User name**, such as `AmazonSNSAdmin`.
4. Select **Programmatic access** and **AWS Management Console access**.
5. Set a **Console password** and then choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Attach existing policies directly**.
7. Type `AmazonSNS` into the filter, choose `AmazonSNSFullAccess`, and then choose **Next: Tags**.
8. On the **Add tags (optional)** page, choose **Next: Review**.
9. On the **Review** page, choose **Create user**.

The IAM user is created and the **Access key ID** is displayed, for example:

#### AKIAIOSFODNN7EXAMPLE

10. To display your **Secret access key**, choose **Show**, for example:

wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

**Important**

You can view or download your secret access key *only* when you create your credentials (however, you can create new credentials at any time).

11. To download your credentials, choose **Download .csv**. Keep this file in a secure location.

## Next steps

Now that you're prepared to work with Amazon SNS, [get started \(p. 22\)](#) by creating a topic, creating a subscription for the topic, publishing a message to the topic, and deleting the subscription and topic.

# Getting started with Amazon SNS

This section helps you become more familiar with Amazon SNS by showing you how to manage topics, subscriptions, and messages using the Amazon SNS console.

## Topics

- [Prerequisites \(p. 22\)](#)
- [Step 1: Create a topic \(p. 22\)](#)
- [Step 2: Create a subscription to the topic \(p. 22\)](#)
- [Step 3: Publish a message to the topic \(p. 23\)](#)
- [Step 4: Delete the subscription and topic \(p. 23\)](#)
- [Next steps \(p. 23\)](#)

## Prerequisites

Before you begin, complete the steps in [Setting up access for Amazon SNS \(p. 20\)](#).

## Step 1: Create a topic

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, choose **Create topic**.
4. By default, the console creates a FIFO topic. Choose **Standard**.
5. In the **Details** section, enter a **Name** for the topic, such as *MyTopic*.
6. Scroll to the end of the form and choose **Create topic**.

The console opens the new topic's **Details** page.

## Step 2: Create a subscription to the topic

1. In the left navigation pane, choose **Subscriptions**.
  2. On the **Subscriptions** page, choose **Create subscription**.
  3. On the **Create subscription** page, choose the **Topic ARN** field to see a list of the topics in your AWS account.
  4. Choose the topic that you created in the previous step.
  5. For **Protocol**, choose **Email**.
  6. For **Endpoint**, enter an email address that can receive notifications.
  7. Choose **Create subscription**.
- The console opens the new subscription's **Details** page.
8. Check your email inbox and choose **Confirm subscription** in the email from AWS Notifications. The sender ID is usually "no-reply@sns.amazonaws.com".
  9. Amazon SNS opens your web browser and displays a subscription confirmation with your subscription ID.

## Step 3: Publish a message to the topic

1. In the left navigation pane, choose **Topics**.
2. On the **Topics** page, choose the topic that you created earlier, and then choose **Publish message**.

The console opens the **Publish message to topic** page.

3. (Optional) In the **Message details** section, enter a **Subject**, such as:

Hello from Amazon SNS!

4. In the **Message body** section, choose **Identical payload for all delivery protocols**, and then enter a message body, such as:

Publishing a message to an SNS topic.

5. Choose **Publish message**.

The message is published to the topic, and the console opens the topic's **Details** page.

6. Check your email inbox and verify that you received an email from Amazon SNS with the published message.

## Step 4: Delete the subscription and topic

1. On the navigation panel, choose **Subscriptions**.
2. On the **Subscriptions** page, choose a *confirmed* subscription and then choose **Delete**.

**Note**

You can't delete a pending confirmation. After 3 days, Amazon SNS deletes it automatically.

3. In the **Delete subscription** dialog box, choose **Delete**.

The subscription is deleted.

4. On the navigation panel, choose **Topics**.
5. On the **Topics** page, choose a topic and then choose **Delete**.

**Important**

When you delete a topic, you also delete all subscriptions to the topic.

6. On the **Delete topic** *MyTopic* dialog box, enter `delete me` and then choose **Delete**.

The topic is deleted.

## Next steps

Now that you've created a topic with a subscription and sent messages to the topic, you might want to try the following:

- Explore the [AWS Developer Center](#).
- Learn about protecting your data in the [Security \(p. 343\)](#) section.
- Enable [server-side encryption \(p. 349\)](#) for a topic.
- Enable server-side encryption for a topic with an [encrypted Amazon Simple Queue Service \(Amazon SQS\) queue \(p. 350\)](#) subscribed.
- Subscribe [AWS Event Fork Pipelines \(p. 159\)](#) to a topic.

# Configuring Amazon SNS

Use the [Amazon SNS console](#) to create and configure Amazon SNS topics and subscriptions. For more information about Amazon SNS, see [What is Amazon SNS? \(p. 1\)](#)

## Topics

- [Creating an Amazon SNS topic \(p. 24\)](#)
- [Subscribing to an Amazon SNS topic \(p. 31\)](#)
- [Deleting an Amazon SNS subscription and topic \(p. 31\)](#)
- [Configuring tags for an Amazon SNS topic \(p. 36\)](#)

## Creating an Amazon SNS topic

An Amazon SNS topic is a logical access point that acts as a *communication channel*. A topic lets you group multiple *endpoints* (such as AWS Lambda, Amazon SQS, HTTP/S, or an email address).

To broadcast the messages of a message-producer system (for example, an e-commerce website) working with multiple other services that require its messages (for example, checkout and fulfillment systems), you can create a topic for your producer system.

The first and most common Amazon SNS task is creating a topic. This page shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to create a topic.

During creation, you choose a topic type (standard or FIFO) and name the topic. After creating a topic, you can't change the topic type or name. All other configuration choices are optional during topic creation, and you can edit them later.

### Important

Do not add personally identifiable information (PII) or other confidential or sensitive information in topic names. Topic names are accessible to other Amazon Web Services, including CloudWatch Logs. Topic names are not intended to be used for private or sensitive data.

## Topics

- [To create a topic using the AWS Management Console \(p. 24\)](#)
- [To create a topic using an AWS SDK \(p. 26\)](#)

## To create a topic using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. Do one of the following:
  - If no topics have ever been created under your AWS account before, read the description of Amazon SNS on the home page.

- If topics have been created under your AWS account before, on the navigation panel, choose **Topics**.
3. On the **Topics** page, choose **Create topic**.
  4. On the **Create topic** page, in the **Details** section, do the following:
    - a. For **Type**, choose a topic type (**Standard** or **FIFO**).
    - b. Enter a **Name** for the topic. For a [FIFO topic \(p. 38\)](#), add **.fifo** to the end of the name.
    - c. (Optional) Enter a **Display name** for the topic.
    - d. (Optional) For a FIFO topic, you can choose **content-based message deduplication** to enable default message deduplication. For more information, see [Message deduplication for FIFO topics \(p. 49\)](#).
  5. (Optional) Expand the **Encryption** section and do the following. For more information, see [Encryption at rest \(p. 344\)](#).
    - a. Choose **Enable encryption**.
    - b. Specify the customer master key (CMK). For more information, see [Key terms \(p. 345\)](#).

For each CMK type, the **Description**, **Account**, and **CMK ARN** are displayed.

**Important**

If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms>ListAliases` and `kmsDescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SNS console.

Ask the owner of the CMK to grant you these permissions. For more information, see the [AWS KMS API Permissions: Actions and Resources Reference](#) in the [AWS Key Management Service Developer Guide](#).

- The AWS managed CMK for Amazon SNS (**Default**) alias/`aws/sns` is selected by default.

**Note**

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SNS for a topic, AWS KMS creates the AWS managed CMK for Amazon SNS.
- Alternatively, the first time you use the **Publish** action on a topic with SSE enabled, AWS KMS creates the AWS managed CMK for Amazon SNS.
- To use a custom CMK from your AWS account, choose the **Customer master key (CMK)** field and then choose the custom CMK from the list.

**Note**

For instructions on creating custom CMKs, see [Creating Keys](#) in the [AWS Key Management Service Developer Guide](#)

- To use a custom CMK ARN from your AWS account or from another AWS account, enter it into the **Customer master key (CMK)** field.

6. (Optional) By default, only the topic owner can publish or subscribe to the topic. To configure additional access permissions, expand the **Access policy** section. For more information, see [Identity and access management in Amazon SNS \(p. 364\)](#) and [Example cases for Amazon SNS access control \(p. 374\)](#).

**Note**

When you create a topic using the console, the default policy uses the `aws:SourceOwner` condition key. This key is similar to `aws:SourceAccount`.

7. (Optional) To configure how Amazon SNS retries failed message delivery attempts, expand the **Delivery retry policy (HTTP/S)** section. For more information, see [Amazon SNS message delivery retries \(p. 92\)](#).

8. (Optional) To configure how Amazon SNS logs the delivery of messages to CloudWatch, expand the **Delivery status logging** section. For more information, see [Amazon SNS message delivery status \(p. 87\)](#).
9. (Optional) To add metadata tags to the topic, expand the **Tags** section, enter a **Key** and a **Value** (optional) and choose **Add tag**. For more information, see [Configuring tags for an Amazon SNS topic \(p. 36\)](#).
10. Choose **Create topic**.

The topic is created and the **MyTopic** page is displayed.

The topic's **Name**, **ARN**, (optional) **Display name**, and **Topic owner**'s AWS account ID are displayed in the **Details** section.

11. Copy the topic ARN to the clipboard, for example:

```
arn:aws:sns:us-east-2:123456789012:MyTopic
```

## To create a topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code examples show how to create an Amazon SNS topic.

.NET

### AWS SDK for .NET

```
/// <summary>
/// Creates a new SNS topic using the supplied topic name.
/// </summary>
/// <param name="client">The initialized SNS client object used to
/// create the new topic.</param>
/// <param name="topicName">A string representing the topic name.</param>
/// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>
public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
{
    var request = new CreateTopicRequest
    {
        Name = topicName,
    };

    var response = await client.CreateTopicAsync(request);

    return response.TopicArn;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for .NET API Reference](#).

C++

### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::String topic_name = argv[1];
    Aws::SNS::SNSClient sns;

    Aws::SNS::Model::CreateTopicRequest ct_req;
    ct_req.SetName(topic_name);

    auto ct_out = sns.CreateTopic(ct_req);

    if (ct_out.IsSuccess())
    {
        std::cout << "Successfully created topic " << topic_name << std::endl;
    }
    else
    {
        std::cout << "Error creating topic " << topic_name << ":" <<
            ct_out.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for C++ API Reference](#).

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Go API Reference](#).

## Java

### SDK for Java 2.x

```
public static String createSNSTopic(SnsClient snsClient, String topicName ) {

    CreateTopicResponse result = null;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();
    } catch (SnsException e) {

        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

- Find instructions and more code on [GitHub](#).

- For API details, see [CreateTopic](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {CreateTopicCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { Name: "TOPIC_NAME" }; //TOPIC_NAME

const run = async () => {
    try {
        const data = await snsClient.send(new CreateTopicCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateTopic](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Create a Simple Notification Service topics in your AWS account at the requested
 * region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */
```

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topicname = 'myTopic';

try {
    $result = $SnSclient->createTopic([
        'Name' => $topicname,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [CreateTopic in AWS SDK for PHP API Reference](#).

## Python

### [SDK for Python \(Boto3\)](#)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_topic(self, name):
        """
        Creates a notification topic.

        :param name: The name of the topic to create.
        :return: The newly created topic.
        """
        try:
            topic = self.sns_resource.create_topic(Name=name)
            logger.info("Created topic %s with ARN %s.", name, topic.arn)
        except ClientError:
            logger.exception("Couldn't create topic %s.", name)
            raise
        else:
            return topic
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### [SDK for Ruby](#)

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def topic_created?(sns_client, topic_name)

  sns_client.create_topic(name: topic_name)
  rescue StandardError => e
    puts "Error while creating the topic named '#{topic_name}': #{e.message}"
  end

# Full example call:
def run_me
  topic_name = 'TOPIC_NAME'
  region = 'REGION'

  sns_client = Aws::SNS::Client.new(region: region)

  puts "Creating the topic '#{topic_name}'..."

  if topic_created?(sns_client, topic_name)
    puts 'The topic was created.'
  else
    puts 'The topic was not created. Stopping program.'
    exit 1
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [CreateTopic in AWS SDK for Ruby API Reference](#).

## Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn.as_deref().unwrap_or_default()
    );

    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Rust API reference](#).

## Subscribing to an Amazon SNS topic

To receive messages published to [a topic \(p. 24\)](#), you must *subscribe* an [endpoint \(p. 31\)](#) to the topic. When you subscribe an endpoint to a topic, the endpoint begins to receive messages published to the associated topic.

**Note**

HTTP(S) endpoints, email addresses, and AWS resources in other AWS accounts require confirmation of the subscription before they can receive messages.

### To subscribe an endpoint to an Amazon SNS topic

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:
  - a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a topic.
  - b. For **Protocol**, choose an endpoint type. The available endpoint types are:
    - [HTTP/HTTPS \(p. 134\)](#)
    - [Email/Email-JSON \(p. 260\)](#)
    - [Amazon Kinesis Data Firehose \(p. 103\)](#)
    - [Amazon SQS \(p. 124\)](#)

**Note**

To subscribe to an [SNS FIFO topic \(p. 38\)](#), choose this option.

- [AWS Lambda \(p. 123\)](#)
  - [Platform application endpoint \(p. 234\)](#)
  - [SMS \(p. 166\)](#)
- c. For **Endpoint**, enter the endpoint value, such as an email address or the ARN of an Amazon SQS queue.
  - d. Kinesis Data Firehose endpoints only: For **Subscription role ARN**, specify the ARN of the IAM role that you created for writing to Kinesis Data Firehose delivery streams. For more information, see [Prerequisites for subscribing Kinesis Data Firehose delivery streams to Amazon SNS topics \(p. 103\)](#).
  - e. (Optional) For Kinesis Data Firehose, Amazon SQS, HTTP/S endpoints, you can also enable raw message delivery. For more information, see [Amazon SNS raw message delivery \(p. 83\)](#).
  - f. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see [Amazon SNS subscription filter policies \(p. 71\)](#).
  - g. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#).
  - h. Choose **Create subscription**.

The console creates the subscription and opens the subscription's **Details** page.

## Deleting an Amazon SNS subscription and topic

You can delete a subscription from an Amazon SNS topic, or you can delete the whole topic. Note that you can't delete a subscription that's pending confirmation. After three days, Amazon SNS deletes the unconfirmed subscription automatically.

## Topics

- [To delete an Amazon SNS subscription and topic using the AWS Management Console \(p. 32\)](#)
- [To delete a subscription and topic using an AWS SDK \(p. 32\)](#)

# To delete an Amazon SNS subscription and topic using the AWS Management Console

## To delete a subscription using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, select a subscription with a **Status of Confirmed**, and then choose **Delete**.
4. In the **Delete subscription** dialog box, choose **Delete**.

The console deletes the subscription.

When you delete a topic, Amazon SNS deletes the subscriptions associated with the topic.

## To delete a topic using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic, and then choose **Delete**.
4. In the **Delete topic** dialog box, enter `delete me`, and then choose **Delete**.

The console deletes the topic.

# To delete a subscription and topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code examples show how to delete an Amazon SNS topic and all subscriptions to that topic.

.NET

### AWS SDK for .NET

```
/// <summary>
/// This example deletes an existing Amazon Simple Notification Service
/// (Amazon SNS) topic. The example was created using the AWS SDK for .NET
/// version 3.7 and .NET Core 5.0.
/// </summary>
public class DeleteSNSTopic
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-east-2:704825161248:ExampleSNSTopic";
```

```
IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

    var response = await client.DeleteTopicAsync(topicArn);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for .NET API Reference*.

C++

#### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::String topic_arn = argv[1];
    Aws::SNS::SNSClient sns;

    Aws::SNS::Model::DeleteTopicRequest dt_req;
    dt_req.SetTopicArn(topic_arn);

    auto dt_out = sns.DeleteTopic(dt_req);

    if (dt_out.IsSuccess())
    {
        std::cout << "Successfully deleted topic " << topic_arn << std::endl;
    }
    else
    {
        std::cout << "Error deleting topic " << topic_arn << ":" <<
            dt_out.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for C++ API Reference*.

Java

#### SDK for Java 2.x

```
public static void deleteSNSTopic(SnsClient snsClient, String topicArn ) {

    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
            result.sdkHttpResponse().statusCode());
    }
}
```

```
        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js

// Import required AWS SDK clients and commands for Node.js
import {DeleteTopicCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { TopicArn: "TOPIC_ARN" }; //TOPIC_ARN

const run = async () => {
    try {
        const data = await snsClient.send(new DeleteTopicCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';
```

```
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Deletes a SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for Python (Boto3) API Reference*.

# Configuring tags for an Amazon SNS topic

You can track your Amazon SNS resources (for example, for cost allocation) by adding, removing, and listing metadata tags for Amazon SNS topics. This page shows how to add, update, and remove tags for a topic using the AWS Management Console and the AWS SDK for Java.

## Important

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to other Amazon Web Services, including billing. Tags are not intended to be used for private or sensitive data.

## Topics

- [To list, add, and remove, metadata tags for an Amazon SNS topic using the AWS Management Console \(p. 36\)](#)
- [To add metadata tags to a topic using an AWS SDK \(p. 36\)](#)

## Note

Currently, tag-based access control isn't available.

## To list, add, and remove, metadata tags for an Amazon SNS topic using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic and then choose **Edit**.
4. Expand the **Tags** section.  
The tags added to the topic are listed.
5. Modify topic tags:
  - To add a tag, choose **Add tag** and enter a **Key** and **Value** (optional),
  - To remove a tag, choose **Remove tag** next to a key-value pair.
6. Choose **Save changes**

## To add metadata tags to a topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code example shows how to add tags to an Amazon SNS topic.

Java

### SDK for Java 2.x

```
public static void addTopicTags(SnsClient snsClient, String topicArn) {  
    try {  
        Tag tag = Tag.builder()  
    }  
}
```

```
.key("Team")
.value("Development")
.build();

Tag tag2 = Tag.builder()
.key("Environment")
.value("Gamma")
.build();

List<Tag> tagList = new ArrayList<>();
tagList.add(tag);
tagList.add(tag2);

TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
.resourceArn(topicArn)
.tags(tagList)
.build();

snsClient.tagResource(tagResourceRequest);
System.out.println("Tags have been added to "+topicArn);

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [TagResource](#) in *AWS SDK for Java 2.x API Reference*.

# Message ordering and deduplication (FIFO topics)

You can use Amazon SNS FIFO (first in, first out) topics and [Amazon Simple Queue Service \(Amazon SQS\) FIFO queues](#) together to provide strict message ordering and message deduplication. The FIFO capabilities of each of these services work together to act as a fully managed service to integrate distributed applications that require data consistency in near-real time.

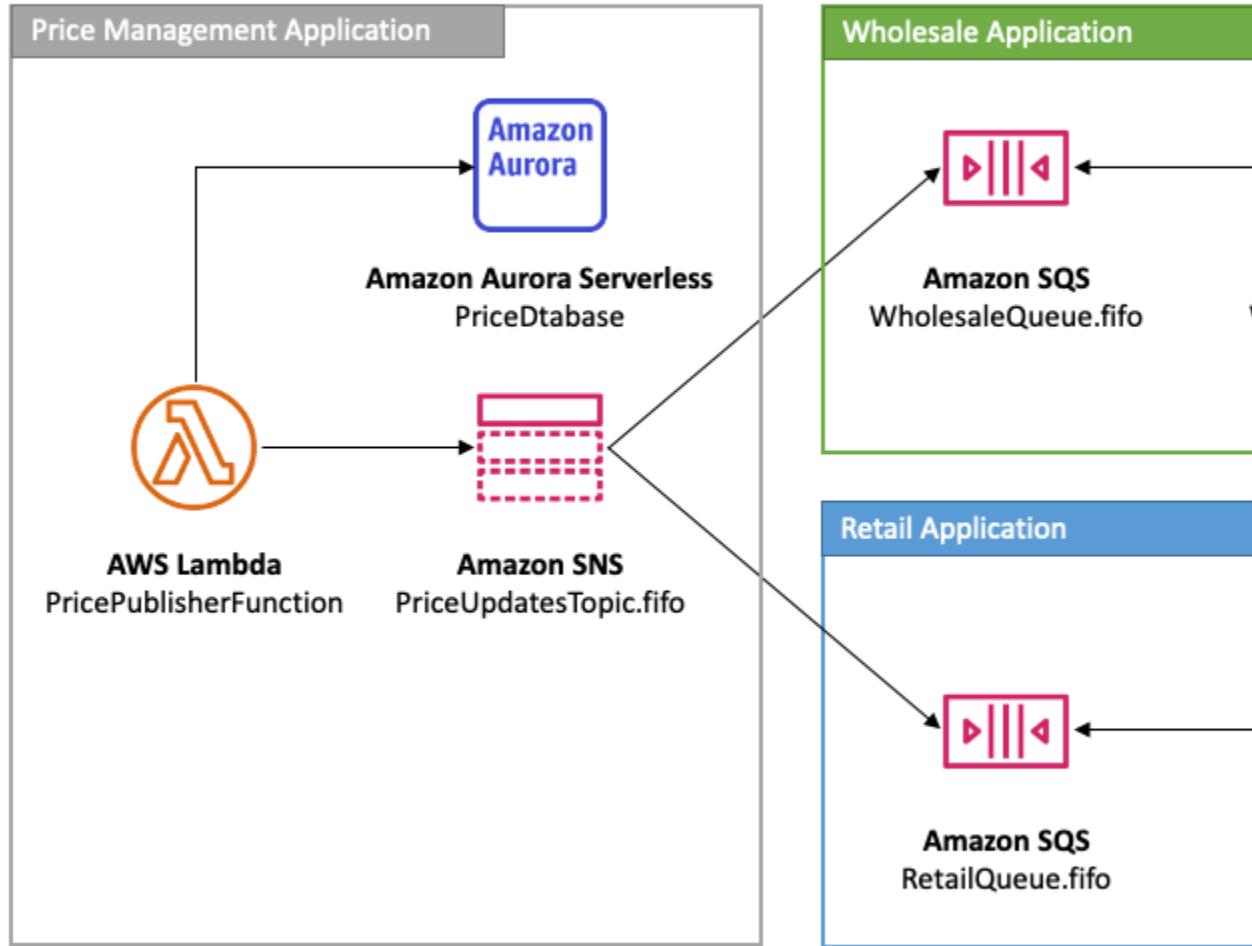
## Topics

- [FIFO topics example use case \(p. 38\)](#)
- [Message ordering details for FIFO topics \(p. 40\)](#)
- [Message grouping for FIFO topics \(p. 45\)](#)
- [Message delivery for FIFO topics \(p. 46\)](#)
- [Message filtering for FIFO topics \(p. 47\)](#)
- [Message deduplication for FIFO topics \(p. 49\)](#)
- [Message security for FIFO topics \(p. 51\)](#)
- [Message durability for FIFO topics \(p. 51\)](#)
- [Code examples for FIFO topics \(p. 53\)](#)

## FIFO topics example use case

The following example describes an ecommerce platform built by an auto parts manufacturer using Amazon SNS FIFO topics and Amazon SQS FIFO queues. The platform is composed of three serverless applications:

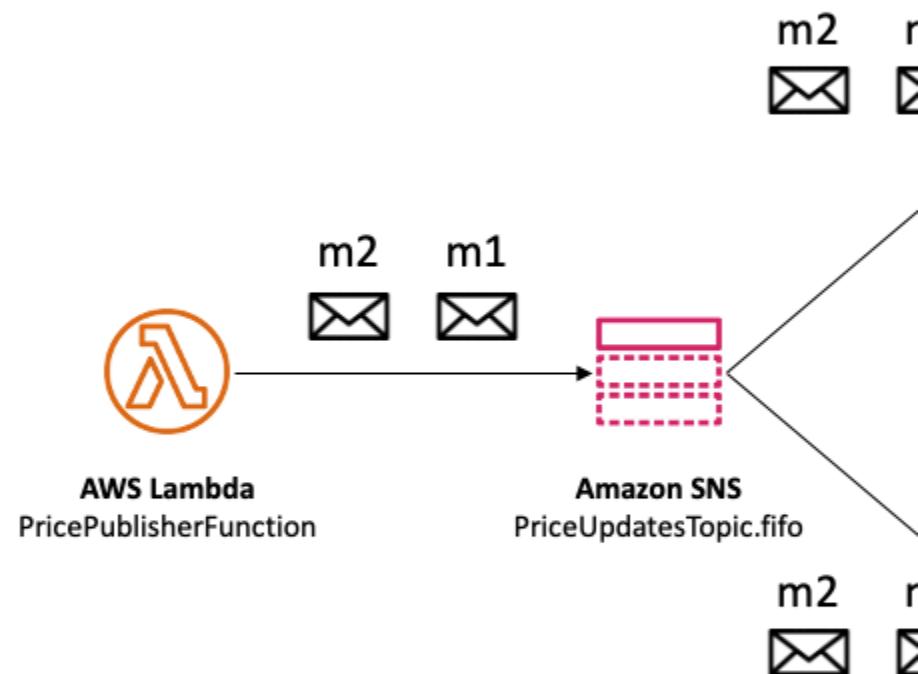
- Inventory managers use a price management application to set the price for each item in stock. At this company, product prices can change based on currency exchange fluctuation, market demand, and shifts in sales strategy. The price management application uses an AWS Lambda function that publishes price updates to an SNS FIFO topic whenever prices change.
- A wholesale application provides the backend for a website where auto body shops and car manufacturers can buy the company's auto parts in bulk. To get price change notifications, the wholesale application subscribes its SQS FIFO queue to the price management application's SNS FIFO topic.
- A retail application provides the backend for another website where car owners and car tuning enthusiasts can purchase individual auto parts for their vehicles. To get price change notifications, the retail application also subscribes its SQS FIFO queue to the price management application's SNS FIFO topic.



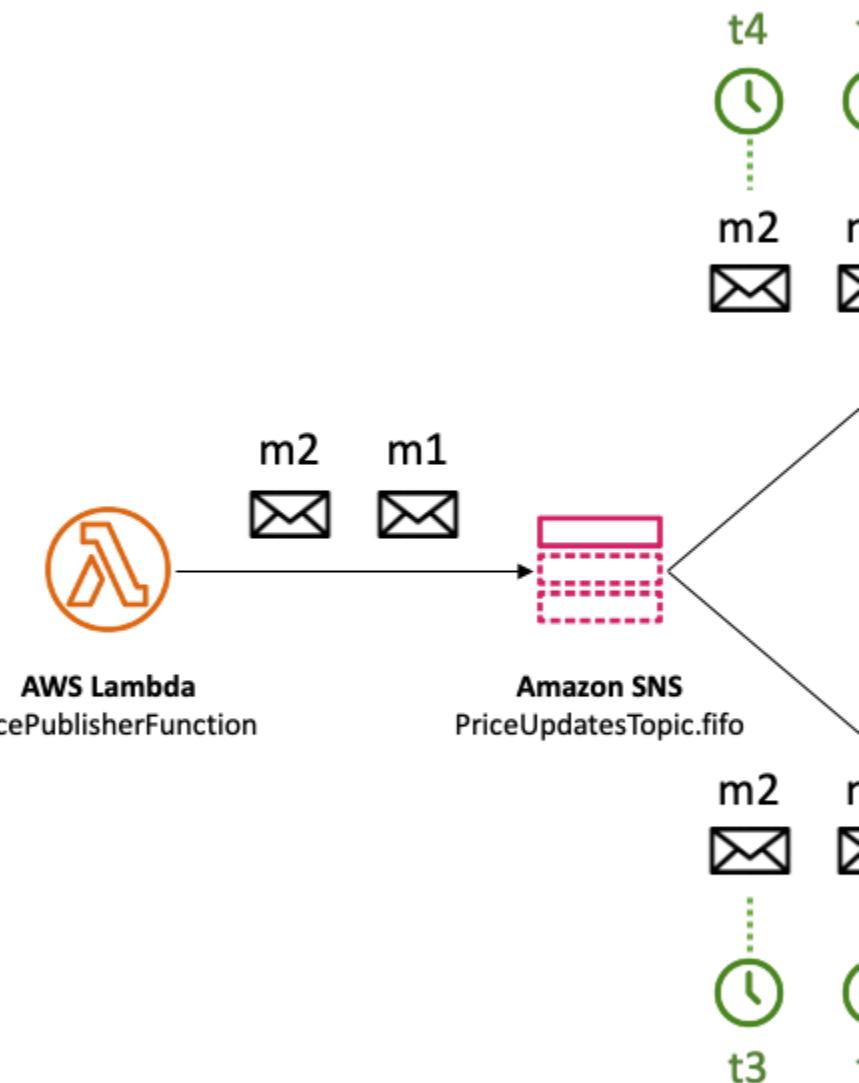
For the wholesale and retail applications to receive price updates in the correct order, the price management application must use a strictly ordered message distribution system. Using SNS FIFO topics and SQS FIFO queues enables the processing of messages in order and with no duplication. For more information, see [Message ordering details for FIFO topics \(p. 40\)](#). For code snippets that implement this use case, see [Code examples for FIFO topics \(p. 53\)](#).

## Message ordering details for FIFO topics

An Amazon SNS FIFO topic delivers messages to subscribed Amazon SQS FIFO queues in the exact order that the messages are published to the topic. With an SQS FIFO queue, the queues' consumers receive the messages in the exact order that the messages are sent to the queue. This setup preserves end-to-end message ordering, as shown in the following example based on the [FIFO topics example use case \(p. 38\)](#).

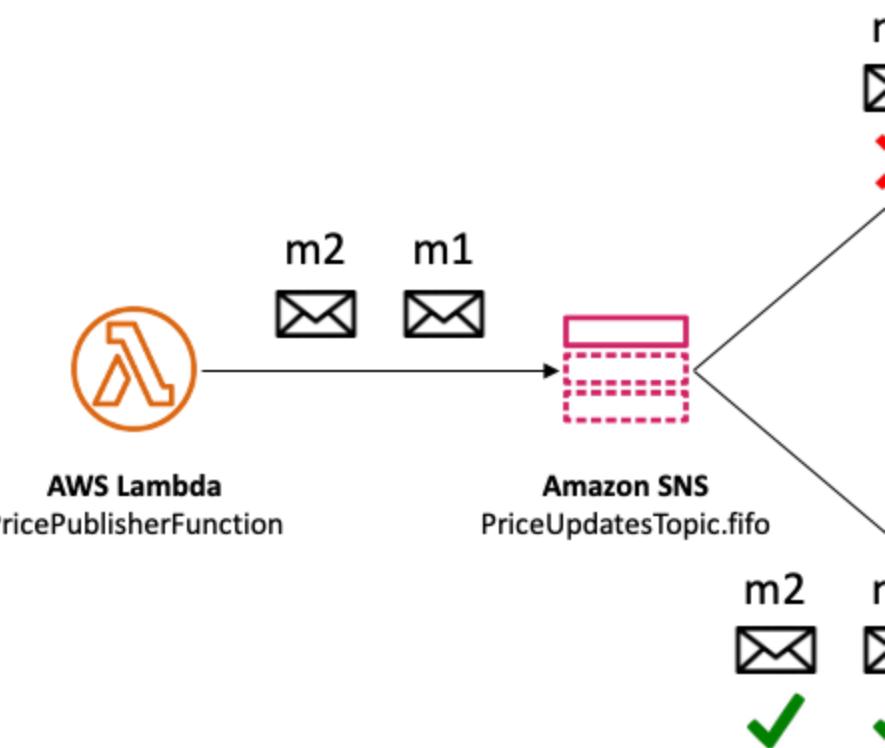


Note that there is no implied ordering of the subscribers. The following example shows that message **m1** is delivered first to the wholesale subscriber and then to the retail subscriber. Message **m2** is delivered first to the retail subscriber and then to the wholesale subscriber. Though the two messages are delivered to the subscribers in a different order, message ordering is preserved for each subscriber. Each subscriber is perceived in isolation from any other subscribers.



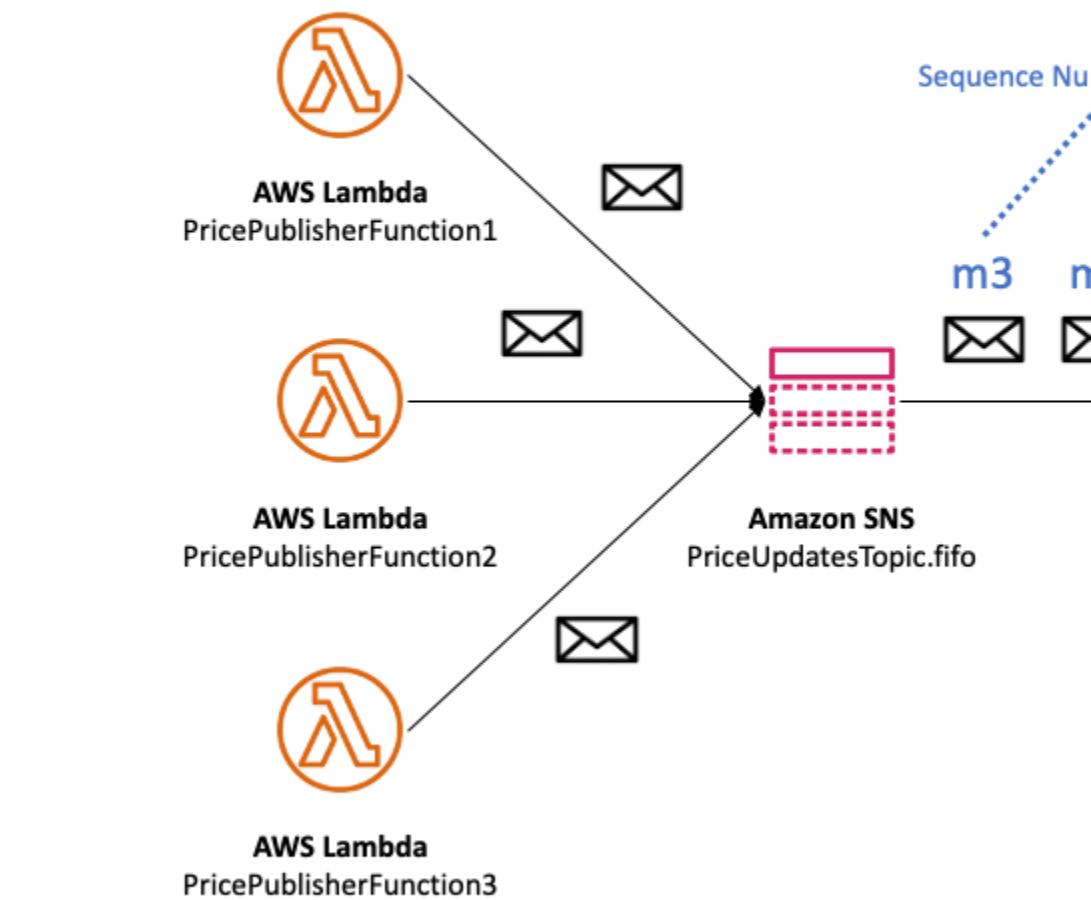
If an SQS FIFO queue subscriber becomes unreachable, it can get out of sync. For example, say the wholesale application queue owner mistakenly changes the [Amazon SQS queue policy](#) in a way

that prevents the Amazon SNS service principal from delivering messages to the queue. In this case, wholesale price updates aren't delivered, but retail price updates succeed, causing the subscribers to be out of sync. When the wholesale application queue owner corrects the queue policy, Amazon SNS resumes delivering messages to the subscribed queue. Any messages that were published to the topic while the queue was incorrectly configured are dropped, unless the subscription has a [dead-letter queue \(p. 96\)](#) configured.



You can have multiple applications (or multiple threads within the same application) publishing messages to an SNS FIFO topic in parallel. When you do this, you effectively delegate message sequencing to the Amazon SNS service. To determine the established sequence of messages, you can check the sequence number.

The sequence number is a large, non-consecutive, ever-increasing number that Amazon SNS assigns to each message that you publish. The sequence number is passed to the subscribed SQS FIFO queues as part of the message body. However, if you enable [raw message delivery \(p. 83\)](#), the message that's delivered to the SQS FIFO queue doesn't include the sequence number or any other SNS message metadata.



Amazon SNS FIFO topics define ordering in the context of a message group. For more information, see [Message grouping for FIFO topics \(p. 45\)](#).

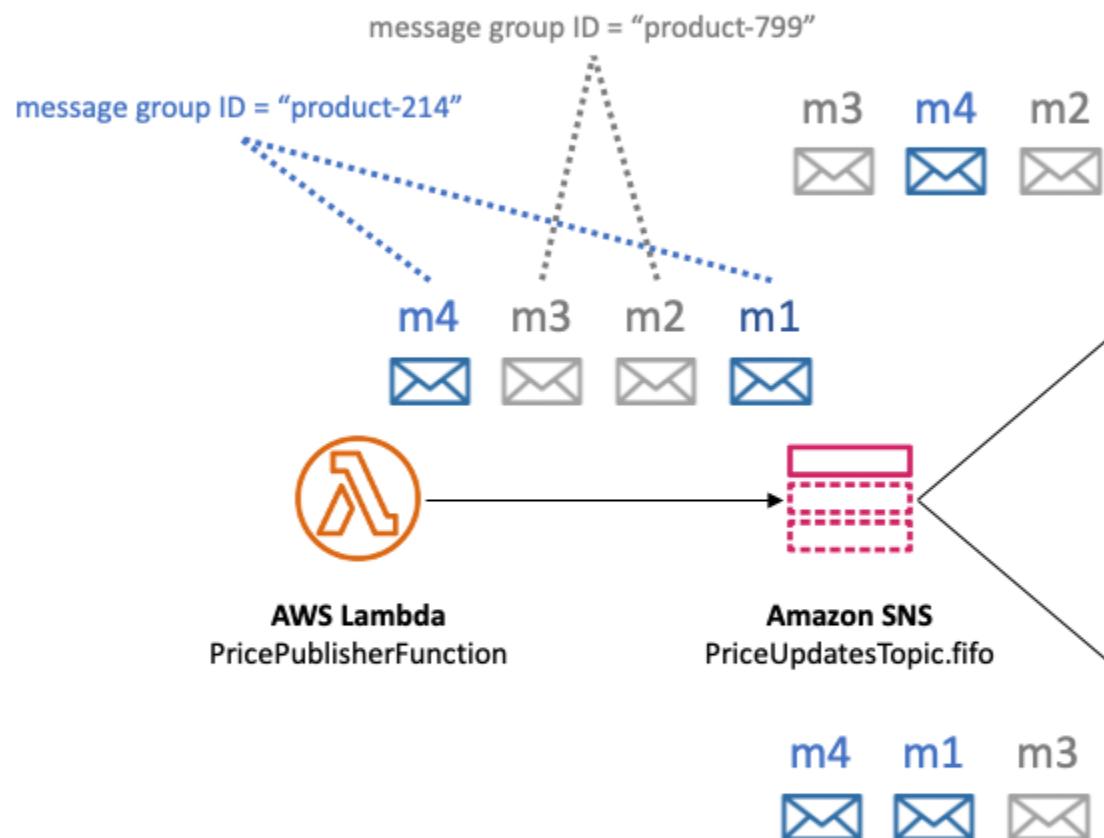
## Message grouping for FIFO topics

Messages that belong to the same group are processed one by one, in a strict order relative to the group.

When you publish messages to an Amazon SNS FIFO topic, you set the message group ID. The group ID is a mandatory token that specifies that a message belongs to a specific message group. The SNS FIFO topic passes the group ID to the subscribed Amazon SQS FIFO queues. There is no limit to the number of group IDs in SNS FIFO topics or SQS FIFO queues.

There's no affinity between a message group and a subscription. Therefore, messages that are published to any message group are delivered to all subscribed queues, subject to any filter policies attached to subscriptions. For more information, see [Message delivery for FIFO topics \(p. 46\)](#) and [Message filtering for FIFO topics \(p. 47\)](#).

In the [auto parts price management example use case \(p. 38\)](#), there's a dedicated message group for each product sold in the platform. The same SNS FIFO topic is used for processing all price updates. The sequence of price updates is preserved within the context of a single auto parts product, but *not* across multiple products. The following diagram shows how this works. Notice that for the product with the **product-214** message group ID, the **m1** message is always processed before the **m4** message. This sequence is preserved throughout the workflow, from Amazon SNS to Amazon SQS to AWS Lambda. Similarly, for the product with the **product-799** message group ID, the **m2** message is always processed before the **m3** message. The **product-214** and **product-799** message groups are independent of each other, so there is no relationship between how their messages are sequenced.



## Message delivery for FIFO topics

To preserve strict message ordering, Amazon SNS restricts the set of supported delivery protocols for Amazon SNS FIFO topics. Currently, the endpoint protocol must be Amazon SQS, with an Amazon SQS FIFO queue's Amazon Resource Name (ARN) as the endpoint.

**Note**

To fan out messages from Amazon SNS FIFO topics to AWS Lambda functions, extra steps are required. First, subscribe Amazon SQS FIFO queues to the topic. Then configure the queues to trigger the functions. For more information, see the [SQS FIFO as an event source](#) post on the [AWS Compute Blog](#).

SNS FIFO topics can't deliver messages to customer managed endpoints, such as email addresses, mobile apps, phone numbers for text messaging (SMS), or HTTP(S) endpoints. These endpoint types aren't guaranteed to preserve strict message ordering. Attempts to subscribe customer managed endpoints to SNS FIFO topics result in errors.

SNS FIFO topics support the same message filtering capabilities as standard topics. For more information, see [Message filtering for FIFO topics \(p. 47\)](#) and the [Simplify Your Pub/Sub Messaging with Amazon SNS Message Filtering](#) post on the [AWS Compute Blog](#).

## Message filtering for FIFO topics

Amazon SNS FIFO topics support message filtering. Using message filtering simplifies your architecture by offloading the message routing logic from your publisher systems and the message filtering logic from your subscriber systems.

When you subscribe an Amazon SQS FIFO queue to an SNS FIFO topic, you can use message filtering to specify that the subscriber receives a subset of messages, rather than all of them. Each subscriber can set its own filter policy as a subscription attribute. If the filter policy matches the incoming message's attributes, the topic delivers a copy of the message to the subscriber. If there's no match, the topic doesn't deliver a copy of the message.

In the [auto parts price management example use case \(p. 38\)](#), assume that the following Amazon SNS filter policies are set:

- For the wholesale queue, the filter policy `{"business": ["wholesale"]}` matches every message with an attribute named "business" and with "wholesale" in the set of values. In the following diagram, the attribute in message **m1** is `String` with a value of "wholesale". The attribute in message **m3** is `String.Array` with a value of "wholesale, retail". Thus, both **m1** and **m3** match the filter policy's criteria, and both messages are delivered to the wholesale queue.
- For the retail queue, the filter policy `{"business": ["retail"]}` matches every message with an attribute named "business" and with "retail" in the set of values. In the diagram, the attribute in message **m2** is `String` with a value of "retail". The attribute in message **m3** is `String.Array` with a value of "wholesale, retail". Thus, both **m2** and **m3** match the filter policy's criteria, and both messages are delivered to the retail queue.

The following diagram shows the effect of messaging filtering using these filter policies.



SNS FIFO topics support a variety of matching operators, including attribute string values, attribute numeric values, and attribute keys. For more information, see [Amazon SNS message filtering \(p. 71\)](#).

SNS FIFO topics don't deliver duplicate messages to subscribed endpoints. For more information, see [Message deduplication for FIFO topics \(p. 49\)](#).

# Message deduplication for FIFO topics

Amazon SNS FIFO topics and Amazon SQS FIFO queues support message deduplication, which provides exactly-once message delivery and processing as long as the following conditions are met:

- The subscribed SQS FIFO queue exists and has permissions that allow the Amazon SNS service principal to deliver messages to the queue.
- The SQS FIFO queue consumer processes the message and deletes it from the queue before the visibility timeout expires.
- The Amazon SNS subscription topic has no [message filtering \(p. 47\)](#). When you configure message filtering, SNS FIFO topics support at-most-once delivery, as messages can be filtered out based on your subscription filter policies.
- There are no network disruptions that prevent acknowledgment of the message delivery.

## Note

Message deduplication applies to an entire SNS FIFO topic, not to an individual [message group \(p. 45\)](#).

When you publish a message to an SNS FIFO topic, the message must include a deduplication ID. This ID is included in the message that the SNS FIFO topic delivers to the subscribed SQS FIFO queues.

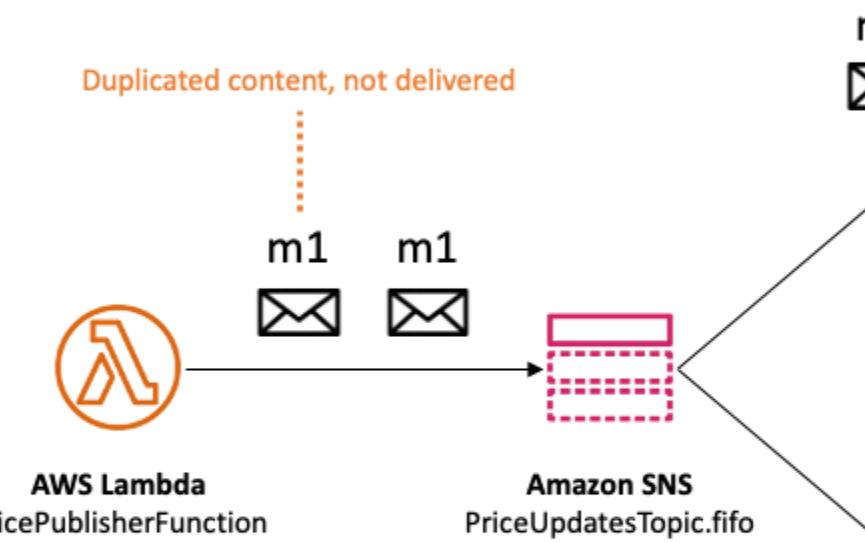
If a message with a particular deduplication ID is successfully published to an SNS FIFO topic, any message published with the same deduplication ID, within the five-minute deduplication interval, is accepted but not delivered. The SNS FIFO topic continues to track the message deduplication ID, even after the message is delivered to subscribed endpoints.

If the message body is guaranteed to be unique for each published message, you can enable content-based deduplication for an Amazon SNS FIFO topic and the subscribed SQS FIFO queues. Amazon SNS uses the message body to generate a unique hash value to use as the deduplication ID for each message, so you don't need to set a deduplication ID when you send each message.

## Note

Message attributes are not included in the hash calculation.

In the [auto parts price management example use case \(p. 38\)](#), the company must set a universally unique deduplication ID for each price update. This is because the message body can be identical even when the message attribute is different for wholesale and retail. However, if the company added the business type (wholesale or retail) to the message body alongside the product ID and product price, they could enable content-based duplication in the SNS FIFO topic and the subscribed SQS FIFO queues.



In addition to message ordering and deduplication, SNS FIFO topics support message server-side encryption (SSE) with AWS KMS keys, and message privacy via VPC endpoints with AWS PrivateLink. For more information, see [Message security for FIFO topics \(p. 51\)](#).

## Message security for FIFO topics

You can choose to have Amazon SNS and Amazon SQS encrypt messages sent to FIFO topics and queues, using [AWS Key Management Service \(AWS KMS\) customer master keys \(CMKs\)](#). You can create encrypted FIFO topics and queues, or choose to encrypt existing FIFO topics and queues. Amazon SNS and Amazon SQS encrypt only the body of the message. They don't encrypt the message attributes, resource metadata, or resource metrics.

**Note**

Adding encryption to an existing FIFO topic or queue doesn't encrypt any backlogged messages, and removing encryption from a topic or queue leaves backlogged messages encrypted.

SNS FIFO topics decrypt the messages immediately before delivering them to subscribed endpoints. SQS FIFO queues decrypt the message just before returning them to the consumer application. For more information, see [Data encryption \(p. 344\)](#) and the [Encrypting messages published to Amazon SNS with AWS KMS](#) post on the [AWS Compute Blog](#).

In addition, SNS FIFO topics and SQS FIFO queues support message privacy with [Interface VPC endpoints](#) powered by AWS PrivateLink. Using interface endpoints, you can send messages from Amazon Virtual Private Cloud (Amazon VPC) subnets to FIFO topics and queues without traversing the public internet. This model keeps your messaging within the AWS infrastructure and network, which enhances the overall security of your application. When you use AWS PrivateLink, you don't need to set up an internet gateway, network address translation (NAT), or virtual private network (VPN). For more information, see [Internetwork traffic privacy \(p. 352\)](#) and the [Securing messages published to Amazon SNS with AWS PrivateLink](#) post on the [AWS Security Blog](#).

SNS FIFO topics also support dead-letter queues and message storage across Availability Zones. For more information, see [Message durability for FIFO topics \(p. 51\)](#).

## Message durability for FIFO topics

Amazon SNS FIFO topics and Amazon SQS FIFO queues are durable. Both resource types store messages redundantly across multiple Availability Zones, and provide dead-letter queues to handle exceptional cases.

In Amazon SNS, message delivery fails when the Amazon SNS topic can't access a subscribed Amazon SQS queue due to a client-side or server-side error:

- Client-side errors occur when the SNS FIFO topic has stale subscription metadata. Two common causes of client-side errors are when the SQS FIFO queue owner does one of the following:
  - Deletes the queue.
  - Changes the queue policy in a way that prevents the Amazon SNS service principal from delivering messages to it.

Amazon SNS doesn't retry delivering messages that failed due to client-side errors.

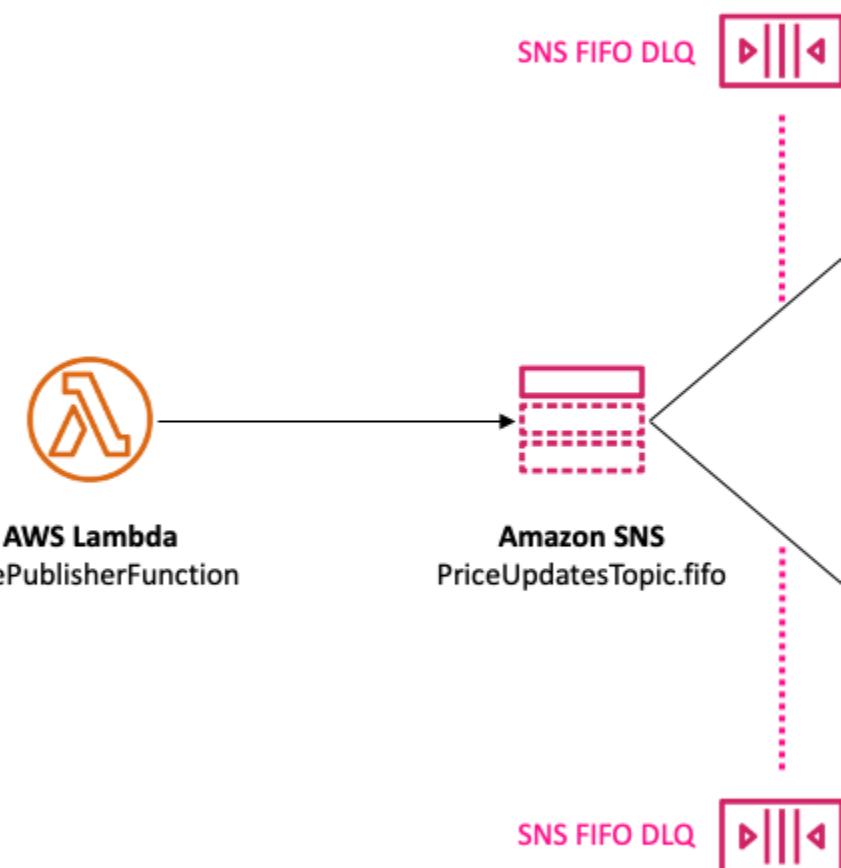
- Server-side errors can occur in these situations:
  - The Amazon SQS service is unavailable.
  - Amazon SQS fails to process a valid request from the Amazon SNS service.

When server-side errors occur, SNS FIFO topics retry the failed deliveries up to 100,015 times over 23 days. For more information, see [Amazon SNS message delivery retries \(p. 92\)](#).

For any type of error, Amazon SNS can sideline messages to Amazon SQS dead-letter queues so data isn't lost.

In Amazon SQS, message processing fails when the consumer application fails to receive the message, process it, and delete it from the queue. When the maximum number of receive requests fail, Amazon SQS can sideline messages to dead-letter queues so data isn't lost.

In the [auto parts price management example use case \(p. 38\)](#), the company can assign an SQS FIFO dead-letter queue (DLQ) to each SNS FIFO topic subscription, as well as to each subscribed SQS FIFO queue. This protects the company from any price update loss.



The dead-letter queue associated with an SNS FIFO subscription, or with an SQS FIFO queue, must be an SQS FIFO queue. The dead-letter queue must be in the same AWS Region and AWS account as the SNS FIFO subscription or SQS FIFO queue that it protects. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#) and the [Designing durable serverless apps with DLQs for Amazon SNS, Amazon SQS, AWS Lambda](#) post on the [AWS Compute Blog](#).

## Code examples for FIFO topics

You can use the following code examples to integrate the [auto parts price management example use case \(p. 38\)](#) with SNS FIFO topics and SQS FIFO queues.

### Topics

- [Using an AWS SDK \(p. 53\)](#)
- [Using AWS CloudFormation \(p. 55\)](#)

## Using an AWS SDK

Using an AWS SDK, you create an Amazon SNS FIFO topic by setting its `FifoTopic` attribute to true. You create an Amazon SQS FIFO queue by setting its `FifoQueue` attribute to true. Also, you must add the `.fifo` suffix to the name of each FIFO resource. After you create a FIFO topic or queue, you can't convert it into a standard topic or queue.

The following code example creates these FIFO resources:

- The SNS FIFO topic that distributes the price updates
- The SQS FIFO queues that provide these updates to the two applications (wholesale and retail)
- The SNS FIFO subscriptions that connect both of the queues to the topic

This example sets [filter policies \(p. 71\)](#) on the subscriptions. If you test the example by publishing a message to the topic, make sure that you publish the message with the `business` attribute. Specify either `retail` or `wholesale` for the attribute value. Otherwise, the message is filtered out and not delivered to the subscribed queues. For more information, see [Message filtering for FIFO topics \(p. 47\)](#).

Java

### SDK for Java 1.x

Create a FIFO topic and FIFO queues. Subscribe the queues to the topic.

```
// Create API clients
AWSCredentialsProvider credentials = getCredentials();

AmazonSNS sns = new AmazonSNSClient(credentials);
AmazonSQS sqs = new AmazonSQSClient(credentials);

// Create FIFO topic
Map<String, String> topicAttributes = new HashMap<String, String>();
topicAttributes.put("FifoTopic", "true");
topicAttributes.put("ContentBasedDeduplication", "false");

String topicArn = sns.createTopic(
```

```

        new CreateTopicRequest()
            .withName("PriceUpdatesTopic.fifo")
            .withAttributes(topicAttributes)
    ).getTopicArn();

    // Create FIFO queues

    Map<String, String> queueAttributes = new HashMap<String, String>();
    queueAttributes.put("FifoQueue", "true");

    // Disable content-based deduplication because messages published with the same
    // body
    // might carry different attributes that must be processed independently.
    // The price management system uses the message attributes to define whether a
    // given
    // price update applies to the wholesale application or to the retail application.
    queueAttributes.put("ContentBasedDeduplication", "false");

    String wholesaleQueueUrl = sqs.createQueue(
        new CreateQueueRequest()
            .withName("WholesaleQueue.fifo")
            .withAttributes(queueAttributes)
    ).getQueueUrl();

    String retailQueueUrl = sqs.createQueue(
        new CreateQueueRequest()
            .withName("RetailQueue.fifo")
            .withAttributes(queueAttributes)
    ).getQueueUrl();

    // Subscribe FIFO queues to FIFO topic, setting required permissions

    String wholesaleSubscriptionArn =
        Topics.subscribeQueue(sns, sqs, topicArn, wholesaleQueueUrl);

    String retailSubscriptionArn =
        Topics.subscribeQueue(sns, sqs, topicArn, retailQueueUrl);

```

Set a filter policy on each subscription so that each subscriber application receives only the price updates that it needs.

```

// Set the Amazon SNS subscription filter policies

SNSMessageFilterPolicy wholesalePolicy = new SNSMessageFilterPolicy();
wholesalePolicy.addAttribute("business", "wholesale");
wholesalePolicy.apply(sns, wholesaleSubscriptionArn);

SNSMessageFilterPolicy retailPolicy = new SNSMessageFilterPolicy();
retailPolicy.addAttribute("business", "retail");
retailPolicy.apply(sns, retailSubscriptionArn);

```

Compose and publish a message that updates the wholesale price.

```

// Publish message to FIFO topic

String subject = "Price Update";
String payload = "{\"product\": 214, \"price\": 79.99}";
String groupId = "PID-214";
String dedupId = UUID.randomUUID().toString();
String attributeName = "business";

```

```

String attributeValue = "wholesale";

Map<String, MessageAttributeValue> attributes = new HashMap<>();

attributes.put(
    attributeName,
    new MessageAttributeValue()
        .withDataType("String")
        .withStringValue(attributeValue));

sns.publish(
    new PublishRequest()
        .withTopicArn(topicArn)
        .withSubject(subject)
        .withMessage(payload)
        .withMessageGroupId(groupId);
        .withMessageDuplicationId(dedupId)
        .withMessageAttributes(attributes));

```

- Find instructions and more code on [GitHub](#).

## Receiving messages from FIFO subscriptions

You can now receive price updates in the wholesale and retail applications. As shown in the [the section called “FIFO topics use case” \(p. 38\)](#), the point of entry for each consumer application is the SQS FIFO queue, which its corresponding AWS Lambda function can poll automatically. When an SQS FIFO queue is an event source for a Lambda function, Lambda scales its fleet of pollers as needed to efficiently consume messages.

For more information, see [Using AWS Lambda with Amazon SQS](#) in the *AWS Lambda Developer Guide*. For information on writing your own queue pollers, see [Recommendations for Amazon SQS standard and FIFO queues](#) in the *Amazon Simple Queue Service Developer Guide* and [ReceiveMessage](#) in the *Amazon Simple Queue Service API Reference*.

## Using AWS CloudFormation

AWS CloudFormation enables you to use a template file to create and configure a collection of AWS resources together as a single unit. This section has an example template that creates the following:

- The SNS FIFO topic that distributes the price updates
- The SQS FIFO queues that provide these updates to the two applications (wholesale and retail)
- The SNS FIFO subscriptions that connect both of the queues to the topic
- A [filter policy \(p. 71\)](#) that specifies that subscriber applications receive only the price updates that they need

### Note

If you test this code sample by publishing a message to the topic, make sure that you publish the message with the `business` attribute. Specify either `retail` or `wholesale` for the attribute value. Otherwise, the message is filtered out and not delivered to the subscribed queues.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "PriceUpdatesTopic": {
            "Type": "AWS::SNS::Topic",

```

```

    "Properties": {
        "TopicName": "PriceUpdatesTopic fifo",
        "FifoTopic": true,
        "ContentBasedDeduplication": false
    },
    "WholesaleQueue": {
        "Type": "AWS::SQS::Queue",
        "Properties": {
            "QueueName": "WholesaleQueue fifo",
            "FifoQueue": true,
            "ContentBasedDeduplication": false
        }
    },
    "RetailQueue": {
        "Type": "AWS::SQS::Queue",
        "Properties": {
            "QueueName": "RetailQueue fifo",
            "FifoQueue": true,
            "ContentBasedDeduplication": false
        }
    },
    "WholesaleSubscription": {
        "Type": "AWS::SNS::Subscription",
        "Properties": {
            "TopicArn": {
                "Ref": "PriceUpdatesTopic"
            },
            "Endpoint": {
                "Fn::GetAtt": [
                    "WholesaleQueue",
                    "Arn"
                ]
            },
            "Protocol": "sq",
            "RawMessageDelivery": "false",
            "FilterPolicy": {
                "business": [
                    "wholesale"
                ]
            }
        }
    },
    "RetailSubscription": {
        "Type": "AWS::SNS::Subscription",
        "Properties": {
            "TopicArn": {
                "Ref": "PriceUpdatesTopic"
            },
            "Endpoint": {
                "Fn::GetAtt": [
                    "RetailQueue",
                    "Arn"
                ]
            },
            "Protocol": "sq",
            "RawMessageDelivery": "false",
            "FilterPolicy": {
                "business": [
                    "retail"
                ]
            }
        }
    },
    "SalesQueuesPolicy": {
        "Type": "AWS::SQS::QueuePolicy",

```

```
"Properties": {
    "PolicyDocument": {
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sns.amazonaws.com"
                },
                "Action": [
                    "sns:SendMessage"
                ],
                "Resource": "*",
                "Condition": {
                    "ArnEquals": {
                        "aws:SourceArn": {
                            "Ref": "PriceUpdatesTopic"
                        }
                    }
                }
            }
        ]
    },
    "Queues": [
        {
            "Ref": "WholesaleQueue"
        },
        {
            "Ref": "RetailQueue"
        }
    ]
}
```

For more information about deploying AWS resources using an AWS CloudFormation template, see [Get Started](#) in the *AWS CloudFormation User Guide*.

# Amazon SNS message publishing

After you [create an Amazon SNS topic \(p. 24\)](#) and [subscribe \(p. 31\)](#) an endpoint to it, you can *publish* messages to the topic. When a message is published, Amazon SNS attempts to deliver the message to the subscribed [endpoints \(p. 31\)](#).

**Important**

You can publish messages only to topics and endpoints in the same AWS Region.

**Topics**

- [To publish messages to Amazon SNS topics using the AWS Management Console \(p. 58\)](#)
- [To publish a message to a topic using an AWS SDK \(p. 59\)](#)
- [Publishing large messages with Amazon SNS and Amazon S3 \(p. 65\)](#)
- [Amazon SNS message attributes \(p. 68\)](#)

## To publish messages to Amazon SNS topics using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic, and then choose **Publish message**.  
The console opens the **Publish message to topic** page.
4. In the **Message details** section, do the following:
  - a. (Optional) Enter a message **Subject**.
  - b. For a [FIFO topic \(p. 38\)](#), enter a **Message group ID**. Messages in the same message group are delivered in the order that they are published.
  - c. For a FIFO topic, enter a **Message deduplication ID**. This ID is optional if you enabled the **Content-based message deduplication** setting for the topic.
  - d. (Optional) For [mobile push notifications \(p. 258\)](#), enter a **Time to Live (TTL)** value in seconds. This is the amount of time that a push notification service—such as Apple Push Notification Service (APNs) or Firebase Cloud Messaging (FCM)—has to deliver the message to the endpoint.
5. In the **Message body** section, do one of the following:
  - a. Choose **Identical payload for all delivery protocols**, and then enter a message.
  - b. Choose **Custom payload for each delivery protocol**, and then enter a JSON object to define the message to send for each delivery protocol.  
For more information, see [Publishing with platform-specific payload \(p. 244\)](#).
6. In the **Message attributes** section, add any attributes that you want Amazon SNS to match with the subscription attribute **FilterPolicy** to decide whether the subscribed endpoint is interested in the published message.
  - a. For **Type**, choose an attribute type, such as **String.Array**.

**Note**

For attribute type **String.Array**, enclose the array in square brackets ([ ]). Within the array, enclose string values in double quotation marks. You don't need quotation marks for numbers or for the keywords `true`, `false`, and `null`.

- b. Enter an attribute **Name**, such as `customer_interests`.
- c. Enter an attribute **Value**, such as `["soccer", "rugby", "hockey"]`.

If the attribute type is **String**, **String.Array**, or **Number**, Amazon SNS evaluates the message attribute against a subscription's [filter policy \(p. 71\)](#) (if present) before sending the message to the subscription.

For more information, see [Amazon SNS message attributes \(p. 68\)](#).

7. Choose **Publish message**.

The message is published to the topic, and the console opens the topic's **Details** page.

## To publish a message to a topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code examples show how to publish messages to an Amazon SNS topic.

.NET

**AWS SDK for .NET**

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish in AWS SDK for .NET API Reference](#).

C++

**SDK for C++**

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String message = argv[1];
    Aws::String topic_arn = argv[2];

    Aws::SNS::Model::PublishRequest psms_req;
    psms_req.SetMessage(message);
    psms_req.SetTopicArn(topic_arn);

    auto psms_out = sns.Publish(psms_req);

    if (psms_out.IsSuccess())
    {
        std::cout << "Message published successfully " << std::endl;
    }
    else
    {
        std::cout << "Error while publishing message " <<
        psms_out.GetError().GetMessage()
        << std::endl;
    }
}
```

```
    }

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for C++ API Reference*.

Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Go API Reference*.

Java

### SDK for Java 2.x

```
public static void pubTopic(SnsClient snsClient, String message, String topicArn) {

    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out.println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {PublishCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
var params = {
    Message: "MESSAGE_TEXT", // MESSAGE_TEXT
    TopicArn: "TOPIC_ARN", //TOPIC_ARN
};

const run = async () => {
    try {
        const data = await snsClient.send(new PublishCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
```

```
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

Publish a message with attributes so that a subscription can filter based on attributes.

```
class SnsWrapper:  
    """Encapsulates Amazon SNS topic and subscription functions."""  
    def __init__(self, sns_resource):  
        """  
        :param sns_resource: A Boto3 Amazon SNS resource.  
        """  
        self.sns_resource = sns_resource  
  
    def publish_message(topic, message, attributes):  
        """  
        Publishes a message, with attributes, to a topic. Subscriptions can be  
        filtered  
        based on message attributes so that a subscription receives messages only  
        when specified attributes are present.  
  
        :param topic: The topic to publish to.  
        :param message: The message to publish.  
        :param attributes: The key-value attributes to attach to the message.  
        Values  
            must be either `str` or `bytes`.  
        :return: The ID of the message.  
        """  
        try:  
            att_dict = {}  
            for key, value in attributes.items():  
                if isinstance(value, str):  
                    att_dict[key] = {'DataType': 'String', 'StringValue': value}  
                elif isinstance(value, bytes):  
                    att_dict[key] = {'DataType': 'Binary', 'BinaryValue': value}  
            response = topic.publish(Message=message, MessageAttributes=att_dict)  
            message_id = response['MessageId']  
            logger.info(  
                "Published message with attributes %s to topic %s.", attributes,  
                topic.arn)  
        except ClientError:  
            logger.exception("Couldn't publish message to topic %s.", topic.arn)  
            raise  
        else:  
            return message_id
```

Publish a message that takes different forms based on the protocol of the subscriber.

```
class SnsWrapper:  
    """Encapsulates Amazon SNS topic and subscription functions."""  
    def __init__(self, sns_resource):  
        """  
        :param sns_resource: A Boto3 Amazon SNS resource.
```

```

"""
    self.sns_resource = sns_resource

def publish_multi_message(
        topic, subject, default_message, sms_message, email_message):
"""
    Publishes a multi-format message to a topic. A multi-format message takes
    different forms based on the protocol of the subscriber. For example,
    an SMS subscriber might receive a short, text-only version of the message
    while an email subscriber could receive an HTML version of the message.

:param topic: The topic to publish to.
:param subject: The subject of the message.
:param default_message: The default version of the message. This version is
    sent to subscribers that have protocols that are
not
    otherwise specified in the structured message.
:param sms_message: The version of the message sent to SMS subscribers.
:param email_message: The version of the message sent to email subscribers.
:returns: The ID of the message.
"""
try:
    message = {
        'default': default_message,
        'sms': sms_message,
        'email': email_message
    }
    response = topic.publish(
        Message=json.dumps(message), Subject=subject,
        MessageStructure='json')
    message_id = response['MessageId']
    logger.info("Published multi-format message to topic %s.", topic.arn)
except ClientError:
    logger.exception("Couldn't publish message to topic %s.", topic.arn)
    raise
else:
    return message_id

```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish in AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### **SDK for Ruby**

```

require 'aws-sdk-sns' # v2: require 'aws-sdk'

def message_sent?(sns_client, topic_arn, message)

    sns_client.publish(topic_arn: topic_arn, message: message)
rescue StandardError => e
    puts "Error while sending the message: #{e.message}"
end

def run_me

topic_arn = 'SNS_TOPIC_ARN'
region = 'REGION'
message = 'MESSAGE' # The text of the message to send.

```

```
sns_client = Aws::SNS::Client.new(region: region)

puts "Message sending."

if message_sent?(sns_client, topic_arn, message)
  puts 'The message was sent.'
else
  puts 'The message was not sent. Stopping program.'
  exit 1
end
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for Ruby API Reference](#).

## Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`, topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);
    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in [AWS SDK for Rust API reference](#).

# Publishing large messages with Amazon SNS and Amazon S3

To publish large Amazon SNS messages, you can use the [The Amazon SNS Extended Client Library for Java](#). This library is useful for messages that are larger than the current maximum of 256 KB, up to maximum of 2 GB. The library saves actual payload to an Amazon S3 bucket and publishes the reference of the stored Amazon S3 object to the topic. Subscribed Amazon SQS queues can use the [Amazon SQS Extended Client Library for Java](#) to de-reference and retrieve payloads from Amazon S3. Other endpoints, such as Lambda, can use the [Payload Offloading Java Common Library for AWS](#) to de-reference and retrieve the payload.

## Prerequisites

The following are the prerequisites for using the Amazon SNS Extended Client Library for Java:

- An AWS SDK.

The example on this page uses the AWS Java SDK. To install and set up the SDK, see [Set up the AWS SDK for Java](#) in the *AWS SDK for Java Developer Guide*.

- An AWS account with the proper credentials.

To create an AWS account, navigate to the [AWS home page](#), and then choose **Create an AWS Account**. Follow the instructions.

For information about credentials, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java Developer Guide*.

- Java 8 or better.
- [The Amazon SNS Extended Client Library for Java](#) (also available from [Maven](#)).

## Configuring message storage

The Amazon SNS Extended Payload library uses on the Payload Offloading Java Common Library for AWS for message storage and retrieval. You can configure the following Amazon S3 [message storage options](#):

- Custom message sizes threshold – Messages with payloads and attributes that exceed this size are automatically stored in Amazon S3.
- `alwaysThroughS3` flag – Set this value to `true` to force all message payloads to be stored in Amazon S3. For example:

```
SNSExtendedClientConfiguration snsExtendedClientConfiguration = new
    SNSExtendedClientConfiguration() .withPayloadSupportEnabled(s3Client,
    BUCKET_NAME).withAlwaysThroughS3(true);
```

- Custom KMS key – The key to use for server-side encryption in your Amazon S3 bucket.
- Bucket name – The name of the Amazon S3 bucket for storing message payloads.

## Example: Publishing messages to Amazon SNS with payload stored in Amazon S3

The following shows an example of how to do the following:

- Create a sample topic and queue.
- Subscribe the queue to receive messages from the topic.
- Publish a test message.

The message payload is stored in Amazon S3 and the reference to it is published. The Amazon SQS Extended Client is used to receive the message.

#### SDK for Java 1.x

To publish a large message, use the Amazon SNS Extended Client Library for Java. The message that you send references an Amazon S3 object containing the actual message content.

```
import com.amazonaws.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.sqs.javamessaging.ExtendedClientConfiguration;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;
import com.amazonaws.services.sns.util.Topics;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import software.amazon sns AmazonsNSExtendedClient;
import software.amazon sns SNSExtendedClientConfiguration;

public class Example {

    public static void main(String[] args) {
        final String BUCKET_NAME = "extended-client-bucket";
        final String TOPIC_NAME = "extended-client-topic";
        final String QUEUE_NAME = "extended-client-queue";
        final Regions region = Regions.DEFAULT_REGION;

        //Message threshold controls the maximum message size that will be allowed to
        be published
        //through SNS using the extended client. Payload of messages exceeding this
        value will be stored in
        //S3. The default value of this parameter is 256 KB which is the maximum
        message size in SNS (and SQS).
        final int EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD = 32;

        //Initialize SNS, SQS and S3 clients
        final AmazonSNS snsClient =
        AmazonSNSClientBuilder.standard().withRegion(region).build();
        final AmazonSQS sqsClient =
        AmazonSQSClientBuilder.standard().withRegion(region).build();
        final AmazonS3 s3Client =
        AmazonS3ClientBuilder.standard().withRegion(region).build();

        //Create bucket, topic, queue and subscription
        s3Client.createBucket(BUCKET_NAME);
        final String topicArn = snsClient.createTopic(
            new CreateTopicRequest().withName(TOPIC_NAME)
        ).getTopicArn();
        final String queueUrl = sqsClient.createQueue(
            new CreateQueueRequest().withQueueName(QUEUE_NAME)
        ).getQueueUrl();
```

```

final String subscriptionArn = Topics.subscribeQueue(
    snsClient, sqsClient, topicArn, queueUrl
);

//To read message content stored in S3 transparently through SQS extended
client,
//set the RawMessageDelivery subscription attribute to TRUE
final SetSubscriptionAttributesRequest subscriptionAttributesRequest = new
SetSubscriptionAttributesRequest();
subscriptionAttributesRequest.setSubscriptionArn(subscriptionArn);
subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
subscriptionAttributesRequest.setAttributeValue("TRUE");
snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);

//Initialize SNS extended client
//PayloadSizeThreshold triggers message content storage in S3 when the
threshold is exceeded
//To store all messages content in S3, use AlwaysThroughS3 flag
final SNSExtendedClientConfiguration snsExtendedClientConfiguration = new
SNSExtendedClientConfiguration()
    .withPayloadSupportEnabled(s3Client, BUCKET_NAME)
    .withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
final AmazonSNSExtendedClient snsExtendedClient = new
AmazonSNSExtendedClient(snsClient, snsExtendedClientConfiguration);

//Publish message via SNS with storage in S3
final String message = "This message is stored in S3 as it exceeds the
threshold of 32 bytes set above.";
snsExtendedClient.publish(topicArn, message);

//Initialize SQS extended client
final ExtendedClientConfiguration sqsExtendedClientConfiguration = new
ExtendedClientConfiguration()
    .withPayloadSupportEnabled(s3Client, BUCKET_NAME);
final AmazonSQSExtendedClient sqsExtendedClient =
    new AmazonSQSExtendedClient(sqsClient, sqsExtendedClientConfiguration);

//Read the message from the queue
final ReceiveMessageResult result = sqsExtendedClient.receiveMessage(queueUrl);
System.out.println("Received message is " +
result.getMessages().get(0).getBody());
}
}

```

- Find instructions and more code on [GitHub](#).

## Other endpoint protocols

Both the Amazon SNS and Amazon SQS libraries use the [Payload Offloading Java Common Library for AWS](#) to store and retrieve message payloads with Amazon S3. Any Java-enabled endpoint (for example, an HTTPS endpoint that's implemented in Java) can use the same library to de-reference the message content.

Endpoints that can't use the Payload Offloading Java Common Library for AWS can still publish messages with payloads stored in Amazon S3. The following is an example of an Amazon S3 reference that is published by the above code example:

```
[ "software.amazon.payloadoffloading.PayloadS3Pointer",
{
```

```
    "s3BucketName": "extended-client-bucket",
    "s3Key": "xxxx-xxxxx-xxxxx-xxxxxx"
}
]
```

## Amazon SNS message attributes

Amazon SNS supports delivery of message attributes, which let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) about the message. For attribute mapping between Amazon SNS and Amazon SQS, each message can have up to 10 attributes. When using raw mode or an endpoint other than Amazon SQS, a message can have more than 10 attributes.

Message attributes are optional and separate from—but are sent together with—the message body. The receiver can use this information to decide how to handle the message without having to process the message body first.

For information about sending messages with attributes using the AWS Management Console or the AWS SDK for Java, see the [To publish messages to Amazon SNS topics using the AWS Management Console \(p. 58\)](#) tutorial.

**Note**

Message attributes are sent only when the message structure is String, not JSON.

You can also use message attributes to help structure the push notification message for mobile endpoints. In this scenario, the message attributes are used only to help structure the push notification message. The attributes are not delivered to the endpoint as they are when sending messages with message attributes to Amazon SQS endpoints.

You can also use message attributes to make your messages filterable using subscription filter policies. You can apply filter policies to topic subscriptions. When a filter policy is applied, a subscription receives only those messages that have attributes that the policy accepts. For more information, see [Amazon SNS message filtering \(p. 71\)](#).

## Message attribute items and validation

Each message attribute consists of the following items:

- **Name** – The message attribute name can contain the following characters: A-Z, a-z, 0-9, underscore(\_), hyphen(-), and period (.). The name must not start or end with a period, and it should not have successive periods. The name is case-sensitive and must be unique among all attribute names for the message. The name can be up to 256 characters long. The name cannot start with AWS . or Amazon . (or any variations in casing) because these prefixes are reserved for use by Amazon Web Services.
- **Type** – The supported message attribute data types are `String`, `String.Array`, `Number`, and `Binary`. The data type has the same restrictions on the content as the message body. The data type is case-sensitive, and it can be up to 256 bytes long. For more information, see the [Message attribute data types and validation \(p. 69\)](#) section.
- **Value** – The user-specified message attribute value. For string data types, the value attribute has the same restrictions on the content as the message body. For more information, see the `Publish` action in the *Amazon Simple Notification Service API Reference*.

Name, type, and value must not be empty or null. In addition, the message body should not be empty or null. All parts of the message attribute, including name, type, and value, are included in the message size restriction, which is 256 KB.

## Message attribute data types and validation

Message attribute data types identify how the message attribute values are handled by Amazon SNS. For example, if the type is a number, Amazon SNS validates that it's a number.

Amazon SNS supports the following logical data types for all endpoints except as noted:

- **String** – Strings are Unicode with UTF-8 binary encoding. For a list of code values, see [http://en.wikipedia.org/wiki/ASCII#ASCII\\_printable\\_characters](http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters).

### Note

Surrogate values are not supported in the message attributes. For example, using a surrogate value to represent an emoji will give you the following error: `Invalid attribute value was passed in for message attribute.`

- **String.Array** – An array, formatted as a string, that can contain multiple values. The values can be strings, numbers, or the keywords `true`, `false`, and `null`.

This data type isn't supported for AWS Lambda subscriptions. If you specify this data type for Lambda endpoints, it's passed as the `String` data type in the JSON payload that Amazon SNS delivers to Lambda.

- **Number** – Numbers are positive or negative integers or floating-point numbers. Numbers have sufficient range and precision to encompass most of the possible values that integers, floats, and doubles typically support. A number can have a value from  $-10^9$  to  $10^9$ , with 5 digits of accuracy after the decimal point. Leading and trailing zeroes are trimmed.

This data type isn't supported for AWS Lambda subscriptions. If you specify this data type for Lambda endpoints, it's passed as the `string` data type in the JSON payload that Amazon SNS delivers to Lambda.

- **Binary** – Binary type attributes can store any binary data; for example, compressed data, encrypted data, or images.

## Reserved message attributes for mobile push notifications

The following table lists the reserved message attributes for mobile push notification services that you can use to structure your push notification message:

Push notification service	Reserved message attribute
ADM	<code>AWS.SNS.MOBILE.ADM.TTL</code>
APNs	<code>AWS.SNS.MOBILE.APNS_MDM.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_MDM_SANDBOX.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_PASSBOOK.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_PASSBOOK_SANDBOX.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_SANDBOX.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_VOIP.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_VOIP_SANDBOX.TTL</code>
	<code>AWS.SNS.MOBILE.APNS_COLLAPSE_ID</code>

<b>Push notification service</b>	<b>Reserved message attribute</b>
	AWS.SNS.MOBILE.APNS.PRIORITY
	AWS.SNS.MOBILE.APNS.PUSH_TYPE
	AWS.SNS.MOBILE.APNS.TOPIC
	AWS.SNS.MOBILE.APNS.TTL
Baidu	AWS.SNS.MOBILE.BAIDU.DeployStatus
	AWS.SNS.MOBILE.BAIDU.MessageKey
	AWS.SNS.MOBILE.BAIDU.MessageType
	AWS.SNS.MOBILE.BAIDU.TTL
FCM	AWS.SNS.MOBILE.FCM.TTL
	AWS.SNS.MOBILE.GCM.TTL
macOS	AWS.SNS.MOBILE.MACOS_SANDBOX.TTL
	AWS.SNS.MOBILE.MACOS.TTL
MPNS	AWS.SNS.MOBILE.MPNS.NotificationClass
	AWS.SNS.MOBILE.MPNS.TTL
	AWS.SNS.MOBILE.MPNS.Type
WNS	AWS.SNS.MOBILE.WNS.CachePolicy
	AWS.SNS.MOBILE.WNS.Group
	AWS.SNS.MOBILE.WNS.Match
	AWS.SNS.MOBILE.WNS.SuppressPopup
	AWS.SNS.MOBILE.WNS.Tag
	AWS.SNS.MOBILE.WNS.TTL
	AWS.SNS.MOBILE.WNS.Type

# Amazon SNS message filtering

By default, an Amazon SNS topic subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a *filter policy* to the topic subscription.

A filter policy is a simple JSON object containing attributes that define which messages the subscriber receives. When you publish a message to a topic, Amazon SNS compares the message attributes to the attributes in the filter policy for each of the topic's subscriptions. If any of the attributes match, Amazon SNS sends the message to the subscriber. Otherwise, Amazon SNS skips the subscriber without sending the message. If a subscription doesn't have a filter policy, the subscription receives every message published to its topic.

You can simplify your use of Amazon SNS by consolidating your message filtering criteria into your topic subscriptions. This allows you to offload the message filtering logic from subscribers and the message routing logic from publishers, eliminating the need to filter messages by creating a separate topic for each condition. You can use a single topic, differentiating your messages using attributes. Each subscriber receives and processes only the messages accepted by its filter policy.

For example, you can use a single topic to publish all messages generated by transactions from your retail website. To indicate the transaction state, you can assign an attribute (such as `order_placed`, `order_cancelled`, or `order_declined`) to each message. By creating subscriptions with filter policies, you can route each message to the queue designed to process the transaction state of the message.

For more information, see [Filter Messages Published to Topics](#).

## Topics

- [Amazon SNS subscription filter policies \(p. 71\)](#)
- [Applying a subscription filter policy \(p. 79\)](#)
- [Removing a subscription filter policy \(p. 82\)](#)

# Amazon SNS subscription filter policies

A subscription filter policy allows you to specify attribute names and assign a list of values to each attribute name. For more information, see [Amazon SNS message filtering \(p. 71\)](#).

When Amazon SNS evaluates message attributes against the subscription filter policy, it ignores message attributes that aren't specified in the policy.

## Important

AWS services such as IAM and Amazon SNS use a distributed computing model called eventual consistency. Additions or changes to a subscription filter policy require up to 15 minutes to fully take effect.

A subscription accepts a message under the following conditions:

- Each attribute name in a filter policy matches an attribute name assigned to the message.
- For each matching attribute name, at least one match exists between the following:
  - The values of the attribute name in the filter policy
  - The message attributes

## Topics

- [Example filter policies \(p. 72\)](#)

- [Filter policy constraints \(p. 73\)](#)
- [Attribute string value matching \(p. 74\)](#)
- [Attribute numeric value matching \(p. 76\)](#)
- [Attribute key matching \(p. 77\)](#)
- [AND/OR logic \(p. 78\)](#)

## Example filter policies

The following example shows a message payload sent by an Amazon SNS topic that publishes customer transactions. The `MessageAttributes` field includes attributes that describe the transaction:

- Customer's interests
- Store name
- Event state
- Purchase price in USD

Because this message includes the `MessageAttributes` field, any topic subscription that includes a filter policy can selectively accept or reject the message.

```
{  
    "Type": "Notification",  
    "MessageId": "a1b2c34d-567e-8f90-g1h2-i345j67klmn8",  
    "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",  
    "Message": "message-body-with-transaction-details",  
    "Timestamp": "2019-11-03T23:28:01.631Z",  
    "SignatureVersion": "4",  
    "Signature": "signature",  
    "UnsubscribeURL": "unsubscribe-url",  
    "MessageAttributes": {  
        "customer_interests": {  
            "Type": "String.Array",  
            "Value": "[\"soccer\", \"rugby\", \"hockey\"]"  
        },  
        "store": {  
            "Type": "String",  
            "Value": "example_corp"  
        },  
        "event": {  
            "Type": "String",  
            "Value": "order_placed"  
        },  
        "price_usd": {  
            "Type": "Number",  
            "Value": "210.75"  
        }  
    }  
}
```

For information about applying attributes to a message, see [Amazon SNS message attributes \(p. 68\)](#).

The following filter policies accept or reject messages based on their attribute names and values.

### A policy that accepts the example message

The attributes in the following subscription filter policy match the attributes assigned to the example message.

If any single attribute in this policy doesn't match an attribute assigned to the message, the policy rejects the message.

```
{
  "store": ["example_corp"],
  "event": [{"anything-but": "order_cancelled"}],
  "customer_interests": [
    "rugby",
    "football",
    "baseball"
  ],
  "price_usd": [{"numeric": [">=", 100]}]
}
```

## A policy that rejects the example message

The following subscription filter policy has multiple mismatches between its attributes and the attributes assigned to the example message. Because the `encrypted` attribute name isn't present in the message attributes, this policy attribute causes the message to be rejected regardless of the value assigned to it.

If any mismatches occur, the policy rejects the message.

```
{
  "store": ["example_corp"],
  "event": ["order_cancelled"],
  "encrypted": [false],
  "customer_interests": [
    "basketball",
    "baseball"
  ]
}
```

## Filter policy constraints

When you create a filter policy, keep the following constraints in mind:

- For the `String` data type, the attribute comparison between policy and message is case-sensitive.
- A numeric policy attribute can have a value from  $-10^9$  to  $10^9$ , with 5 digits of accuracy after the decimal point.
- The total combination of values must not exceed 150. Calculate the total combination by multiplying the number of values in each array.

Consider the following policy:

```
{
  "key_a": ["value_one", "value_two", "value_three"],
  "key_b": ["value_one"],
  "key_c": ["value_one", "value_two"]
}
```

The first array has three values, the second has one value, and the third has two values. The total combination is calculated as follows:

```
3 x 1 x 2 = 6
```

- Amazon SNS compares policy attributes only to message attributes that have the following data types:

- `String`
- `String.Array`
- `Number`
- Amazon SNS ignores message attributes with the `Binary` data type.
- The JSON of the filter policy can contain the following:
  - strings enclosed in quotation marks
  - numbers
  - the keywords `true`, `false`, and `null`, without quotation marks
- When you use the Amazon SNS API, you must pass the JSON of the filter policy as a valid UTF-8 string.
- A filter policy can have a maximum of 5 attribute names.
- The maximum size of a policy is 256 KB.
- By default, you can have up to 200 filter policies per AWS account, per region. To increase this quota, submit a [quota increase request](#).

## Attribute string value matching

You can use string values to match message attributes and filter messages. String values are enclosed in double quotation marks in the JSON policy.

You can use the following string operations to match message attributes.

### Exact matching

Exact matching occurs when a policy attribute value matches one or more message attribute values.

Consider the following policy attribute:

```
"customer_interests": ["rugby", "tennis"]
```

It matches the following message attributes:

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

```
"customer_interests": {"Type": "String", "Value": "tennis"}
```

However, it doesn't match the following message attribute:

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

### Prefix matching

When a policy attribute includes the keyword `prefix`, it matches any message attribute value that begins with the specified characters.

Consider the following policy attribute:

```
"customer_interests": [{"prefix": "bas"}]
```

It matches either of the following message attributes:

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String", "Value": "basketball"}
```

However, it doesn't match the following message attribute:

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

## Anything-but matching

When a policy attribute value includes the keyword `anything-but`, it matches any message attribute that *doesn't* include any of the policy attribute values.

Consider the following policy attribute:

```
"customer_interests": [{"anything-but": ["rugby", "tennis"]}]
```

It matches either of the following message attributes:

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String", "Value": "football"}
```

It also matches the following message attribute (because it contains a value that *isn't* rugby or tennis):

```
"customer_interests": {"Type": "String.Array", "Value": "[\"rugby\", \"baseball\"]"}
```

However, it doesn't match the following message attribute:

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

## Using a prefix with the anything-but operator

For attribute `string` matching, you can also use a prefix with the `anything-but` operator.

For example, the following policy attribute denies the `order-` prefix:

```
"event": [{"anything-but": {"prefix": "order-"}}]
```

It matches either of the following attributes:

```
"event": {"Type": "String", "Value": "data-entry"}
```

```
"event": {"Type": "String", "Value": "order_number"}
```

However, it *doesn't* match the following attribute:

```
"event": {"Type": "String", "Value": order-cancelled}
```

## IP address matching

You can use the `cidr` operator to check whether an incoming message originates from a specific IP address or subnet.

Consider the following policy attribute:

```
"source_ip": [{"cidr": "10.0.0.0/24"}]
```

It matches either of the following message attributes:

```
"source_ip": {"Type": "String", "Value": "10.0.0.0"}
```

```
"source_ip": {"Type": "String", "Value": "10.0.0.255"}
```

However, it *doesn't* match the following message attribute:

```
"source_ip": {"Type": "String", "Value": "10.1.1.0"}
```

## Attribute numeric value matching

You can use numeric values to match message attributes and filter messages. Numeric values aren't enclosed in double quotation marks in the JSON policy. You can use the following numeric operations to match message attributes.

**Note**

Prefixes are supported for attribute *string* matching only.

## Exact matching

When a policy attribute value includes the keyword `numeric` and the operator `=`, it matches any message attribute that has the same name and an equal numeric value.

Consider the following policy attribute:

```
"price_usd": [{"numeric": ["=", 301.5]}]
```

It matches either of the following message attributes:

```
"price_usd": {"Type": "Number", "Value": 301.5}
```

```
"price_usd": {"Type": "Number", "Value": 3.015e2}
```

## Anything-but matching

When a policy attribute value includes the keyword `anything-but`, it matches any message attribute that *doesn't* include any of the policy attribute values.

Consider the following policy attribute:

```
"price": [{"anything-but": [100, 500]}]
```

It matches either of the following message attributes:

```
"price": {"Type": "Number", "Value": 101}
```

```
"price": {"Type": "Number", "Value": 100.1}
```

It also matches the following message attribute (because it contains a value that *isn't* 100 or 500):

```
"price": {"Type": "Number.Array", "Value": "[100, 50]"}
```

However, it doesn't match the following message attribute:

```
"price": {"Type": "Number", "Value": 100}
```

## Value range matching

In addition to the operator `=`, a numeric policy attribute can include the following operators: `<`, `<=`, `>`, and `>=`.

Consider the following policy attribute:

```
"price_usd": [{"numeric": ["<", 0]}]
```

It matches any message attributes with negative numeric values.

Consider another message attribute:

```
"price_usd": [{"numeric": [>, 0, "<=", 150]}]
```

It matches any message attributes with positive numbers up to and including 150.

## Attribute key matching

You can use the `exists` operator to return incoming messages with or without specified attributes in the filter policy:

- Use `"exists": true` to return incoming messages that include the specified attribute.

For example, the following attribute uses the `exists` operator with a value of `true`:

```
"store": [{"exists": true}]
```

It matches any message that contains the `store` attribute key, such as the following:

```
"store": "fans"  
"customer_interests": ["baseball", "basketball"]
```

However, it doesn't match any message *without* the `store` attribute key, such as the following:

```
"customer_interests": ["baseball", "basketball"]
```

- Use "exists": false to return incoming messages that *don't* include the specified attribute.

The following example shows the effect of using the exists operator with a value of false:

```
"store": [{"exists": false}]
```

It *doesn't* match any message that contains the store attribute key, such as the following:

```
"store": "fans"
"customer_interests": ["baseball", "basketball"]
```

However, it matches any messages *without* the store attribute key, such as the following:

```
"customer_interests": ["baseball", "basketball"]
```

## AND/OR logic

You can use operations that include AND/OR logic to match message attributes.

### AND logic

You can apply AND logic using multiple attribute names.

Consider the following policy:

```
{
  "customer_interests": ["rugby"],
  "price_usd": [{"numeric": [">>, 100]}]
}
```

It matches any message attributes with the value of customer\_interests set to rugby *and* the value of price\_usd set to a number larger than 100.

### OR logic

You can apply OR logic by assigning multiple values to an attribute name.

Consider the following policy attribute:

```
"customer_interests": ["rugby", "football", "baseball"]
```

It matches any message attributes with the value of customer\_interests set to rugby, football, or baseball.

#### Note

Currently, you can't use SNS filters to apply OR logic across different message attributes. Instead, you can use multiple SNS subscriptions with different endpoints to achieve the same effect.

For example, assume that you have message attributes named customer\_interests and customer\_preferences. To apply OR logic across both attributes, create an SNS subscription to match each message attribute. Then, you can use your subscriber application to consume both types of messages through the different endpoints.

# Applying a subscription filter policy

You can apply a filter policy to an Amazon SNS subscription using the Amazon SNS console. Or, to apply policies programmatically, you can use the Amazon SNS API, the AWS Command Line Interface (AWS CLI), or any AWS SDK that supports Amazon SNS.

## Important

AWS services such as IAM and Amazon SNS use a distributed computing model called eventual consistency. Additions or changes to a subscription filter policy require up to 15 minutes to fully take effect.

## AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Subscriptions**.
3. Select a subscription and then choose **Edit**.
4. On the **Edit EXAMPLE1-23bc-4567-d890-ef12g3hij456** page, expand the **Subscription filter policy** section.
5. In the **JSON editor** field, provide the JSON body of your filter policy.
6. Choose **Save changes**.

Amazon SNS applies your filter policy to the subscription.

## AWS CLI

To apply a filter policy with the AWS Command Line Interface (AWS CLI), use the `set-subscription-attributes` command, as shown in the following example:

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns:... --attribute-name FilterPolicy --attribute-value "{\"store\":[\"example_corp\"], \"event\":[\"order_placed\"]}"
```

For the `--attribute-name` option, specify `FilterPolicy`. For `--attribute-value`, specify your JSON policy.

To provide valid JSON for your policy, enclose the attribute names and values in double quotes. You must also enclose the entire policy argument in quotes. To avoid escaping quotes, you can use single quotes to enclose the policy and double quotes to enclose the JSON names and values, as shown in the example.

To verify that your filter policy was applied, use the `get-subscription-attributes` command. The attributes in the terminal output should show your filter policy for the `FilterPolicy` key, as shown in the following example:

```
$ aws sns get-subscription-attributes --subscription-arn arn:aws:sns:...
{
    "Attributes": {
        "Endpoint": "endpoint . . .",
        "Protocol": "https",
        "RawMessageDelivery": "false",
        "EffectiveDeliveryPolicy": "delivery policy . . .",
        "ConfirmationWasAuthenticated": "true",
        "FilterPolicy": "{\"store\":[\"example_corp\"], \"event\":[\"order_placed\"]}",
        "Owner": "111122223333",
        ...
    }
}
```

```
        "SubscriptionArn": "arn:aws:sns: . . .",
        "TopicArn": "arn:aws:sns: . . ."
    }
}
```

## AWS SDKs

The following code examples show how to set an Amazon SNS filter policy.

Java

### SDK for Java 2.x

```
public static void usePolicy(SnsClient snsClient, String subscriptionArn) {

    try {
        SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();

        // Add a filter policy attribute with a single value
        fp.addAttribute("store", "example_corp");
        fp.addAttribute("event", "order_placed");

        // Add a prefix attribute
        fp.addAttributePrefix("customer_interests", "bas");

        // Add an anything-but attribute
        fp.addAttributeAnythingBut("customer_interests", "baseball");

        // Add a filter policy attribute with a list of values
        ArrayList<String> attributeValues = new ArrayList<>();
        attributeValues.add("rugby");
        attributeValues.add("soccer");
        attributeValues.add("hockey");
        fp.addAttribute("customer_interests", attributeValues);

        // Add a numeric attribute
        fp.addAttribute("price_usd", "=", 0);

        // Add a numeric attribute with a range
        fp.addAttributeRange("price_usd", ">", 0, "<=", 100);

        // Apply the filter policy attributes to an Amazon SNS subscription
        fp.apply(snsClient, subscriptionArn);

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSubscriptionAttributes](#) in *AWS SDK for Java 2.x API Reference*.

Python

### SDK for Python (Boto3)

```
class SnsWrapper:
```

```
"""Encapsulates Amazon SNS topic and subscription functions."""
def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

    def add_subscription_filter(subscription, attributes):
        """
        Adds a filter policy to a subscription. A filter policy is a key and a
        list of values that are allowed. When a message is published, it must have
        an
        attribute that passes the filter or it will not be sent to the
        subscription.

        :param subscription: The subscription the filter policy is attached to.
        :param attributes: A dictionary of key-value pairs that define the filter.
        """
        try:
            att_policy = {key: [value] for key, value in attributes.items()}
            subscription.set_attributes(
                AttributeName='FilterPolicy',
                AttributeValue=json.dumps(att_policy))
            logger.info("Added filter to subscription %s.", subscription.arn)
        except ClientError:
            logger.exception(
                "Couldn't add filter to subscription %s.", subscription.arn)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSubscriptionAttributes](#) in *AWS SDK for Python (Boto3) API Reference*.

## Amazon SNS API

To apply a filter policy with the Amazon SNS API, make a request to the [SetSubscriptionAttributes](#) action. Set the `AttributeName` parameter to `FilterPolicy`, and set the `AttributeValue` parameter to your filter policy JSON.

## AWS CloudFormation

To apply a filter policy using AWS CloudFormation, use a JSON or YAML template to create a AWS CloudFormation stack. For more information, see the [FilterPolicy property](#) of the `AWS::SNS::Subscription` resource in the *AWS CloudFormation User Guide* and the [example AWS CloudFormation template](#).

1. Sign in to the [AWS CloudFormation console](#).
2. Choose **Create Stack**.
3. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.
4. On the **Specify Details** page, do the following:
  - a. For **Stack Name**, type `MyFilterPolicyStack`.
  - b. For **myHttpEndpoint**, type the HTTP endpoint to be subscribed to your topic.

**Tip**

If you don't have an HTTP endpoint, create one.

5. On the **Options** page, choose **Next**.

6. On the **Review** page, choose **Create**.

## Removing a subscription filter policy

To stop filtering the messages that are sent to a subscription, remove the subscription's filter policy by overwriting it with an empty JSON body. After you remove the policy, the subscription accepts every message that's published to it.

### AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Subscriptions**.
3. Select a subscription and then choose **Edit**.
4. On the **Edit EXAMPLE1-23bc-4567-d890-ef12g3hij456** page, expand the **Subscription filter policy** section.
5. In the **JSON editor** field, provide an empty JSON body for your filter policy: {}.
6. Choose **Save changes**.

Amazon SNS applies your filter policy to the subscription.

### AWS CLI

To remove a filter policy with the AWS CLI, use the `set-subscription-attributes` command and provide an empty JSON body for the `--attribute-value` argument:

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns:... --attribute-name FilterPolicy --attribute-value "{}"
```

### Amazon SNS API

To remove a filter policy with the Amazon SNS API, make a request to the `SetSubscriptionAttributes` action. Set the `AttributeName` parameter to `FilterPolicy`, and provide an empty JSON body for the `AttributeValue` parameter.

# Amazon SNS message delivery

This section describes how message delivery works.

## Topics

- [Amazon SNS raw message delivery \(p. 83\)](#)
- [Sending Amazon SNS messages to an Amazon SQS queue in a different account \(p. 84\)](#)
- [Sending Amazon SNS messages to an Amazon SQS queue or AWS Lambda function in a different Region \(p. 86\)](#)
- [Amazon SNS message delivery status \(p. 87\)](#)
- [Amazon SNS message delivery retries \(p. 92\)](#)
- [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#)

## Amazon SNS raw message delivery

To avoid having [Amazon Kinesis Data Firehose \(p. 103\)](#), [Amazon SQS \(p. 124\)](#), and [HTTP/S \(p. 134\)](#) endpoints process the JSON formatting of messages, Amazon SNS allows raw message delivery:

- When you enable raw message delivery for Amazon Kinesis Data Firehose or Amazon SQS endpoints, any Amazon SNS metadata is stripped from the published message and the message is sent as is.
- When you enable raw message delivery for HTTP/S endpoints, the HTTP header `x-amz-sns-rawdelivery` with its value set to `true` is added to the message, indicating that the message has been published without JSON formatting.

To enable raw message delivery using an AWS SDK, you must use the `SetSubscriptionAttribute` API action and set the value of the `RawMessageDelivery` attribute to `true`.

## Enabling raw message delivery using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic subscribed to an Kinesis Data Firehose, Amazon SQS, or HTTP/S endpoint.
4. On the **MyTopic** page, in the **Subscription** section, choose a subscription and choose **Edit**.
5. On the **Edit EXAMPLE1-23bc-4567-d890-ef12g3hij456** page, in the **Details** section, choose **Enable raw message delivery**.
6. Choose **Save changes**.

## Message format examples

In the following examples, the same message is sent to the same Amazon SQS queue twice. The only difference is that raw message delivery is disabled for the first message, and enabled for the second.

- Raw message delivery is **disabled**

```
{  
    "Type": "Notification",  
    "MessageId": "dc1e94d9-56c5-5e96-808d-cc7f68faa162",  
    "TopicArn": "arn:aws:sns:us-east-2:11122223333:ExampleTopic1",  
    "Subject": "TestSubject",  
    "Message": "This is a test message.",  
    "Timestamp": "2021-02-16T21:41:19.978Z",  
    "SignatureVersion": "1",  
    "Signature": "FMG5tlZhJNHLHUXvZgtZzlk24FzVa7oX0T4P03neeXw8ZEXZx6z35j2FOTuNYShn2h0bKNC/  
zLTnMyIxEzmi2X1shOBWsJHkrW2xkR58ABZF+4uWHEE73yDVR4SyYAikP9jstZzDRm  
+bcVs8+T0yaLiEGLrIIIL4esi1llhIkglErCuy5btPcWXBdio2fpCRD5x9oR6gmE/  
rd5071X1cluvnv4r1Lkk4pqP2/iUfxFZva1xLSRvgymfm6D9hNklVypfy  
+7TalMD0lzmJuOrExtnSibZew3foxgx8GT+lbZkLd0ZtdtRJ1IyPRP44eyq78sU0Eo/LsDr0Iak4ZDpg8dXg==",  
    "SigningCertURL": "https://sns.us-east-2.amazonaws.com/  
SimpleNotificationService-010a507c1833636cd94bdb98bd93083a.pem",  
    "UnsubscribeURL": "https://sns.us-east-2.amazonaws.com/?  
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-  
east-2:11122223333:ExampleTopic1:e1039402-24e7-40a3-a0d4-797da162b297"  
}
```

- Raw message delivery is **enabled**

```
This is a test message.
```

## Sending Amazon SNS messages to an Amazon SQS queue in a different account

You can publish a notification to an Amazon SNS topic with one or more subscriptions to Amazon SQS queues in another account. You set up the topic and queues the same way you would if they were in the same account (see [Fanout to Amazon SQS queues \(p. 124\)](#)). The major difference is how you handle subscription confirmation, and that depends on how you subscribe the queue to the topic.

### Topics

- [Queue owner creates subscription \(p. 84\)](#)
- [A user who does not own the queue creates subscription \(p. 85\)](#)

## Queue owner creates subscription

The account that created the Amazon SQS queue is the queue owner. When the queue owner creates a subscription, the subscription doesn't require confirmation. The queue begins to receive notifications from the topic as soon as the `Subscribe` action completes. To let the queue owner subscribe to the topic owner's topic, the topic owner must give the queue owner's account permission to call the `Subscribe` action on the topic.

### Step 1: To set the topic policy using the AWS Management Console

- Sign in to the [Amazon SNS console](#).
- On the navigation panel, choose **Topics**.
- Select a topic and then choose **Edit**.
- On the **Edit MyTopic** page, expand the **Access policy** section.

5. Enter the following policy:

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "111122223333"  
            },  
            "Action": "sns:Subscribe",  
            "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"  
        }]  
}
```

This policy gives account 111122223333 permission to call sns:Subscribe on MyTopic in account 123456789012.

6. Choose **Save changes**.

A user with the credentials for account 111122223333 can subscribe to MyTopic.

## Step 2: To add an Amazon SQS queue subscription to a topic in another AWS account using the AWS Management Console

Before you begin, make sure you have the ARNs for your topic and queue and that you have [given permission to the topic to send messages to the queue \(p. 125\)](#).

1. On the navigation panel, choose **Subscriptions**.
2. On the **Subscriptions** page, choose **Create subscription**
3. On the **Create subscription** page, in the **Details** section, do the following:
  - a. For **Topic ARN**, enter the ARN of the topic.
  - b. For **Protocol**, choose **Amazon SQS**.
  - c. For **Endpoint**, enter the ARN of the queue.
  - d. Choose **Create subscription**.

### Note

- To be able to communicate with the service, the queue must have permissions for Amazon SNS.
- Because you are the owner of the queue, you don't have to confirm the subscription.

## A user who does not own the queue creates subscription

Any user who creates a subscription but isn't the owner of the queue must confirm the subscription.

When you use the **Subscribe** action, Amazon SNS sends a subscription confirmation to the queue. The subscription is displayed in the Amazon SNS console, with its subscription ID set to **Pending Confirmation**.

To confirm the subscription, a user with permission to read messages from the queue must visit the subscription URL. Until the subscription is confirmed, no notifications published to the topic are sent to the queue. To confirm the subscription, you can use the Amazon SQS console or the [ReceiveMessage](#) action.

**Note**

Before you subscribe an endpoint to the topic, make sure that the queue can receive messages from the topic by setting the `sqs:SendMessage` permission for the queue. For more information, see [Step 2: Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue \(p. 125\)](#).

## To confirm a subscription using the AWS Management Console

1. Sign in to the [Amazon SQS console](#).
2. Select the queue that has a pending subscription to the topic.
3. Choose **Queue Actions, View/Delete Messages** and then choose **Start Polling for Messages**.  
A message with the subscription confirmation is received in the queue.
4. In the **Body** column, do the following:
  - a. Choose **More Details**.
  - b. In the **Message Details** dialog box, find and note the **SubscribeURL** value, for example:

```
https://sns.us-west-2.amazonaws.com/?  
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-  
east-2:123456789012:MyTopic&Token=2336412f37fb...
```

5. In a web browser, navigate to the URL.

An XML response is displayed, for example:

```
<ConfirmSubscriptionResponse>  
  <ConfirmSubscriptionResult>  
    <SubscriptionArn>arn:aws:sns:us-east-2:123456789012:MyTopic:1234a567-  
bc89-012d-3e45-6fg7h890123i</SubscriptionArn>  
  </ConfirmSubscriptionResult>  
  <ResponseMetadata>  
    <RequestId>abcd1efg-23hi-jkl4-m5no-p67q8rstuvwxyz9</RequestId>  
  </ResponseMetadata>  
</ConfirmSubscriptionResponse>
```

The subscribed queue is ready to receive messages from the topic.

6. (Optional) If you view the topic subscription in the Amazon SNS console, you can see that the **Pending Confirmation** message has been replaced by the subscription ARN in the **Subscription ID** column.

## Sending Amazon SNS messages to an Amazon SQS queue or AWS Lambda function in a different Region

Amazon SNS supports cross-region deliveries, both for Regions that are enabled by default and for [opt-in Regions \(p. 87\)](#). For the current list of AWS Regions that Amazon SNS supports, including opt-in Regions, see [Amazon Simple Notification Service endpoints and quotas](#) in the *Amazon Web Services General Reference*.

Amazon SNS supports the cross-region delivery of notifications to Amazon SQS queues and to AWS Lambda functions. When one of the Regions is an opt-in Region, you must specify a different Amazon SNS service principal in the subscribed resource's policy.

## Opt-in Regions

Amazon SNS supports the following opt-in Regions:

- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Europe (Milan)
- Middle East (Bahrain)

For information on enabling an opt-in Region, see [Managing AWS Regions](#) in the *Amazon Web Services General Reference*.

When you use Amazon SNS to deliver messages from opt-in Regions to Regions that are enabled by default, you must alter the resource policy created for the queue. Replace the principal `sns.amazonaws.com` with `sns.<opt-in-region>.amazonaws.com`. For example:

- To subscribe an Amazon SQS queue in US East (N. Virginia) to an SNS topic in Asia Pacific (Hong Kong), change the principal in the queue policy to `sns.ap-east-1.amazonaws.com`. Opt-in regions include any regions launched after March 20, 2019, which includes Asia Pacific (Hong Kong), Middle East (Bahrain), EU (Milano), and Africa (Cape Town). Regions launched prior to March 20, 2019 are enabled by default.

**Note**

AWS also supports cross-region delivery to Amazon SQS from a region that is enabled by default to an opt-in region. However, cross-region forwarding of SNS messages from opt-in regions to other opt-in regions is not supported.

- To subscribe an AWS Lambda function in US East (N. Virginia) to an SNS topic in Asia Pacific (Hong Kong), change the principal in the AWS Lambda function policy to `sns.ap-east-1.amazonaws.com`. Opt-in regions include any regions launched after March 20, 2019, which includes Asia Pacific (Hong Kong), Middle East (Bahrain), EU (Milano), and Africa (Cape Town). Regions launched prior to March 20, 2019 are enabled by default.

**Note**

AWS doesn't support cross-region delivery to Lambda from a region that is enabled by default to an opt-in region. Also, cross-region forwarding of SNS messages from opt-in regions to other opt-in regions is not supported.

## Amazon SNS message delivery status

Amazon SNS provides support to log the delivery status of notification messages sent to topics with the following Amazon SNS endpoints:

- HTTP
- Amazon Kinesis Data Firehose
- AWS Lambda
- Platform application endpoint
- Amazon Simple Queue Service

After you configure the message delivery status attributes, log entries are sent to CloudWatch Logs for messages sent to topic subscribers. Logging message delivery status helps provide better operational insight, such as the following:

- Knowing whether a message was delivered to the Amazon SNS endpoint.
- Identifying the response sent from the Amazon SNS endpoint to Amazon SNS.
- Determining the message dwell time (the time between the publish timestamp and just before handing off to an Amazon SNS endpoint).

To configure topic attributes for message delivery status, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

#### Topics

- [Configuring delivery status logging using the AWS Management Console \(p. 88\)](#)
- [Configuring message delivery status attributes for topics subscribed to Amazon SNS endpoints using the AWS SDKs \(p. 88\)](#)

## Configuring delivery status logging using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic and then choose **Edit**.
4. On the **Edit MyTopic** page, expand the **Delivery status logging** section.
5. Choose the protocol for which you want to log delivery status; for example **AWS Lambda**.
6. Enter the **Success sample rate** (the percentage of successful messages for which you want to receive CloudWatch Logs).
7. In the **IAM roles** section, do one of the following:
  - To choose an existing service role from your account, choose **Use existing service role** and then specify IAM roles for successful and failed deliveries.
  - To create a new service role in your account, choose **Create new service role**, choose **Create new roles** to define the IAM roles for successful and failed deliveries in the IAM console.  
To give Amazon SNS write access to use CloudWatch Logs on your behalf, choose **Allow**.
8. Choose **Save changes**.

You can now view and parse the CloudWatch Logs containing the message delivery status. For more information about using CloudWatch, see the [CloudWatch Documentation](#).

## Configuring message delivery status attributes for topics subscribed to Amazon SNS endpoints using the AWS SDKs

The AWS SDKs provide APIs in several languages for using message delivery status attributes with Amazon SNS.

### Topic attributes

You can use the following topic attribute name values for message delivery status:

## HTTP

- `HTTPSuccessFeedbackRoleArn`
- `HTTPSuccessFeedbackSampleRate`
- `HTTPFailureFeedbackRoleArn`

## Amazon Kinesis Data Firehose

- `FirehoseSuccessFeedbackRoleArn`
- `FirehoseSuccessFeedbackSampleRate`
- `FirehoseFailureFeedbackRoleArn`

## AWS Lambda

- `LambdaSuccessFeedbackRoleArn`
- `LambdaSuccessFeedbackSampleRate`
- `LambdaFailureFeedbackRoleArn`

## Platform application endpoint

- `ApplicationSuccessFeedbackRoleArn`
- `ApplicationSuccessFeedbackSampleRate`
- `ApplicationFailureFeedbackRoleArn`

### Note

In addition to being able to configure topic attributes for message delivery status of notification messages sent to Amazon SNS application endpoints, you can also configure application attributes for the delivery status of push notification messages sent to push notification services. For more information, see [Using Amazon SNS Application Attributes for Message Delivery Status](#).

## Amazon SQS

- `SQSSuccessFeedbackRoleArn`
- `SQSSuccessFeedbackSampleRate`
- `SQSFailureFeedbackRoleArn`

The `<ENDPOINT>SuccessFeedbackRoleArn` and `<ENDPOINT>FailureFeedbackRoleArn` attributes are used to give Amazon SNS write access to use CloudWatch Logs on your behalf. The `<ENDPOINT>SuccessFeedbackSampleRate` attribute is for specifying the sample rate percentage (0-100) of successfully delivered messages. After you configure the `<ENDPOINT>FailureFeedbackRoleArn` attribute, then all failed message deliveries generate CloudWatch Logs.

## AWS SDK examples to configure topic attributes

The following code examples show how to set Amazon SNS topic attributes.

Java

[SDK for Java 2.x](#)

```
public static void setTopAttr(SnsClient snsClient, String attribute, String topicArn, String value) {

    try {

        SetTopicAttributesRequest request = SetTopicAttributesRequest.builder()
            .attributeName(attribute)
            .attributeValue(value)
            .topicArn(topicArn)
            .build();

        SetTopicAttributesResponse result =
snsClient.setTopicAttributes(request);
        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode() + "\n\nTopic " + request.topicArn()
        + " updated " + request.attributeName() + " to " +
request.attributeValue());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetTopicAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SetTopicAttributesCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
    AttributeName: "ATTRIBUTE_NAME", // ATTRIBUTE_NAME
    TopicArn: "TOPIC_ARN", // TOPIC_ARN
    AttributeValue: "NEW_ATTRIBUTE_VALUE", //NEW_ATTRIBUTE_VALUE
};

const run = async () => {
    try {
        const data = await snsClient.send(new SetTopicAttributesCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    }
```

```
    } catch (err) {
      console.log("Error", err.stack);
    }
  };
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [SetTopicAttributes](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
$attribute = 'Policy | DisplayName | DeliveryPolicy';
$value = 'First Topic';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->setTopicAttributes([
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetTopicAttributes](#) in [AWS SDK for PHP API Reference](#).

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

policy = '{
    "Version": "2008-10-17",
    "Id": "__default_policy_ID",
    "Statement": [
        {
            "Sid": "__default_statement_ID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": ["SNS:Publish"],
            "Resource": "' + MY_TOPIC_ARN + ''",
            "Condition": {
                "ArnEquals": {
                    "AWS:SourceArn": "' + MY_RESOURCE_ARN + ''"
                }
            }
        ]
}'
# Replace us-west-2 with the AWS Region you're using for Amazon SNS.
sns = Aws::SNS::Resource.new(region: 'REGION')

# Get topic by ARN
topic = sns.topic()

# Add policy to topic
topic.set_attributes({
    attribute_name: "POLICY_NAME",
    attribute_value: policy
})
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [SetTopicAttributes](#) in [AWS SDK for Ruby API Reference](#).

## Amazon SNS message delivery retries

Amazon SNS defines a *delivery policy* for each delivery protocol. The delivery policy defines how Amazon SNS retries the delivery of messages when server-side errors occur (when the system that hosts the subscribed endpoint becomes unavailable). When the delivery policy is exhausted, Amazon SNS stops retrying the delivery and discards the message—unless a dead-letter queue is attached to the subscription. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#).

### Topics

- [Delivery protocols and policies \(p. 92\)](#)
- [Delivery policy stages \(p. 93\)](#)
- [Creating an HTTP/S delivery policy \(p. 94\)](#)

## Delivery protocols and policies

### Note

- With the exception of HTTP/S, you can't change Amazon SNS-defined delivery policies. Only HTTP/S supports custom policies. See [Creating an HTTP/S delivery policy \(p. 94\)](#).

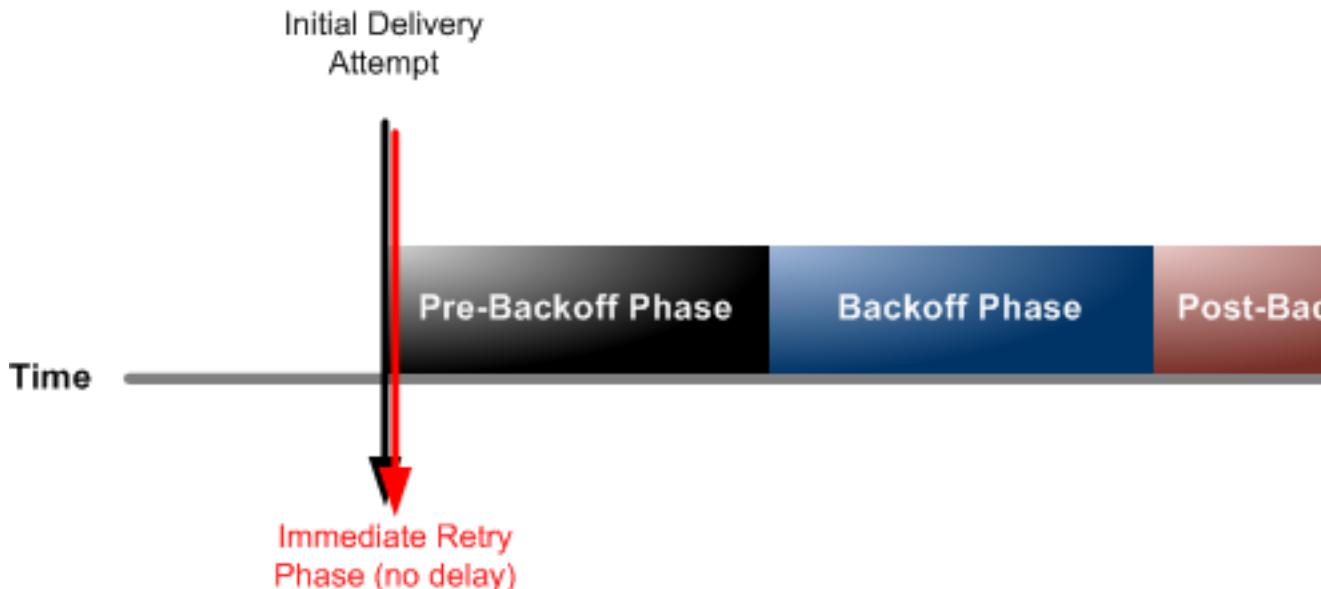
- Amazon SNS applies jittering to delivery retries. For more information, see the [Exponential Backoff and Jitter](#) post on the *AWS Architecture Blog*.

Endpoint type	Delivery protocols	Immediate retry (no delay) phase	Pre-backoff phase	Backoff phase	Post-backoff phase	Total attempts
AWS managed endpoints	Amazon Kinesis Data Firehose <sup>1</sup>	3 times, without delay	2 times, 1 second apart	10 times, with exponential backoff, from 1 second to 20 seconds	100,000 times, 20 seconds apart	100,015 times, over 23 days
	AWS Lambda					
	Amazon SQS					
Customer managed endpoints	SMTP	0 times, without delay	2 times, 10 seconds apart	10 times, with exponential backoff, from 10 seconds to 600 seconds (10 minutes)	38 times, 600 seconds (10 minutes) apart	50 attempts, over 6 hours
	SMS					
	Mobile push					

<sup>1</sup> For throttling errors with the Kinesis Data Firehose protocol, Amazon SNS uses the same delivery policy as for customer managed endpoints.

## Delivery policy stages

The following diagram shows the phases of a delivery policy.



Each delivery policy is comprised of four phases.

1. **Immediate Retry Phase (No Delay)** – This phase occurs immediately after the initial delivery attempt. There is no delay between retries in this phase.
2. **Pre-Backoff Phase** – This phase follows the Immediate Retry Phase. Amazon SNS uses this phase to attempt a set of retries before applying a backoff function. This phase specifies the number of retries and the amount of delay between them.
3. **Backoff Phase** – This phase controls the delay between retries by using the retry-backoff function. This phase sets a minimum delay, a maximum delay, and a retry-backoff function that defines how quickly the delay increases from the minimum to the maximum delay. The backoff function can be arithmetic, exponential, geometric, or linear.
4. **Post-Backoff Phase** – This phase follows the backoff phase. It specifies a number of retries and the amount of delay between them. This is the final phase.

## Creating an HTTP/S delivery policy

You can use a delivery policy and its four phases to define how Amazon SNS retries the delivery of messages to HTTP/S endpoints. Amazon SNS lets you override the default retry policy for HTTP endpoints when you might, for example, want to customize the policy based your HTTP server's capacity.

You can set your HTTP/S delivery policy as a JSON object at the subscription or topic level. When you define the policy at the topic level, it applies to all HTTP/S subscriptions associated with the topic.

You should customize your delivery policy according to your HTTP/S server's capacity. You can set the policy as a topic attribute or a subscription attribute. If all HTTP/S subscriptions in your topic target the same HTTP/S server, we recommend that you set the delivery policy as a topic attribute, so that it remains valid for all HTTP/S subscriptions in the topic. Otherwise, you must compose a delivery policy for each HTTP/S subscription in your topic, according the capacity of the HTTP/S server that the policy targets.

The following JSON object represents a delivery policy that instructs Amazon SNS to retry a failed HTTP/S delivery attempt, as follows:

1. 3 times immediately in the no-delay phase
2. 2 times (1 second apart) in the pre-backoff phase
3. 10 times (with exponential backoff from 1 second to 60 seconds)
4. 35 times (60 seconds apart) in the post-backoff phase

In this sample delivery policy, Amazon SNS makes a total of 50 attempts before discarding the message. To keep the message after the retries specified in the delivery policy are exhausted, configure your subscription to move undeliverables messages to a dead-letter queue (DLQ). For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#).

### Note

This delivery policy also instructs Amazon SNS to throttle deliveries to no more than 10 per second, using the `maxReceivesPerSecond` property. This self-throttling rate could result in more messages published (inbound traffic) than delivered (outbound traffic). When there's more inbound than outbound traffic, your subscription can accumulate a large message backlog, which might cause high message delivery latency. In your delivery policies, be sure to specify a value for `maxReceivesPerSecond` that doesn't adversely impact your workload.

```
{  
  "healthyRetryPolicy": {  
    "minDelayTarget": 1,  
    "maxDelayTarget": 60,  
    "numRetries": 50,  
    "numNoDelayRetries": 3,  
    "numMinDelayRetries": 2,  
  },  
  "maxReceivesPerSecond": 10,  
}
```

```

        "numMaxDelayRetries": 35,
        "backoffFunction": "exponential"
    },
    "throttlePolicy": {
        "maxReceivesPerSecond": 10
    }
}

```

The delivery policy is composed of a retry policy and a throttle policy. In total, there are eight attributes in a delivery policy.

Policy	Description	Constraint
<code>minDelayTarget</code>	The minimum delay for a retry. <b>Unit:</b> Seconds	1 to maximum delay <b>Default:</b> 20
<code>maxDelayTarget</code>	The maximum delay for a retry. <b>Unit:</b> Seconds	Minimum delay to 3,600 <b>Default:</b> 20
<code>numRetries</code>	The total number of retries, including immediate, pre-backoff, backoff, and post-backoff retries.	0 to 100 <b>Default:</b> 3
<code>numNoDelayRetries</code>	The number of retries to be done immediately, with no delay between them.	0 or greater <b>Default:</b> 0
<code>numMinDelayRetries</code>	The number of retries in the pre-backoff phase, with the specified minimum delay between them.	0 or greater <b>Default:</b> 0
<code>numMaxDelayRetries</code>	The number of retries in the post-backoff phase, with the maximum delay between them.	0 or greater <b>Default:</b> 0
<code>backoffFunction</code>	The model for backoff between retries.	One of four options: <ul style="list-style-type: none"> <li>• arithmetic</li> <li>• exponential</li> <li>• geometric</li> <li>• linear</li> </ul> <b>Default:</b> linear
<code>maxReceivesPerSecond</code>	The maximum number of deliveries per second, per subscription.	1 or greater <b>Default:</b> No throttling

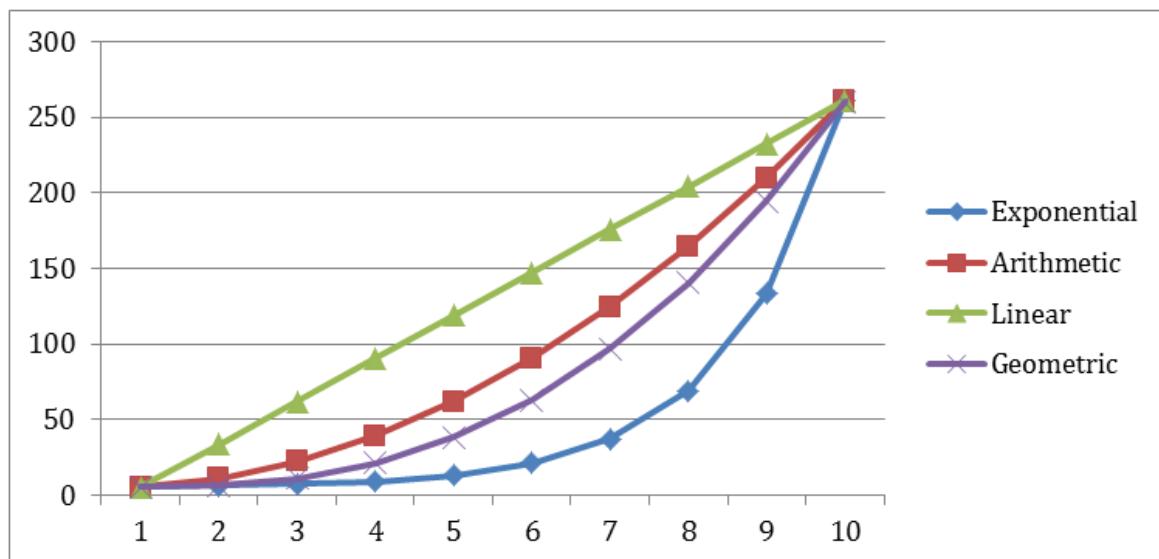
Amazon SNS uses the following formula to calculate the number of retries in the backoff phase:

```
numRetries - numNoDelayRetries - numMinDelayRetries - numMaxDelayRetries
```

You can use three parameters to control the frequency of retries in the backoff phase.

- `minDelayTarget` – Defines the delay associated with the first retry attempt in the backoff phase.
- `maxDelayTarget` – Defines the delay associated with the final retry attempt in the backoff phase.
- `backoffFunction` – Defines the algorithm that Amazon SNS uses to calculate the delays associated with all of the retry attempts between the first and last retries in the backoff phase. You can use one of four retry-backoff functions.

The following diagram shows how each retry backoff function affects the delay associated with retries during the backoff phase: A delivery policy with the total number of retries set to 10, the minimum delay set to 5 seconds, and the maximum delay set to 260 seconds. The vertical axis represents the delay in seconds associated with each of the 10 retries. The horizontal axis represents the number of retries, from the first to the tenth attempt.



## Amazon SNS dead-letter queues (DLQs)

A dead-letter queue is an Amazon SQS queue that an Amazon SNS subscription can target for messages that can't be delivered to subscribers successfully. Messages that can't be delivered due to client errors or server errors are held in the dead-letter queue for further analysis or reprocessing. For more information, see [Configuring an Amazon SNS dead-letter queue for a subscription \(p. 98\)](#) and [Amazon SNS message delivery retries \(p. 92\)](#).

### Note

- The Amazon SNS subscription and Amazon SQS queue must be under the same AWS account and Region.
- For a [FIFO topic \(p. 38\)](#), use an Amazon SQS FIFO queue as a dead-letter queue for the Amazon SNS subscription.
- To use an encrypted Amazon SQS queue as a dead-letter queue, you must use a custom CMK with a key policy that grants the Amazon SNS service principal access to AWS KMS API actions. For more information, see [Encryption at rest \(p. 344\)](#) in this guide and [Protecting Amazon SQS Data Using Server-Side Encryption \(SSE\) and AWS KMS](#) in the *Amazon Simple Queue Service Developer Guide*.

### Topics

- [Why do message deliveries fail? \(p. 97\)](#)

- [How do dead-letter queues work? \(p. 97\)](#)
- [How are messages moved into a dead-letter queue? \(p. 97\)](#)
- [How can I move messages out of a dead-letter queue? \(p. 98\)](#)
- [How can I monitor and log dead-letter queues? \(p. 98\)](#)
- [Configuring an Amazon SNS dead-letter queue for a subscription \(p. 98\)](#)

## Why do message deliveries fail?

In general, message delivery fails when Amazon SNS can't access a subscribed endpoint due to a *client-side* or *server-side error*. When Amazon SNS receives a client-side error, or continues to receive a server-side error for a message beyond the number of retries specified by the corresponding retry policy, Amazon SNS discards the message—unless a dead-letter queue is attached to the subscription. Failed deliveries don't change the status of your subscriptions. For more information, see [Amazon SNS message delivery retries \(p. 92\)](#).

### Client-side errors

Client-side errors can happen when Amazon SNS has stale subscription metadata. These errors commonly occur when an owner deletes the endpoint (for example, a Lambda function subscribed to an Amazon SNS topic) or when an owner changes the policy attached to the subscribed endpoint in a way that prevents Amazon SNS from delivering messages to the endpoint. Amazon SNS doesn't retry the message delivery that fails as a result of a client-side error.

### Server-side errors

Server-side errors can happen when the system responsible for the subscribed endpoint becomes unavailable or returns an exception that indicates that it can't process a valid request from Amazon SNS. When server-side errors occur, Amazon SNS retries the failed deliveries using either a linear or exponential backoff function. For server-side errors caused by AWS managed endpoints backed by Amazon SQS or AWS Lambda, Amazon SNS retries delivery up to 100,015 times, over 23 days.

Customer managed endpoints (such as HTTP, SMTP, SMS, or mobile push) can also cause server-side errors. Amazon SNS retries delivery to these types of endpoints as well. While HTTP endpoints support customer-defined retry policies, Amazon SNS sets an internal delivery retry policy to 50 times over 6 hours, for SMTP, SMS, and mobile push endpoints.

## How do dead-letter queues work?

A dead-letter queue is attached to an Amazon SNS subscription (rather than a topic) because message deliveries happen at the subscription level. This lets you identify the original target endpoint for each message more easily.

A dead-letter queue associated with an Amazon SNS subscription is an ordinary Amazon SQS queue. For more information about the message retention period, see [Quotas Related to Messages](#) in the *Amazon Simple Queue Service Developer Guide*. You can change the message retention period using the Amazon SQS [SetQueueAttributes](#) API action. To make your applications more resilient, we recommend setting the maximum retention period for dead-letter queues to 14 days.

## How are messages moved into a dead-letter queue?

Your messages are moved into a dead-letter queue using a *redrive policy*. A redrive policy is a JSON object that refers to the ARN of the dead-letter queue. The `deadLetterTargetArn` attribute specifies

the ARN. The ARN must point to an Amazon SQS queue in the same AWS account and Region as your Amazon SNS subscription. For more information, see [Configuring an Amazon SNS dead-letter queue for a subscription \(p. 98\)](#).

**Note**

For a [FIFO topic \(p. 38\)](#), use an Amazon SQS FIFO queue as a dead-letter queue for the Amazon SNS subscription.

The following JSON object is a sample redrive policy, attached to an SNS subscription.

```
{  
    "deadLetterTargetArn": "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue"  
}
```

## How can I move messages out of a dead-letter queue?

You can move messages out of a dead-letter queue in two ways:

- **Avoid writing Amazon SQS consumer logic** – Set your dead-letter queue as an event source to the Lambda function to drain your dead-letter queue.
- **Write Amazon SQS consumer logic** – Use the Amazon SQS API, AWS SDK, or AWS CLI to write custom consumer logic for polling, processing, and deleting the messages in the dead-letter queue.

## How can I monitor and log dead-letter queues?

You can use Amazon CloudWatch metrics to monitor dead-letter queues associated with your Amazon SNS subscriptions. All Amazon SQS queues emit CloudWatch metrics at one-minute intervals. For more information, see [Available CloudWatch Metrics for Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*. All Amazon SNS subscriptions with dead-letter queues also emit CloudWatch metrics. For more information, see [Monitoring Amazon SNS topics using CloudWatch \(p. 390\)](#).

To be notified of activity in your dead-letter queues, you can use CloudWatch metrics and alarms. For example, when you expect the dead-letter queue to be always empty, you can create a CloudWatch alarm for the `NumberOfMessagesSent` metric. You can set the alarm threshold to 0 and specify an Amazon SNS topic to be notified when the alarm goes off. This Amazon SNS topic can deliver your alarm notification to any endpoint type (such as an email address, phone number, or mobile pager app).

You can use CloudWatch Logs to investigate the exceptions that cause any Amazon SNS deliveries to fail and for messages to be sent to dead-letter queues. Amazon SNS can log both successful and failed deliveries in CloudWatch. For more information, see [Amazon SNS message delivery status \(p. 87\)](#).

## Configuring an Amazon SNS dead-letter queue for a subscription

A dead-letter queue is an Amazon SQS queue that an Amazon SNS subscription can target for messages that can't be delivered to subscribers successfully. Messages that can't be delivered due to client errors or server errors are held in the dead-letter queue for further analysis or reprocessing. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#) and [Amazon SNS message delivery retries \(p. 92\)](#).

This page shows how you can use the AWS Management Console, an AWS SDK, the AWS CLI, and AWS CloudFormation to configure a dead-letter queue for an Amazon SNS subscription.

## Prerequisites

Before you configure a dead-letter queue, complete the following prerequisites:

1. [Create an Amazon SNS topic \(p. 24\)](#) named `MyTopic`.
2. [Create an Amazon SQS queue](#) named `MyEndpoint`, to be used as the endpoint for the Amazon SNS subscription.
3. (Skip for AWS CloudFormation) [Subscribe the queue to the topic \(p. 124\)](#).
4. [Create another Amazon SQS queue](#) named `MyDeadLetterQueue`, to be used as the dead-letter queue for the Amazon SNS subscription.
5. To give Amazon SNS principal access to the Amazon SQS API action, set the following queue policy for `MyDeadLetterQueue`.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "sns.amazonaws.com"  
      },  
      "Action": "SQS:SendMessage",  
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue",  
      "Condition": {  
        "ArnEquals": {  
          "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:MyTopic"  
        }  
      }  
    }  
  ]  
}
```

### Topics

- [To configure a dead-letter queue for an Amazon SNS subscription using the AWS Management Console \(p. 99\)](#)
- [To configure a dead-letter queue for an Amazon SNS subscription using an AWS SDK \(p. 100\)](#)
- [To configure a dead-letter queue for an Amazon SNS subscription using the AWS CLI \(p. 100\)](#)
- [To configure a dead-letter queue for an Amazon SNS subscription using AWS CloudFormation \(p. 101\)](#)

## To configure a dead-letter queue for an Amazon SNS subscription using the AWS Management Console

Before you begin this tutorial, make sure you complete the [prerequisites \(p. 99\)](#).

1. Sign in to the [Amazon SQS console](#).
2. [Create an Amazon SQS queue](#) or use an existing queue and note the ARN of the queue on the **Details** tab of the queue, for example:

```
arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue
```

### Note

For a [FIFO topic \(p. 38\)](#), use an Amazon SQS FIFO queue as a dead-letter queue for the Amazon SNS subscription.

3. Sign in to the [Amazon SNS console](#).

4. On the navigation panel, choose **Subscriptions**.
5. On the **Subscriptions** page, select an existing subscription and then choose **Edit**.
6. On the **Edit 1234a567-bc89-012d-3e45-6fg7h890123i** page, expand the **Redrive policy (dead-letter queue)** section, and then do the following:
  - a. Choose **Enabled**.
  - b. Specify the ARN of an Amazon SQS queue.
7. Choose **Save changes**.

Your subscription is configured to use a dead-letter queue.

## To configure a dead-letter queue for an Amazon SNS subscription using an AWS SDK

Before you run this example, make sure that you complete the [prerequisites \(p. 99\)](#).

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code example shows how to set an Amazon SQS queue as a dead-letter queue for an Amazon SNS subscription.

Java

### SDK for Java 1.x

```
// Specify the ARN of the Amazon SNS subscription.  
String subscriptionArn =  
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-  
bc89-012d-3e45-6fg7h890123i";  
  
// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.  
String redrivePolicy =  
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-  
east-2:123456789012:MyDeadLetterQueue\"}";  
  
// Set the specified Amazon SQS queue as a dead-letter queue  
// of the specified Amazon SNS subscription by setting the RedrivePolicy attribute.  
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()  
    .withSubscriptionArn(subscriptionArn)  
    .withAttributeName("RedrivePolicy")  
    .withAttributeValue(redrivePolicy);  
sns.setSubscriptionAttributes(request);
```

- Find instructions and more code on [GitHub](#).

## To configure a dead-letter queue for an Amazon SNS subscription using the AWS CLI

Before you begin this tutorial, make sure you complete the [prerequisites \(p. 99\)](#).

1. Install and configure the AWS CLI. For more information, see the [AWS Command Line Interface User Guide](#).
2. Use the following command.

```
aws sns set-subscription-attributes \
--subscription-arn arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i
--attribute-name RedrivePolicy
--attribute-value "{\"deadLetterTargetArn\": \"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}"
```

## To configure a dead-letter queue for an Amazon SNS subscription using AWS CloudFormation

Before you begin this tutorial, make sure you complete the [prerequisites \(p. 99\)](#).

1. Copy the following JSON code to a file named `MyDeadLetterQueue.json`.

```
{
  "Resources": {
    "mySubscription": {
      "Type" : "AWS::SNS::Subscription",
      "Properties" : {
        "Protocol": "sq",
        "Endpoint": "arn:aws:sqs:us-east-2:123456789012:MyEndpoint",
        "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",
        "RedrivePolicy": {
          "deadLetterTargetArn":
            "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue"
        }
      }
    }
  }
}
```

2. Sign in to the [AWS CloudFormation console](#).
3. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose your `MyDeadLetterQueue.json` file, and then choose **Next**.
4. On the **Specify Details** page, enter `MyDeadLetterQueue` for **Stack Name**, and then choose **Next**.
5. On the **Options** page, choose **Next**.
6. On the **Review** page, choose **Create**.

AWS CloudFormation begins to create the `MyDeadLetterQueue` stack and displays the **CREATE\_IN\_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

# Amazon SNS message archiving and analytics

Amazon SNS provides message archiving and analytics through Amazon Kinesis Data Firehose. You can fan out notifications to Kinesis Data Firehose delivery streams, which allows you to send notifications to storage and analytics destinations that Kinesis Data Firehose supports, including Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and more. For information, see the following pages:

- [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#)
- [Example use case for message archiving and analytics \(p. 114\)](#)
- [Working with delivery stream destinations \(p. 105\)](#)

# Using Amazon SNS for application-to-application (A2A) messaging

This section provides information about using Amazon SNS for application-to-application messaging with subscribers.

## Topics

- [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#)
- [Fanout to Lambda functions \(p. 123\)](#)
- [Fanout to Amazon SQS queues \(p. 124\)](#)
- [Fanout to HTTP/S endpoints \(p. 134\)](#)
- [Fanout to AWS Event Fork Pipelines \(p. 148\)](#)

## Fanout to Kinesis Data Firehose delivery streams

You can subscribe [Amazon Kinesis Data Firehose delivery streams](#) to SNS topics, which allows you to send notifications to additional storage and analytics endpoints. Messages published to an SNS topic are sent to the subscribed Kinesis Data Firehose delivery stream, and delivered to the destination as configured in Kinesis Data Firehose. A subscription owner can subscribe up to five Kinesis Data Firehose delivery streams to an SNS topic.

Through Kinesis Data Firehose delivery streams, you can fan out Amazon SNS notifications to Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service (OpenSearch Service), and to third-party service providers such as Datadog, New Relic, MongoDB, and Splunk.

For example, you can use this functionality to permanently store messages sent to a topic in an Amazon S3 bucket for compliance, archival, or other purposes. To do this, create a Kinesis Data Firehose delivery stream with an S3 bucket destination, and subscribe that delivery stream to the SNS topic. As another example, to perform analysis on messages sent to an SNS topic, create a delivery stream with an OpenSearch Service index destination. Then subscribe the Kinesis Data Firehose delivery stream to the SNS topic.

Amazon SNS also supports message delivery status logging for notifications sent to Kinesis Data Firehose endpoints. For more information, see [Amazon SNS message delivery status \(p. 87\)](#).

## Topics

- [Prerequisites for subscribing Kinesis Data Firehose delivery streams to Amazon SNS topics \(p. 103\)](#)
- [Subscribing a Kinesis Data Firehose delivery stream to an Amazon SNS topic \(p. 105\)](#)
- [Working with delivery stream destinations \(p. 105\)](#)
- [Example use case for message archiving and analytics \(p. 114\)](#)

## Prerequisites for subscribing Kinesis Data Firehose delivery streams to Amazon SNS topics

To subscribe an Amazon Kinesis Data Firehose delivery stream to an SNS topic, your AWS account must have:

- A standard SNS topic. For more information, see [Creating an Amazon SNS topic \(p. 24\)](#).
- A Kinesis Data Firehose delivery stream. For more information, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#) and [Grant Your Application Access to Your Kinesis Data Firehose Resources](#) in the *Amazon Kinesis Data Firehose Developer Guide*.
- An AWS Identity and Access Management (IAM) role that trusts the Amazon SNS service principal and has permission to write to the delivery stream. You'll enter this role's Amazon Resource Name (ARN) as the `SubscriptionRoleARN` when you create the subscription. Amazon SNS assumes this role, which allows Amazon SNS to put records in the Kinesis Data Firehose delivery stream.

The following example policy shows the recommended permissions:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "firehose:DescribeDeliveryStream",  
                "firehose>ListDeliveryStreams",  
                "firehose>ListTagsForDeliveryStream",  
                "firehose:PutRecord",  
                "firehose:PutRecordBatch"  
            ],  
            "Resource": [  
                "arn:aws:firehose:us-east-1:111111111111:deliverystream/firehose-sns-delivery-  
                stream"  
            ],  
            "Effect": "Allow"  
        }  
    ]  
}
```

To provide full permission for using Kinesis Data Firehose, you can also use the AWS managed policy `AmazonKinesisFirehoseFullAccess`. Or, to provide stricter permissions for using Kinesis Data Firehose, you can create your own policy. At minimum, the policy must provide permission to run the `PutRecord` operation on a specific delivery stream.

In all cases, you must also edit the trust relationship to include the Amazon SNS service principal. For example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "sns.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

For more information on creating roles, see [Creating a role to delegate permissions to an AWS service in the IAM User Guide](#).

After you've completed these requirements, you can [subscribe the delivery stream to the SNS topic \(p. 105\)](#).

## Subscribing a Kinesis Data Firehose delivery stream to an Amazon SNS topic

To deliver Amazon SNS notifications to [Amazon Kinesis Data Firehose delivery streams \(p. 103\)](#), first make sure that you've addressed all the [prerequisites \(p. 103\)](#).

### To subscribe a Kinesis Data Firehose delivery stream to a topic

1. Sign in to the [Amazon SNS console](#).
2. In the navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:
  - a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a standard topic.
  - b. For **Protocol**, choose **Kinesis Data Firehose**.
  - c. For **Endpoint**, choose the ARN of a Kinesis Data Firehose delivery stream that can receive notifications from Amazon SNS.
  - d. For **Subscription role ARN**, specify the ARN of the AWS Identity and Access Management (IAM) role that you created for writing to Kinesis Data Firehose delivery streams. For more information, see [Prerequisites for subscribing Kinesis Data Firehose delivery streams to Amazon SNS topics \(p. 103\)](#).
  - e. (Optional) To remove any Amazon SNS metadata from published messages, choose **Enable raw message delivery**. For more information, see [Amazon SNS raw message delivery \(p. 83\)](#).
5. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see [Amazon SNS subscription filter policies \(p. 71\)](#).
6. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#).
7. Choose **Create subscription**.

The console creates the subscription and opens the subscription's **Details** page.

## Working with delivery stream destinations

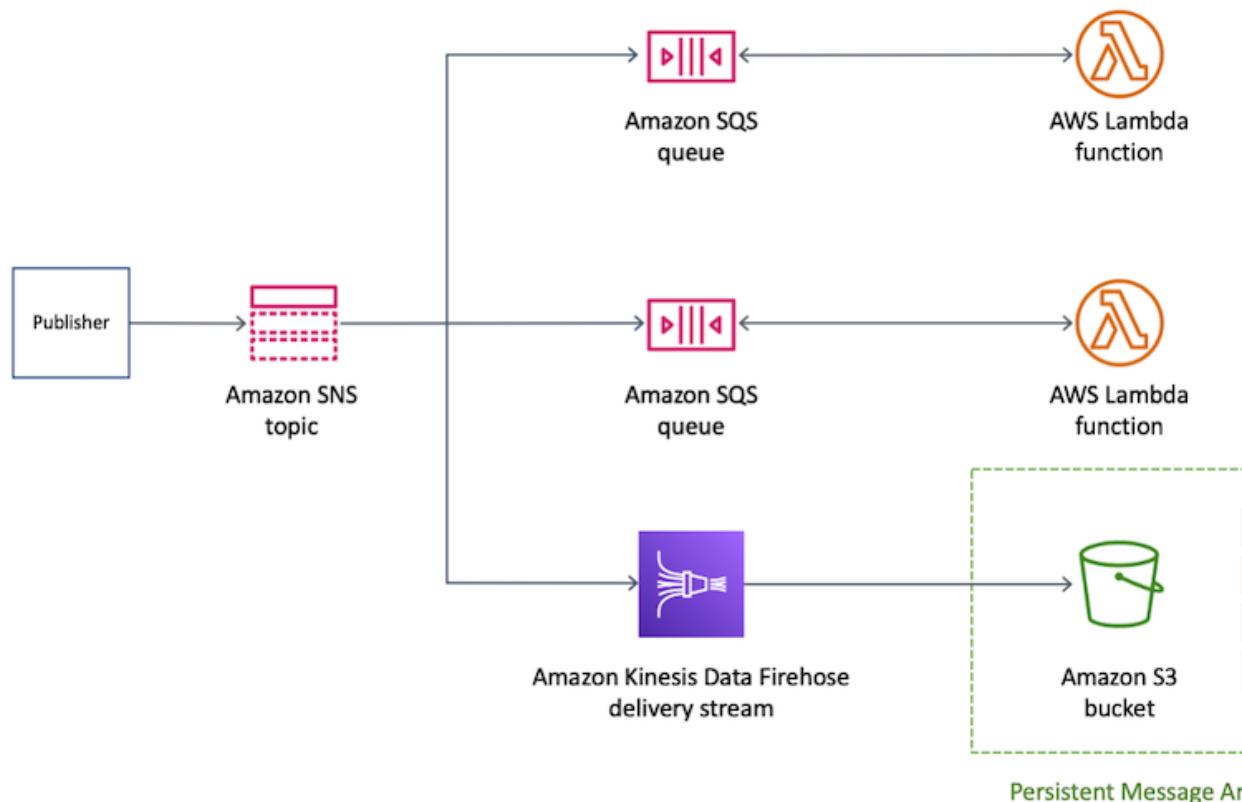
Through [Amazon Kinesis Data Firehose delivery streams \(p. 103\)](#), you can send messages to additional endpoints. This section describes how to work with supported destinations.

### Topics

- [Amazon S3 destinations \(p. 105\)](#)
- [OpenSearch Service destinations \(p. 108\)](#)
- [Amazon Redshift destinations \(p. 110\)](#)
- [HTTP destinations \(p. 113\)](#)

## Amazon S3 destinations

This section provides information about Amazon Kinesis Data Firehose delivery streams that publish data to Amazon Simple Storage Service (Amazon S3).



### Topics

- [Archived message format for Amazon S3 destinations \(p. 106\)](#)
- [Analyzing messages for Amazon S3 destinations \(p. 107\)](#)

### Archived message format for Amazon S3 destinations

The following example shows an Amazon SNS notification sent to an Amazon Simple Storage Service (Amazon S3) bucket, using indents for readability.

#### Note

In this example, raw message delivery is disabled for the published message. When raw message delivery is disabled, Amazon SNS adds JSON metadata to the message, including these properties:

- Type
- MessageId
- TopicArn
- Subject
- Timestamp
- UnsubscribeURL
- MessageAttributes

For more information about raw delivery, see [Amazon SNS raw message delivery \(p. 83\)](#).

```
{
    "Type": "Notification",
    "MessageId": "719a6bbf-f51b-5320-920f-3385b5e9aa56",
    "TopicArn": "arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic",
    "Subject": "My 1st subject",
    "Message": "My 1st body",
    "Timestamp": "2020-11-26T23:48:02.032Z",
    "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5",
    "MessageAttributes": {
        "myKey1": {
            "Type": "String",
            "Value": "myValue1"
        },
        "myKey2": {
            "Type": "String",
            "Value": "myValue2"
        }
    }
}
```

The following example shows three SNS messages sent through an Amazon Kinesis Data Firehose delivery stream to the same Amazon S3 bucket. Buffering is taken into account, and line breaks separate the messages.

```
{"Type":"Notification","MessageId":"d7d2513e-6126-5d77-bbe2-09042bd0a03a","TopicArn":"arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic","Subject":"My 1st subject","Message":"My 1st body","Timestamp":"2020-11-27T00:30:46.100Z","UnsubscribeURL":"https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5","MessageAttributes": {"myKey1": {"Type": "String", "Value": "myValue1"}, "myKey2": {"Type": "String", "Value": "myValue2"}}}
{"Type": "Notification", "MessageId": "0c0696ab-7733-5bfb-b6db-ce913c294d56", "TopicArn": "arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic", "Subject": "My 2nd subject", "Message": "My 2nd body", "Timestamp": "2020-11-27T00:31:22.151Z", "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5", "MessageAttributes": {"myKey1": {"Type": "String", "Value": "myValue1"}}, {"Type": "Notification", "MessageId": "816cd54d-8cfa-58ad-91c9-8d77c7d173aa", "TopicArn": "arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic", "Subject": "My 3rd subject", "Message": "My 3rd body", "Timestamp": "2020-11-27T00:31:39.755Z", "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5"}
```

## Analyzing messages for Amazon S3 destinations

This page describes how to analyze Amazon SNS messages sent through Amazon Kinesis Data Firehose delivery streams to Amazon Simple Storage Service (Amazon S3) destinations.

### To analyze SNS messages sent through Kinesis Data Firehose delivery streams to Amazon S3 destinations

1. Configure your Amazon S3 resources. For instructions, see [Creating a bucket](#) in the *Amazon Simple Storage Service User Guide* and [Working with Amazon S3 Buckets](#) in the *Amazon Simple Storage Service User Guide*.
2. Configure your delivery stream. For instructions, see [Choose Amazon S3 for Your Destination](#) in the *Amazon Kinesis Data Firehose Developer Guide*.

3. Use [Amazon Athena](#) to query the Amazon S3 objects using standard SQL. For more information, see [Getting Started](#) in the *Amazon Athena User Guide*.

### Example query

For this example query, assume the following:

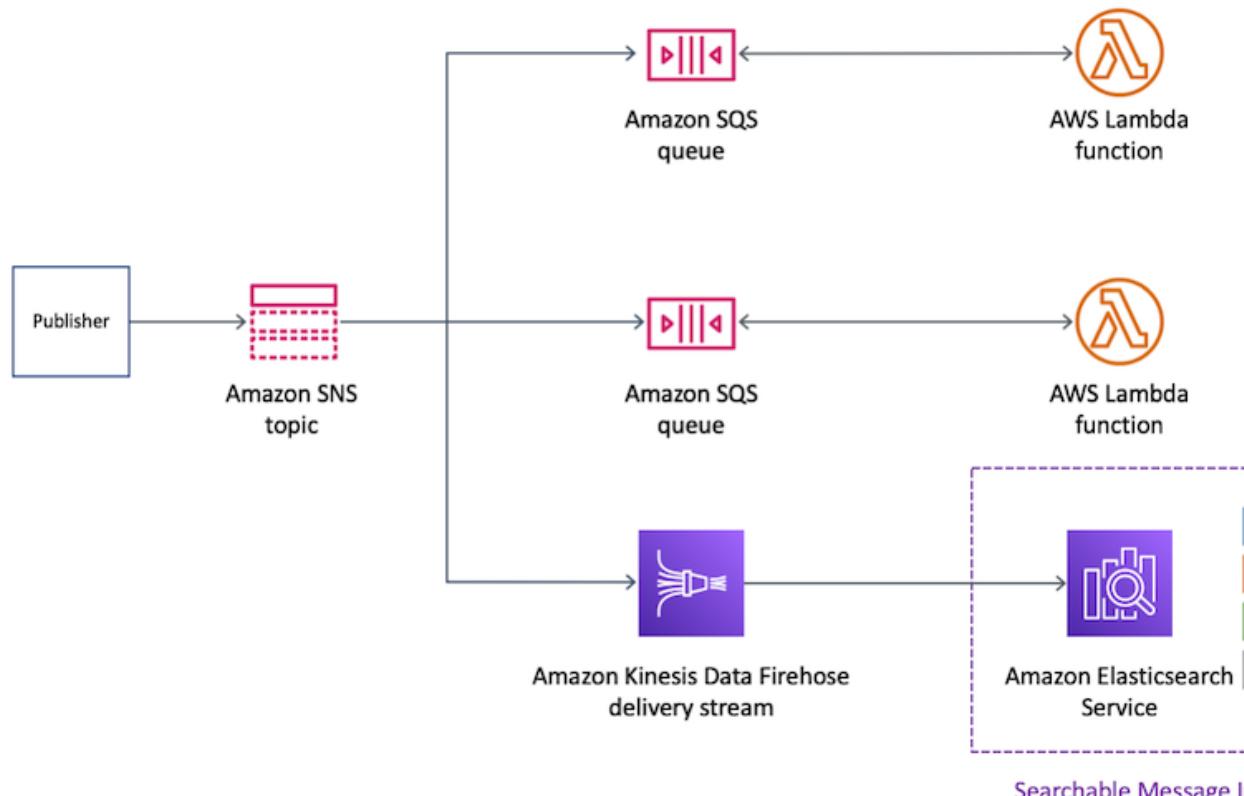
- Messages are stored in the notifications table in the default schema.
- The notifications table includes a timestamp column with a type of string.

The following query returns all SNS messages received in the specified date range:

```
SELECT *  
FROM default.notifications  
WHERE from_iso8601_timestamp(timestamp) BETWEEN TIMESTAMP '2020-12-01 00:00:00' AND  
      TIMESTAMP '2020-12-02 00:00:00';
```

## OpenSearch Service destinations

This section provides information about Amazon Kinesis Data Firehose delivery streams that publish data to Amazon OpenSearch Service (OpenSearch Service).



### Topics

- [Archived message format in OpenSearch Service indices \(p. 109\)](#)
- [Analyzing messages for OpenSearch Service destinations \(p. 109\)](#)

## Archived message format in OpenSearch Service indices

The following example shows an Amazon SNS notification sent to an Amazon OpenSearch Service (OpenSearch Service) index named `my-index`. This index has a time filter field on the `Timestamp` field. The SNS notification is placed in the `_source` property of the payload.

### Note

In this example, raw message delivery is disabled for the published message. When raw message delivery is disabled, Amazon SNS adds JSON metadata to the message, including these properties:

- `Type`
- `MessageId`
- `TopicArn`
- `Subject`
- `Timestamp`
- `UnsubscribeURL`
- `MessageAttributes`

For more information about raw delivery, see [Amazon SNS raw message delivery \(p. 83\)](#).

```
{  
    "_index": "my-index",  
    "_type": "_doc",  
    "_id": "49613100963111323203250405402193283794773886550985932802.0",  
    "_version": 1,  
    "_score": null,  
    "_source": {  
        "Type": "Notification",  
        "MessageId": "bf32e294-46e3-5dd5-a6b3-bad65162e136",  
        "TopicArn": "arn:aws:sns:us-east-1:111111111111:my-topic",  
        "Subject": "Sample subject",  
        "Message": "Sample message",  
        "Timestamp": "2020-12-02T22:29:21.189Z",  
        "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?  
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-  
topic:b5aa9bc1-9c3d-452b-b402-aca2cefc63c9",  
        "MessageAttributes": {  
            "my_attribute": {  
                "Type": "String",  
                "Value": "my_value"  
            }  
        },  
        "fields": {  
            "Timestamp": [  
                "2020-12-02T22:29:21.189Z"  
            ]  
        },  
        "sort": [  
            1606948161189  
        ]  
    }  
}
```

## Analyzing messages for OpenSearch Service destinations

This page describes how to analyze Amazon SNS messages sent through Amazon Kinesis Data Firehose delivery streams to Amazon OpenSearch Service (OpenSearch Service) destinations.

## To analyze SNS messages sent through Kinesis Data Firehose delivery streams to OpenSearch Service destinations

1. Configure your OpenSearch Service resources. For instructions, see [Getting Started with Amazon OpenSearch Service](#) in the *Amazon OpenSearch Service Developer Guide*.
2. Configure your delivery stream. For instructions, see [Choose OpenSearch Service for Your Destination](#) in the *Amazon Kinesis Data Firehose Developer Guide*.
3. Run a query using OpenSearch Service queries and Kibana. For more information, see [Step 3: Search Documents in an OpenSearch Service Domain](#) and [Kibana](#) in the *Amazon OpenSearch Service Developer Guide*.

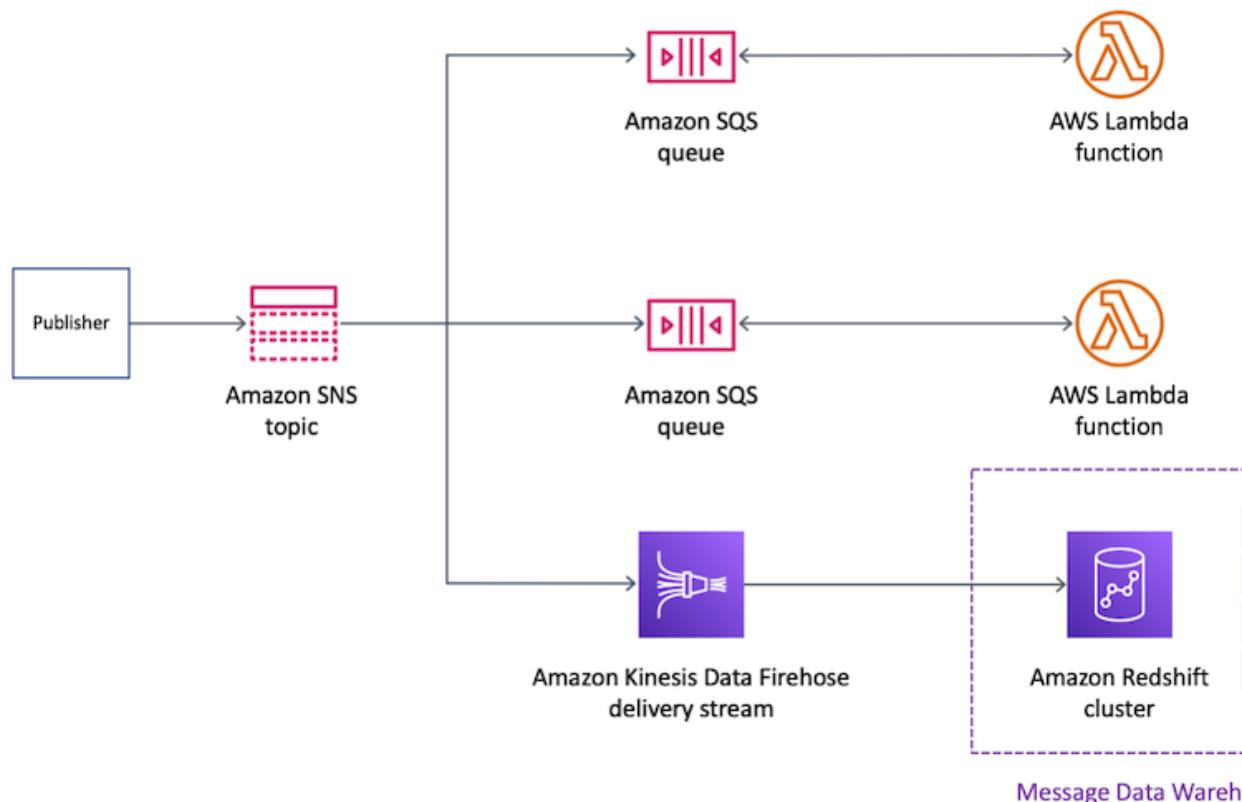
### Example query

The following example queries the `my-index` index for all SNS messages received in the specified date range:

```
POST https://search-my-domain.us-east-1.es.amazonaws.com/my-index/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "Timestamp": {
              "gte": "2020-12-08T00:00:00.000Z",
              "lte": "2020-12-09T00:00:00.000Z",
              "format": "strict_date_optional_time"
            }
          }
        }
      ]
    }
  }
}
```

## Amazon Redshift destinations

This section describes how to fan out Amazon SNS notifications to an Amazon Kinesis Data Firehose delivery stream that publishes data to Amazon Redshift. With this configuration, you can connect to the Amazon Redshift database and use a SQL query tool to query the database for Amazon SNS messages that meet certain criteria.



### Topics

- [Archive table structure for Amazon Redshift destinations \(p. 111\)](#)
- [Analyzing messages for Amazon Redshift destinations \(p. 112\)](#)

### [Archive table structure for Amazon Redshift destinations](#)

For Amazon Redshift endpoints, published Amazon SNS messages are archived as rows in a table. The following is an example.

#### Note

In this example, raw message delivery is disabled for the published message. When raw message delivery is disabled, Amazon SNS adds JSON metadata to the message, including these properties:

- Type
- MessageId
- TopicArn
- Subject
- Message
- Timestamp
- UnsubscribeURL
- MessageAttributes

For more information about raw delivery, see [Amazon SNS raw message delivery \(p. 83\)](#).

Although Amazon SNS adds properties to the message using the capitalization shown in this list, column names in Amazon Redshift tables appear in all lowercase characters. To transform the JSON metadata for the Amazon Redshift endpoint, you can use the SQL `COPY` command. For more information, see [Copy from JSON examples](#) and [Load from JSON data using the 'auto ignorecase' option in the Amazon Redshift Database Developer Guide](#).

<b>type</b>	<b>messageid</b>	<b>topicarn</b>	<b>subject</b>	<b>message</b>	<b>timestamp</b>	<b>unsubscribe</b>	<b>messageattributes</b>
Notification	ea544832-a0d8-581d-9251-108245111101:1	arn:aws:sns:us-east-1:111111111111:my-topic	Sample message	Sample message	2020-12-02T00:18:37Z	2020-12-02T00:18:32.272Z	{"my_attribute": "my_value"} {"Type": "String"} Action=Unsubscribe SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deefbcbf4-45da-b92b-ca77a247813b
Notification	ab124832-a0d8-581d-9251-108245111101:2	arn:aws:sns:us-east-1:111111111111:my-topic	Sample message 2	Sample message 2	2020-12-03T00:18:41Z	2020-12-03T00:18:41.129Z	{"my_attribute2": "my_value2"} {"Type": "String"} Action=Unsubscribe SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deefbcbf4-45da-b92b-ca77a247813b
Notification	ce644832-a0d8-581d-9251-108245111101:3	arn:aws:sns:us-east-1:111111111111:my-topic	Sample message 3	Sample message 3	2020-12-09T00:08:44.405Z	2020-12-09T00:08:44.405Z	{"my_attribute3": "my_value3"} {"Type": "String"} Action=Unsubscribe SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deefbcbf4-45da-b92b-ca77a247813b

For more information about fanning out notifications to Amazon Redshift endpoints, see [Amazon Redshift destinations \(p. 110\)](#).

## Analyzing messages for Amazon Redshift destinations

This page describes how to analyze Amazon SNS messages sent through Amazon Kinesis Data Firehose delivery streams to Amazon Redshift destinations.

### To analyze SNS messages sent through Kinesis Data Firehose delivery streams to Amazon Redshift destinations

1. Configure your Amazon Redshift resources. For instructions, see [Getting started with Amazon Redshift](#) in the [Amazon Redshift Getting Started Guide](#).
2. Configure your delivery stream. For instructions, see [Choose Amazon Redshift for Your Destination](#) in the [Amazon Kinesis Data Firehose Developer Guide](#).
3. Run a query. For more information, see [Querying a database using the query editor](#) in the [Amazon Redshift Cluster Management Guide](#).

## Example query

For this example query, assume the following:

- Messages are stored in the notifications table in the default public schema.
- The `Timestamp` property from the SNS message is stored in the table's `timestamp` column with a column data type of `timestamptz`.

### Note

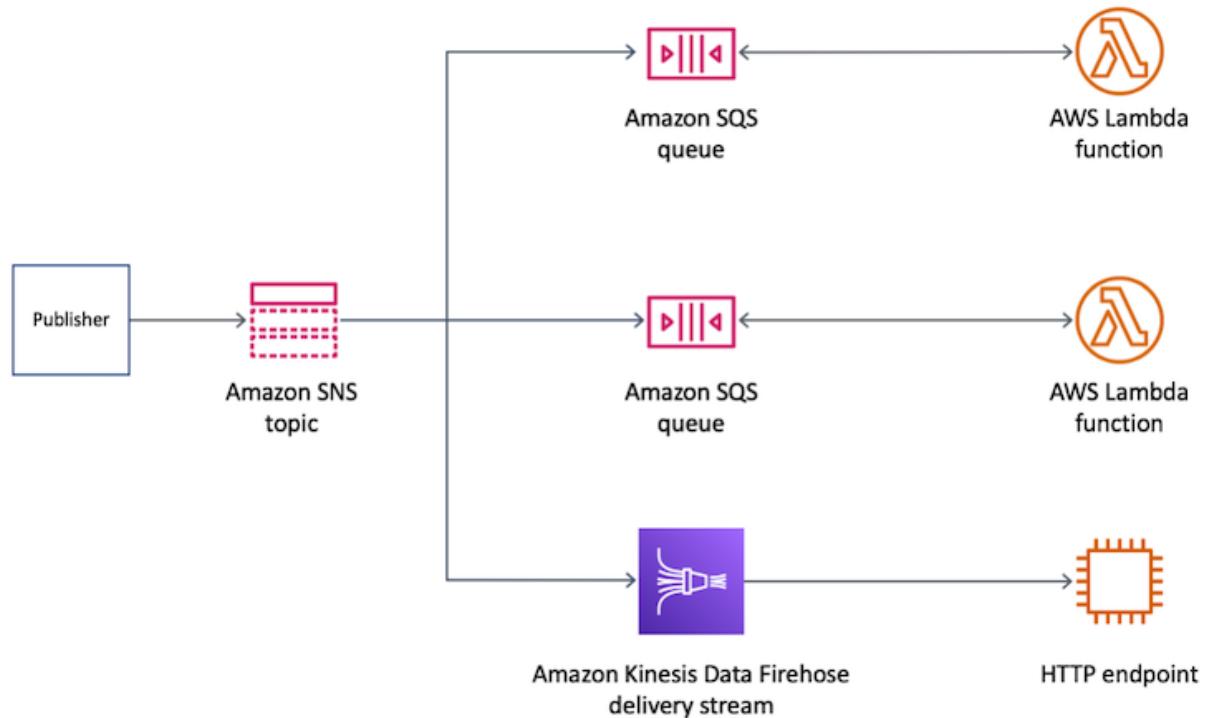
To transform the JSON metadata for the Amazon Redshift endpoint, you can use the SQL `COPY` command. For more information, see [Copy from JSON examples](#) and [Load from JSON data using the 'auto ignorecase' option](#) in the *Amazon Redshift Database Developer Guide*.

The following query returns all SNS messages received in the specified date range:

```
SELECT *  
FROM public.notifications  
WHERE timestamp > '2020-12-01T09:00:00.000Z' AND timestamp < '2020-12-02T09:00:00.000Z';
```

## HTTP destinations

This section provides information about Amazon Kinesis Data Firehose delivery streams that publish data to HTTP endpoints.



## Topics

- [Delivered message format for HTTP destinations \(p. 114\)](#)

## Delivered message format for HTTP destinations

The following is an example HTTP POST request body from Amazon SNS that an Amazon Kinesis Data Firehose delivery stream can send to the HTTP endpoint. The SNS notification is encoded as a base64 payload in the `records` property.

### Note

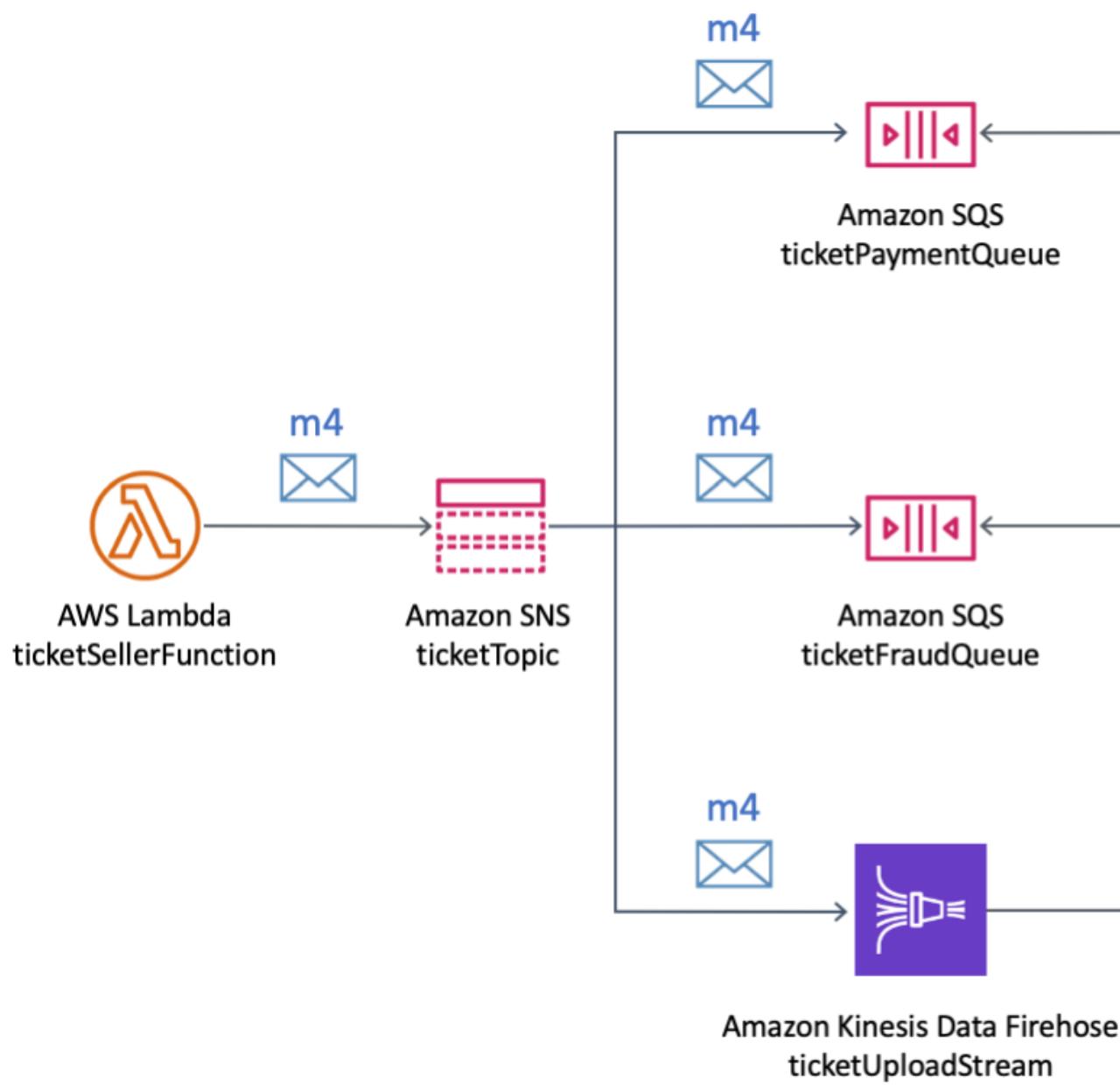
In this example, raw message delivery is disabled for the published message. For more information about raw delivery, see [Amazon SNS raw message delivery \(p. 83\)](#).

```
"body": {
    "requestId": "ebc9e8b2-fce3-4aef-a8f1-71698bf8175f",
    "timestamp": 1606255960435,
    "records": [
        {
            "data":
"eyJUeXB1IjoiTm90aWZpY2F0aW9uIiwitWVzc2FnZUlkJoiMjFkMmUzOGQtMmNhYi01ZjYxLTliYTItYmJiYWfhYzg0MGY2Iiwi"
        }
    ]
}
```

## Example use case for message archiving and analytics

This section provides a tutorial of a common use case for archiving and analyzing Amazon SNS messages.

The setting of this use case is an airline ticketing platform that operates in a regulated environment. The platform is subject to a compliance framework that requires the company to archive all ticket sales for at least five years. To meet the compliance goal on data retention, the company subscribes an Amazon Kinesis Data Firehose delivery stream to an existing SNS topic. The destination for the delivery stream is an Amazon Simple Storage Service (Amazon S3) bucket. With this configuration, all events published to the SNS topic are archived in the Amazon S3 bucket. The following diagram shows the architecture of this configuration:



To run analytics and gain insights on ticket sales, the company runs SQL queries using Amazon Athena. For example, the company can query to learn about the most popular destinations and the most frequent flyers.

To create the AWS resources for this use case, you can use the AWS Management Console or an AWS CloudFormation template.

## Topics

- [Creating the initial resources \(p. 116\)](#)
- [Creating the Kinesis Data Firehose delivery stream \(p. 117\)](#)
- [Subscribing the Kinesis Data Firehose delivery stream to the Amazon SNS topic \(p. 118\)](#)
- [Testing and querying the configuration \(p. 119\)](#)
- [Using an AWS CloudFormation template \(p. 121\)](#)

## Creating the initial resources

This page describes how to create the following resources for the [message archiving and analytics example use case \(p. 114\)](#):

- An Amazon Simple Storage Service (Amazon S3) bucket
- Two Amazon Simple Queue Service (Amazon SQS) queues
- An Amazon SNS topic
- Two Amazon SQS subscriptions to the Amazon SNS topic

### To create the initial resources

1. Create the Amazon S3 bucket:
  - a. Open the [Amazon S3 console](#).
  - b. Choose **Create bucket**.
  - c. For **Bucket name**, enter a globally unique name. Keep the other fields as the defaults.
  - d. Choose **Create bucket**.

For more information about Amazon S3 buckets, see [Creating a bucket](#) in the *Amazon Simple Storage Service User Guide* and [Working with Amazon S3 Buckets](#) in the *Amazon Simple Storage Service User Guide*.

2. Create the two Amazon SQS queues:
  - a. Open the [Amazon SQS console](#).
  - b. Choose **Create queue**.
  - c. For **Type**, choose **Standard**.
  - d. For **Name**, enter **ticketPaymentQueue**.
  - e. Under **Access policy**, for **Choose method**, choose **Advanced**.
  - f. In the JSON policy box, paste the following policy:

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "sns.amazonaws.com"  
            },  
            "Action": "sns:Publish",  
            "Resource": "*",  
            "Condition": {  
                "ArnEquals": {  
                    "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:ticketTopic"  
                }  
            }  
        }  
    ]  
}
```

```
        }  
    ]  
}
```

In this access policy, replace the AWS account number ([123456789012](#)) with your own, and change the AWS Region ([us-east-1](#)) accordingly.

- g. Choose **Create queue**.
- h. Repeat these steps to create a second SQS queue named **ticketFraudQueue**.

For more information on creating SQS queues, see [Creating an Amazon SQS queue \(console\)](#) in the *Amazon Simple Queue Service Developer Guide*.

3. Create the SNS topic:

- a. Open the [Topics page](#) of the Amazon SNS console.
- b. Choose **Create topic**.
- c. Under **Details**, for **Type**, choose **Standard**.
- d. For **Name**, enter **ticketTopic**.
- e. Choose **Create topic**.

For more information on creating SNS topics, see [Creating an Amazon SNS topic \(p. 24\)](#).

4. Subscribe both SQS queues to the SNS topic:

- a. In the [Amazon SNS console](#), on the **ticketTopic** topic's details page, choose **Create subscription**.
- b. Under **Details**, for **Protocol**, choose **Amazon SQS**.
- c. For **Endpoint**, choose the Amazon Resource Name (ARN) of the **ticketPaymentQueue** queue.
- d. Choose **Create subscription**.
- e. Repeat these steps to create a second subscription using the ARN of the **ticketFraudQueue** queue.

For more information on subscribing to SNS topics, see [Subscribing to an Amazon SNS topic \(p. 31\)](#). You can also subscribe SQS queues to SNS topics from the Amazon SQS console.

For more information, see [Subscribing an Amazon SQS queue to an Amazon SNS topic \(console\)](#) in the *Amazon Simple Queue Service Developer Guide*.

You've created the initial resources for this example use case. To continue, see [Creating the Kinesis Data Firehose delivery stream \(p. 117\)](#).

## Creating the Kinesis Data Firehose delivery stream

This page describes how to create the Amazon Kinesis Data Firehose delivery stream for the [message archiving and analytics example use case \(p. 114\)](#).

### To create the Kinesis Data Firehose delivery stream

1. Open the [Amazon Kinesis services console](#).
2. Choose **Kinesis Data Firehose** and then choose **Create delivery stream**.
3. On the **New delivery stream** page, for **Delivery stream name**, enter **ticketUploadStream**, and then choose **Next**.
4. On the **Process records** page, choose **Next**.
5. On the **Choose a destination** page, do the following:

- a. For **Destination**, choose **Amazon S3**.
  - b. Under **S3 destination**, for **S3 bucket**, choose the S3 bucket that you [created initially \(p. 116\)](#).
  - c. Choose **Next**.
6. On the **Configure settings** page, for **S3 buffer conditions**, do the following:
    - For **Buffer size**, enter **1**.
    - For **Buffer interval**, enter **60**.

Using these values for the Amazon S3 buffer lets you quickly test the configuration. The first condition that is satisfied triggers data delivery to the S3 bucket.

7. On the **Configure settings** page, for **Permissions**, choose to create an AWS Identity and Access Management (IAM) role with the required permissions assigned automatically. Then choose **Next**.
8. On the **Review** page, choose **Create delivery stream**.
9. From the **Kinesis Data Firehose delivery streams page**, choose the delivery stream you just created (`ticketUploadStream`). On the **Details** tab, note the stream's Amazon Resource Name (ARN) for later.

For more information on creating delivery streams, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#) in the *Amazon Kinesis Data Firehose Developer Guide*. For more information on creating IAM roles, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

You've created the Kinesis Data Firehose delivery stream with the required permissions. To continue, see [Subscribing the Kinesis Data Firehose delivery stream to the Amazon SNS topic \(p. 118\)](#).

## Subscribing the Kinesis Data Firehose delivery stream to the Amazon SNS topic

This page describes how to create the following for the [message archiving and analytics example use case \(p. 114\)](#):

- The AWS Identity and Access Management (IAM) role that allows the Amazon SNS subscription to put records on the Amazon Kinesis Data Firehose delivery stream
- The Kinesis Data Firehose delivery stream subscription to the SNS topic

### To create the IAM role for the Amazon SNS subscription

1. Open the [Roles page](#) of the IAM console.
2. Choose **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**.
4. For **Choose a use case**, choose **SNS**. Then choose **Next: Permissions**.
5. Choose **Next: Tags**.
6. Choose **Next: Review**.
7. On the **Review** page, for **Role name**, enter `ticketUploadStreamSubscriptionRole`. Then choose **Create role**.
8. When the role is created, choose its name (`ticketUploadStreamSubscriptionRole`).
9. On the role's **Summary** page, choose **Add inline policy**.
10. On the **Create policy** page, choose the **JSON** tab, and then paste the following policy into the box:

```
{  
    "Version": "2012-10-17",
```

```
"Statement": [  
    {  
        "Action": [  
            "firehose:DescribeDeliveryStream",  
            "firehose>ListDeliveryStreams",  
            "firehose>ListTagsForDeliveryStream",  
            "firehose:PutRecord",  
            "firehose:PutRecordBatch"  
        ],  
        "Resource": [  
            "arn:aws:firehose:us-east-1:123456789012:deliverystream/  
ticketUploadStream"  
        ],  
        "Effect": "Allow"  
    }  
]
```

In this policy, replace the AWS account number ([123456789012](#)) with your own, and change the AWS Region ([us-east-1](#)) accordingly.

11. Choose **Review policy**.
12. On the **Review policy** page, for **Name**, enter **FirehoseSnsPolicy**. Then choose **Create policy**.
13. On the role's **Summary** page, note the **Role ARN** for later.

For more information on creating IAM roles, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

### To subscribe the Kinesis Data Firehose delivery stream to the SNS topic

1. Open the [Topics page](#) of the Amazon SNS console.
2. On the **Subscriptions**, tab, choose **Create subscription**.
3. Under **Details**, for **Protocol**, choose **Amazon Kinesis Data Firehose**.
4. For **Endpoint**, enter the Amazon Resource Name (ARN) of the **ticketUploadStream** delivery stream that you created earlier. For example, enter **arn:aws:firehose:us-east-1:123456789012:deliverystream/ticketUploadStream**.
5. For **Subscription role ARN**, enter the ARN of the **ticketUploadStreamSubscriptionRole** IAM role that you created earlier. For example, enter **arn:aws:iam::123456789012:role/ticketUploadStreamSubscriptionRole**.
6. Select the **Enable raw message delivery** check box.
7. Choose **Create subscription**.

You've created the IAM role and SNS topic subscription. To continue, see [Testing and querying the configuration \(p. 119\)](#).

## Testing and querying the configuration

This page describes how to test the [message archiving and analytics example use case \(p. 114\)](#) by publishing a message to the Amazon SNS topic. The instructions include an example query that you can run and adapt to your own needs.

### To test your configuration

1. Open the [Topics page](#) of the Amazon SNS console.
2. Choose the **ticketTopic** topic.
3. Choose **Publish message**.

4. On the **Publish message to topic** page, enter the following for the message body. Add a newline character at the end of the message.

```
{"BookingDate": "2020-12-15", "BookingTime": "2020-12-15\n04:15:05", "Destination": "Miami", "FlyingFrom": "Vancouver", "TicketNumber": "abcd1234"}
```

Keep all other options as their defaults.

5. Choose **Publish message**.

For more information on publishing messages, see [Amazon SNS message publishing \(p. 58\)](#).

6. After the delivery stream interval of 60 seconds, open the [Amazon Simple Storage Service \(Amazon S3\) console](#) and choose the Amazon S3 bucket that you [created initially \(p. 116\)](#).

The published message appears in the bucket.

## To query the data

1. Open the [Amazon Athena console](#).
2. Run a query.

For example, assume that the notifications table in the default schema contains the following data:

```
{"BookingDate": "2020-12-15", "BookingTime": "2020-12-15\n04:15:05", "Destination": "Miami", "FlyingFrom": "Vancouver", "TicketNumber": "abcd1234"},\n{"BookingDate": "2020-12-15", "BookingTime": "2020-12-15\n11:30:15", "Destination": "Miami", "FlyingFrom": "Omaha", "TicketNumber": "efgh5678"},\n {"BookingDate": "2020-12-15", "BookingTime": "2020-12-15\n3:30:10", "Destination": "Miami", "FlyingFrom": "NewYork", "TicketNumber": "ijkl9012"},\n {"BookingDate": "2020-12-15", "BookingTime": "2020-12-15\n12:30:05", "Destination": "Delhi", "FlyingFrom": "Omaha", "TicketNumber": "mnop3456"}
```

To find the top destination, run the following query:

```
SELECT destination\nFROM default.notifications\nGROUP BY destination\nORDER BY count(*) desc\nLIMIT 1;
```

To query for tickets sold during a specific date and time range, run a query like the following:

```
SELECT *\nFROM default.notifications\nWHERE bookingtime\n    BETWEEN TIMESTAMP '2020-12-15 10:00:00'\n        AND TIMESTAMP '2020-12-15 12:00:00';
```

You can adapt both sample queries for your own needs. For more information on using Athena to run queries, see [Getting Started](#) in the [Amazon Athena User Guide](#).

## Cleaning up

To avoid incurring usage charges after you're done testing, delete the following resources that you created during the tutorial:

- Amazon SNS subscriptions
- Amazon SNS topic
- Amazon Simple Queue Service (Amazon SQS) queues
- Amazon S3 bucket
- Amazon Kinesis Data Firehose delivery stream
- AWS Identity and Access Management (IAM) roles and policies

## Using an AWS CloudFormation template

To automate the deployment of the Amazon SNS [message archiving and analytics example use case \(p. 114\)](#), you can use the following YAML template:

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Template for creating an SNS archiving use case
Resources:
  ticketUploadStream:
    DependsOn:
      - ticketUploadStreamRolePolicy
    Type: AWS::KinesisFirehose::DeliveryStream
    Properties:
      S3DestinationConfiguration:
        BucketARN: !Sub 'arn:${AWS::Partition}:s3:::${ticketArchiveBucket}'
        BufferingHints:
          IntervalInSeconds: 60
          SizeInMBs: 1
        CompressionFormat: UNCOMPRESSED
        RoleARN: !GetAtt ticketUploadStreamRole.Arn
  ticketArchiveBucket:
    Type: AWS::S3::Bucket
  ticketTopic:
    Type: AWS::SNS::Topic
  ticketPaymentQueue:
    Type: AWS::SQS::Queue
  ticketFraudQueue:
    Type: AWS::SQS::Queue
  ticketQueuePolicy:
    Type: AWS::SQS::QueuePolicy
    Properties:
      PolicyDocument:
        Statement:
          Effect: Allow
          Principal:
            Service: sns.amazonaws.com
          Action:
            - sqs:SendMessage
          Resource: '*'
          Condition:
            ArnEquals:
              aws:SourceArn: !Ref ticketTopic
        Queues:
          - !Ref ticketPaymentQueue
          - !Ref ticketFraudQueue
  ticketUploadStreamSubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref ticketTopic
      Endpoint: !GetAtt ticketUploadStream.Arn
      Protocol: firehose
      SubscriptionRoleArn: !GetAtt ticketUploadStreamSubscriptionRole.Arn
```

```

ticketPaymentQueueSubscription:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref ticketTopic
    Endpoint: !GetAtt ticketPaymentQueue.Arn
    Protocol: sqs
ticketFraudQueueSubscription:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref ticketTopic
    Endpoint: !GetAtt ticketFraudQueue.Arn
    Protocol: sqs
ticketUploadStreamRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Sid: ''
          Effect: Allow
          Principal:
            Service: firehose.amazonaws.com
          Action: sts:AssumeRole
ticketUploadStreamRolePolicy:
  Type: AWS::IAM::Policy
  Properties:
    PolicyName: FirehoseTicketUploadStreamRolePolicy
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - s3:AbortMultipartUpload
            - s3:GetBucketLocation
            - s3:GetObject
            - s3>ListBucket
            - s3>ListBucketMultipartUploads
            - s3:PutObject
          Resource:
            - !Sub 'arn:aws:s3:::${ticketArchiveBucket}'
            - !Sub 'arn:aws:s3:::${ticketArchiveBucket}/*'
      Roles:
        - !Ref ticketUploadStreamRole
ticketUploadStreamSubscriptionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - sns.amazonaws.com
          Action:
            - sts:AssumeRole
    Policies:
      - PolicyName: SNSKinesisFirehoseAccessPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Action:
              - firehose:DescribeDeliveryStream
              - firehose>ListDeliveryStreams
              - firehose>ListTagsForDeliveryStream
              - firehose:PutRecord
              - firehose:PutRecordBatch

```

```
Effect: Allow
Resource:
- !GetAtt ticketUploadStream.Arn
```

## Fanout to Lambda functions

Amazon SNS and AWS Lambda are integrated so you can invoke Lambda functions with Amazon SNS notifications. When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message. The Lambda function receives the message payload as an input parameter and can manipulate the information in the message, publish the message to other SNS topics, or send the message to other AWS services.

Amazon SNS also supports message delivery status attributes for message notifications sent to Lambda endpoints. For more information, see [Amazon SNS message delivery status \(p. 87\)](#).

### Prerequisites

To invoke Lambda functions using Amazon SNS notifications, you need the following:

- Lambda function
- Amazon SNS topic

For information about creating a Lambda function to use with Amazon SNS, see [Using Lambda with Amazon SNS](#). For information about creating an Amazon SNS topic, see [Create a topic](#).

When you use Amazon SNS to deliver messages from opt-in regions to regions which are enabled by default, you must alter the policy created in the AWS Lambda function by replacing the principal `sns.amazonaws.com` with `sns.<opt-in-region>.amazonaws.com`.

For example, if you want to subscribe a Lambda function in US East (N. Virginia) to an SNS topic in Asia Pacific (Hong Kong), change the principal in the AWS Lambda function policy to `sns.ap-east-1.amazonaws.com`. Opt-in regions include any regions launched after March 20, 2019, which includes Asia Pacific (Hong Kong), Middle East (Bahrain), EU (Milano), and Africa (Cape Town). Regions launched prior to March 20, 2019 are enabled by default.

#### Note

AWS doesn't support cross-region delivery to Lambda from a region that is enabled by default to an opt-in region. Also, cross-region forwarding of SNS messages from opt-in regions to other opt-in regions is not supported.

## Subscribing a function to a topic

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic.
4. In the **Subscriptions** section, choose **Create subscription**.
5. On the **Create subscription** page, in the **Details** section, do the following:
  - a. Verify the chosen **Topic ARN**.
  - b. For **Protocol** choose AWS Lambda.
  - c. For **Endpoint** enter the ARN of a function.

- d. Choose **Create subscription**.

When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message. For information about how to use AWS Lambda with Amazon SNS, including a tutorial, see [Using AWS Lambda with Amazon SNS](#).

## Fanout to Amazon SQS queues

Amazon SNS works closely with Amazon Simple Queue Service (Amazon SQS). These services provide different benefits for developers. Amazon SNS allows applications to send time-critical messages to multiple subscribers through a “push” mechanism, eliminating the need to periodically check or “poll” for updates. Amazon SQS is a message queue service used by distributed applications to exchange messages through a polling model, and can be used to decouple sending and receiving components—without requiring each component to be concurrently available. Using Amazon SNS and Amazon SQS together, messages can be delivered to applications that require immediate notification of an event, and also persisted in an Amazon SQS queue for other applications to process at a later time.

When you subscribe an Amazon SQS queue to an Amazon SNS topic, you can publish a message to the topic and Amazon SNS sends an Amazon SQS message to the subscribed queue. The Amazon SQS message contains the subject and message that were published to the topic along with metadata about the message in a JSON document. The Amazon SQS message will look similar to the following JSON document.

```
{  
    "Type" : "Notification",  
    "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",  
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",  
    "Subject" : "Testing publish to subscribed queues",  
    "Message" : "Hello world!",  
    "Timestamp" : "2012-03-29T05:12:16.901Z",  
    "SignatureVersion" : "1",  
    "Signature" : "EXAMPLEnTrFPa3...",  
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",  
    "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-db410a0f2bee"  
}
```

## Subscribing an Amazon SQS queue to an Amazon SNS topic

To enable an Amazon SNS topic to send messages to an Amazon SQS queue, do one of the following:

- Use the [Amazon SQS console](#), which simplifies the process. For more information, see [Subscribing an Amazon SQS queue to an Amazon SNS topic](#) in the *Amazon Simple Queue Service Developer Guide*.
- Follow these steps:
  1. Get the Amazon Resource Name (ARN) of the queue you want to send messages to and the topic to which you want to subscribe the queue. (p. 125)
  2. Give `sqs:SendMessage` permission to the Amazon SNS topic so that it can send messages to the queue. (p. 125)
  3. Subscribe the queue to the Amazon SNS topic. (p. 126)
  4. Give IAM users or AWS accounts the appropriate permissions to publish to the Amazon SNS topic and read messages from the Amazon SQS queue. (p. 127)

5. Test it out by publishing a message to the topic and reading the message from the queue. (p. 128)

To learn about how to set up a topic to send messages to a queue that is in a different AWS-account; see [Sending Amazon SNS messages to an Amazon SQS queue in a different account \(p. 84\)](#).

To see an AWS CloudFormation template that creates a topic that sends messages to two queues, see [Using an AWS CloudFormation template to create a topic that sends messages to Amazon SQS queues \(p. 129\)](#).

## Step 1: Get the ARN of the queue and topic

When subscribing a queue to your topic, you'll need a copy of the ARN for the queue. Similarly, when giving permission for the topic to send messages to the queue, you'll need a copy of the ARN for the topic.

To get the queue ARN, you can use the Amazon SQS console or the [GetQueueAttributes](#) API action.

### To get the queue ARN from the Amazon SQS console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Select the box for the queue whose ARN you want to get.
3. From the **Details** section, copy the ARN value so that you can use it to subscribe to the Amazon SNS topic.

To get the topic ARN, you can use the Amazon SNS console, the [sns-get-topic-attributes](#) command, or the [GetQueueAttributes](#) API action.

### To get the topic ARN from the Amazon SNS console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose the topic whose ARN you want to get.
3. From the **Details** section, copy the ARN value so that you can use it to give permission for the Amazon SNS topic to send messages to the queue.

## Step 2: Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue

For an Amazon SNS topic to be able to send messages to a queue, you must set a policy on the queue that allows the Amazon SNS topic to perform the `sqS:SendMessage` action.

Before you subscribe a queue to a topic, you need a topic and a queue. If you haven't already created a topic or queue, create them now. For more information, see [Creating a Topic](#), and see [Creating a Queue in the Amazon Simple Queue Service Developer Guide](#).

To set a policy on a queue, you can use the Amazon SQS console or the [SetQueueAttributes](#) API action. Before you start, make sure you have the ARN for the topic that you want to allow to send messages to the queue.

### To set a `SendMessage` policy on a queue using the Amazon SQS console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.

2. Select the box for the queue whose policy you want to set, choose the **Access policy** tab, and then choose **Edit**.
3. In the **Access policy** section, define who can access your queue.
  - Add a condition that allows the action for the topic.
  - Set **Principal** to be the Amazon SNS service, as shown in the example below.

For example, the following policy allows MyTopic to send messages to MyQueue.

```
{  
    "Statement": [ {  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "sns.amazonaws.com"  
        },  
        "Action": "sns:SendMessage",  
        "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",  
        "Condition": {  
            "ArnEquals": {  
                "aws:SourceArn": "arn:aws:sqs:us-east-2:123456789012:MyQueue"  
            }  
        }  
    }]  
}
```

## Step 3: Subscribe the queue to the Amazon SNS topic

To send messages to a queue through a topic, you must subscribe the queue to the Amazon SNS topic. You specify the queue by its ARN. To subscribe to a topic, you can use the Amazon SNS console, the `sns-subscribe` CLI command, or the `Subscribe` API action. Before you start, make sure you have the ARN for the queue that you want to subscribe.

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic.
4. On the **MyTopic** page, in the **Subscriptions** page, choose **Create subscription**.
5. On the **Create subscription** page, in the **Details** section, do the following:
  - a. Verify the **Topic ARN**.
  - b. For **Protocol**, choose **Amazon SQS**.
  - c. For **Endpoint**, enter the ARN of an Amazon SQS queue.
  - d. Choose **Create Subscription**.

When the subscription is confirmed, your new subscription's **Subscription ID** displays its subscription ID. If the owner of the queue creates the subscription, the subscription is automatically confirmed and the subscription should be active almost immediately.

Usually, you'll be subscribing your own queue to your own topic in your own account. However, you can also subscribe a queue from a different account to your topic. If the user who creates the subscription is not the owner of the queue (for example, if a user from account A subscribes a queue from account B to a topic in account A), the subscription must be confirmed. For more information about subscribing a queue from a different account and confirming the subscription, see [Sending Amazon SNS messages to an Amazon SQS queue in a different account \(p. 84\)](#).

## Step 4: Give users permissions to the appropriate topic and queue actions

You should use AWS Identity and Access Management (IAM) to allow only appropriate users to publish to the Amazon SNS topic and to read/delete messages from the Amazon SQS queue. For more information about controlling actions on topics and queues for IAM users, see [Using identity-based policies with Amazon SNS \(p. 380\)](#), and [Identity and access management in Amazon SQS](#) in the Amazon Simple Queue Service Developer Guide.

There are two ways to control access to a topic or queue:

- [Add a policy to an IAM user or group \(p. 127\)](#). The simplest way to give users permissions to topics or queues is to create a group and add the appropriate policy to the group and then add users to that group. It's much easier to add and remove users from a group than to keep track of which policies you set on individual users.
- [Add a policy to topic or queue \(p. 127\)](#). If you want to give permissions to a topic or queue to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

You should use the first method for most cases (apply policies to groups and manage permissions for users by adding or removing the appropriate users to the groups). If you need to give permissions to a user in another account, you should use the second method.

### Adding a policy to an IAM user or group

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sns:Publish` action on the topic `MyTopic`.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "sns:Publish",  
      "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"  
    }]  
}
```

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on the queues `MyQueue1` and `MyQueue2`.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "sqs:ReceiveMessage",  
        "sqs:DeleteMessage"  
      ],  
      "Resource": [  
        "arn:aws:sqs:us-east-2:123456789012:MyQueue1",  
        "arn:aws:sqs:us-east-2:123456789012:MyQueue2"  
      ]  
    }]  
}
```

### Adding a policy to a topic or queue

The following example policies show how to give another account permissions to a topic and queue.

### Note

When you give another AWS account access to a resource in your account, you are also giving IAM users who have admin-level access (wildcard access) permissions to that resource. All other IAM users in the other account are automatically denied access to your resource. If you want to give specific IAM users in that AWS account access to your resource, the account or an IAM user with admin-level access must delegate permissions for the resource to those IAM users. For more information about cross-account delegation, see [Enabling Cross-Account Access](#) in the *Using IAM Guide*.

If you added the following policy to a topic MyTopic in account 123456789012, you would give account 111122223333 permission to perform the sns:Publish action on that topic.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "111122223333"  
            },  
            "Action": "sns:Publish",  
            "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"  
        }  
    ]  
}
```

If you added the following policy to a queue MyQueue in account 123456789012, you would give account 111122223333 permission to perform the sqs:ReceiveMessage and sqs:DeleteMessage actions on that queue.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "111122223333"  
            },  
            "Action": [  
                "sqs:DeleteMessage",  
                "sqs:ReceiveMessage"  
            ],  
            "Resource": [  
                "arn:aws:sqs:us-east-2:123456789012:MyQueue"  
            ]  
        }  
    ]  
}
```

## Step 5: Test the topic's queue subscriptions

You can test a topic's queue subscriptions by publishing to the topic and viewing the message that the topic sends to the queue.

### To publish to a topic using the Amazon SNS console

1. Using the credentials of the AWS account or IAM user with permission to publish to the topic, sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. On the navigation panel, choose the topic and choose **Publish to Topic**.
3. In the **Subject** box, enter a subject (for example, **Testing publish to queue**) in the **Message** box, enter some text (for example, **Hello world!**), and choose **Publish Message**. The following message appears: Your message has been successfully published.

### To view the message from the topic using the Amazon SQS console

1. Using the credentials of the AWS account or IAM user with permission to view messages in the queue, sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Check the box for the queue that is subscribed to the topic.
3. From the **Queue Action** drop-down, choose **View/Delete Messages** and choose **Start Polling for Messages**. A message with a type of **Notification** appears.
4. In the **Body** column, choose **More Details**. The **Message Details** box contains a JSON document that contains the subject and message that you published to the topic. The message looks similar to the following JSON document.

```
{  
    "Type" : "Notification",  
    "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",  
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",  
    "Subject" : "Testing publish to subscribed queues",  
    "Message" : "Hello world!",  
    "Timestamp" : "2012-03-29T05:12:16.901Z",  
    "SignatureVersion" : "1",  
    "Signature" : "EXAMPLEnTrFPa3...",  
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",  
    "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-db410a0f2bee"  
}
```

5. Choose **Close**. You have successfully published to a topic that sends notification messages to a queue.

## Using an AWS CloudFormation template to create a topic that sends messages to Amazon SQS queues

AWS CloudFormation enables you to use a template file to create and configure a collection of AWS resources together as a single unit. This section has an example template that makes it easy to deploy topics that publish to queues. The templates take care of the setup steps for you by creating two queues, creating a topic with subscriptions to the queues, adding a policy to the queues so that the topic can send messages to the queues, and creating IAM users and groups to control access to those resources.

For more information about deploying AWS resources using an AWS CloudFormation template, see [Get Started](#) in the *AWS CloudFormation User Guide*.

## Using an AWS CloudFormation template to set up topics and queues within an AWS account

The example template creates an Amazon SNS topic that can send messages to two Amazon SQS queues with appropriate permissions for members of one IAM group to publish to the topic and another to read messages from the queues. The template also creates IAM users that are added to each group.

You copy the template contents into a file. You can also download the template from the [AWS CloudFormation Templates page](#). On the templates page, choose **Browse sample templates by AWS service** and then choose **Amazon Simple Queue Service**.

MySNSTopic is set up to publish to two subscribed endpoints, which are two Amazon SQS queues (MyQueue1 and MyQueue2). MyPublishTopicGroup is an IAM group whose members have permission

to publish to MySNSTopic using the [Publish](#) API action or [sns-publish](#) command. The template creates the IAM users MyPublishUser and MyQueueUser and gives them login profiles and access keys. The user who creates a stack with this template specifies the passwords for the login profiles as input parameters. The template creates access keys for the two IAM users with MyPublishUserKey and MyQueueUserKey. AddUserToMyPublishTopicGroup adds MyPublishUser to the MyPublishTopicGroup so that the user will have the permissions assigned to the group.

MyRDMessagesQueueGroup is an IAM group whose members have permission to read and delete messages from the two Amazon SQS queues using the [ReceiveMessage](#) and [DeleteMessage](#) API actions. AddUserToMyQueueGroup adds MyQueueUser to the MyRDMessagesQueueGroup so that the user will have the permissions assigned to the group. MyQueuePolicy assigns permission for MySNSTopic to publish its notifications to the two queues.

The following listing shows the AWS CloudFormation template contents.

```
{
    "AWSTemplateFormatVersion" : "2010-09-09",

    "Description" : "AWS CloudFormation Sample Template SNSToSQS: This Template creates an SNS topic that can send messages to two SQS queues with appropriate permissions for one IAM user to publish to the topic and another to read messages from the queues.

        MySNSTopic is set up to publish to two subscribed endpoints, which are two SQS queues (MyQueue1 and MyQueue2). MyPublishUser is an IAM user that can publish to MySNSTopic using the Publish API. MyTopicPolicy assigns that permission to MyPublishUser. MyQueueUser is an IAM user that can read messages from the two SQS queues. MyQueuePolicy assigns those permissions to MyQueueUser. It also assigns permission for MySNSTopic to publish its notifications to the two queues. The template creates access keys for the two IAM users with MyPublishUserKey and MyQueueUserKey. ***Warning*** you will be billed for the AWS resources used if you create a stack from this template.",

    "Parameters": {
        "MyPublishUserPassword": {
            "NoEcho": "true",
            "Type": "String",
            "Description": "Password for the IAM user MyPublishUser",
            "MinLength": "1",
            "MaxLength": "41",
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "must contain only alphanumeric characters."
        },
        "MyQueueUserPassword": {
            "NoEcho": "true",
            "Type": "String",
            "Description": "Password for the IAM user MyQueueUser",
            "MinLength": "1",
            "MaxLength": "41",
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "must contain only alphanumeric characters."
        }
    },

    "Resources": {
        "MySNSTopic": {
            "Type": "AWS::SNS::Topic",
            "Properties": {
                "Subscription": [
                    {
                        "Endpoint": {
                            "Fn::GetAtt": ["MyQueue1", "Arn"]
                        },
                        "Protocol": "sqS"
                    }
                ]
            }
        }
    }
}
```

```

        },
        {
            "Endpoint": {
                "Fn::GetAtt": [ "MyQueue2", "Arn" ]
            },
            "Protocol": "sqS"
        }
    ]
},
"MyQueue1": {
    "Type": "AWS::SQS::Queue"
},
"MyQueue2": {
    "Type": "AWS::SQS::Queue"
},
"MyPublishUser": {
    "Type": "AWS::IAM::User",
    "Properties": {
        "LoginProfile": {
            "Password": {
                "Ref": "MyPublishUserPassword"
            }
        }
    }
},
"MyPublishUserKey": {
    "Type": "AWS::IAM::AccessKey",
    "Properties": {
        "UserName": {
            "Ref": "MyPublishUser"
        }
    }
},
"MyPublishTopicGroup": {
    "Type": "AWS::IAM::Group",
    "Properties": {
        "Policies": [
            {
                "PolicyName": "MyTopicGroupPolicy",
                "PolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": [
                                "sns:Publish"
                            ],
                            "Resource": {
                                "Ref": "MySNSTopic"
                            }
                        }
                    ]
                }
            }
        ]
    }
},
"AddUserToMyPublishTopicGroup": {
    "Type": "AWS::IAM::UserToGroupAddition",
    "Properties": {
        "GroupName": {
            "Ref": "MyPublishTopicGroup"
        },
        "Users": [
            {
                "Ref": "MyPublishUser"
            }
        ]
    }
},
"MyQueueUser": {
    "Type": "AWS::IAM::User",

```

```

    "Properties": {
        "LoginProfile": {
            "Password": {
                "Ref": "MyQueueUserPassword"
            }
        }
    },
    "MyQueueUserKey": {
        "Type": "AWS::IAM::AccessKey",
        "Properties": {
            "UserName": {
                "Ref": "MyQueueUser"
            }
        }
    },
    "MyRDMMessageQueueGroup": {
        "Type": "AWS::IAM::Group",
        "Properties": {
            "Policies": [
                {
                    "PolicyName": "MyQueueGroupPolicy",
                    "PolicyDocument": {
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "sns:DeleteMessage",
                                    "sns:ReceiveMessage"
                                ],
                                "Resource": [
                                    {
                                        "Fn::GetAtt": [ "MyQueue1", "Arn" ]
                                    },
                                    {
                                        "Fn::GetAtt": [ "MyQueue2", "Arn" ]
                                    }
                                ]
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AddUserToMyQueueGroup": {
        "Type": "AWS::IAM::UserToGroupAddition",
        "Properties": {
            "GroupName": {
                "Ref": "MyRDMMessageQueueGroup"
            },
            "Users": [
                {
                    "Ref": "MyQueueUser"
                }
            ]
        }
    },
    "MyQueuePolicy": {
        "Type": "AWS::SQS::QueuePolicy",
        "Properties": {
            "PolicyDocument": {
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "sns.amazonaws.com"
                        },
                        "Action": [ "sns:SendMessage" ],
                        "Resource": "*",
                        "Condition": {
                            "ArnEquals": {
                                "aws:SourceArn": {
                                    "Ref": "MySNSTopic"
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
}

```

```
        }
    }
}
},
"Queues": [
    {
        "Ref": "MyQueue1"
    },
    {
        "Ref": "MyQueue2"
    }
]
},
"Outputs": {
    "MySNSTopicTopicARN": {
        "Value": {
            "Ref": "MySNSTopic"
        }
    },
    "MyQueue1Info": {
        "Value": {
            "Fn::Join": [
                " ",
                [
                    {
                        "ARN:",
                        {
                            "Fn::GetAtt": ["MyQueue1", "Arn"]
                        },
                        "URL:",
                        {
                            "Ref": "MyQueue1"
                        }
                    ]
                ]
            }
        }
    },
    "MyQueue2Info": {
        "Value": {
            "Fn::Join": [
                " ",
                [
                    {
                        "ARN:",
                        {
                            "Fn::GetAtt": ["MyQueue2", "Arn"]
                        },
                        "URL:",
                        {
                            "Ref": "MyQueue2"
                        }
                    ]
                ]
            }
        }
    },
    "MyPublishUserInfo": {
        "Value": {
            "Fn::Join": [
                " ",
                [
                    {
                        "ARN:",
                        {
                            "Fn::GetAtt": ["MyPublishUser", "Arn"]
                        },
                        "Access Key:",
                        {
                            "Ref": "MyPublishUserKey"
                        },
                        "Secret Key:",
                        {
                            "Ref": "MyPublishUserSecret"
                        }
                    ]
                ]
            }
        }
    }
}
```

```
        "Secret Key:",
    {
        "Fn::GetAtt": ["MyPublishUserKey", "SecretAccessKey"]
    }
]
}
},
"MyQueueUserInfo": {
    "Value": {
        "Fn::Join": [
            "",
            [
                "ARN:",
                {
                    "Fn::GetAtt": ["MyQueueUser", "Arn"]
                },
                "Access Key:",
                {
                    "Ref": "MyQueueUserKey"
                },
                "Secret Key:",
                {
                    "Fn::GetAtt": ["MyQueueUserKey", "SecretAccessKey"]
                }
            ]
        ]
    }
}
}
```

## Fanout to HTTP/S endpoints

You can use [Amazon SNS](#) to send notification messages to one or more HTTP or HTTPS endpoints. When you subscribe an endpoint to a topic, you can publish a notification to the topic and Amazon SNS sends an HTTP POST request delivering the contents of the notification to the subscribed endpoint. When you subscribe the endpoint, you choose whether Amazon SNS uses HTTP or HTTPS to send the POST request to the endpoint. If you use HTTPS, then you can take advantage of the support in Amazon SNS for the following:

- **Server Name Indication (SNI)**—This allows Amazon SNS to support HTTPS endpoints that require SNI, such as a server requiring multiple certificates for hosting multiple domains. For more information about SNI, see [Server Name Indication](#).
- **Basic and Digest Access Authentication**—This allows you to specify a username and password in the HTTPS URL for the HTTP POST request, such as `https://user:password@domain.com` or `https://user@domain.com`. The username and password are encrypted over the SSL connection established when using HTTPS. Only the domain name is sent in plaintext. For more information about Basic and Digest Access Authentication, see [RFC-2617](#).

### Note

The client service must be able to support the `HTTP/1.1 401 Unauthorized` header response

The request contains the subject and message that were published to the topic along with metadata about the notification in a JSON document. The request will look similar to the following HTTP POST

request. For details about the HTTP header and the JSON format of the request body, see [HTTP/HTTPS headers \(p. 143\)](#) and [HTTP/HTTPS notification JSON format \(p. 145\)](#).

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: da41e39f-ea4d-435a-b922-c6aae3915ebe
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 761
Content-Type: text/plain; charset=UTF-8
Host: ec2-50-17-44-49.compute-1.amazonaws.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
    "Type" : "Notification",
    "MessageId" : "da41e39f-ea4d-435a-b922-c6aae3915ebe",
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
    "Subject" : "test",
    "Message" : "test message",
    "Timestamp" : "2012-04-25T21:49:25.719Z",
    "SignatureVersion" : "1",
    "Signature" :
    "EXAMPLE1DMXvB8r9R83tGoNn0ecwd5Uj1lzsVsbItzfaMpN2nk5HVSw7XnOn/49IkxDKz8YrlH2qJXj2iZB0Zo2071c4qQk1fMUD
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
    "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55"
}
```

### Topics

- [Subscribing an HTTP/S endpoint to a topic \(p. 135\)](#)
- [Verifying the signatures of Amazon SNS messages \(p. 140\)](#)
- [Parsing message formats \(p. 142\)](#)

## Subscribing an HTTP/S endpoint to a topic

The pages in this section describe how to subscribe HTTP/S endpoints to Amazon SNS topics.

### Topics

- [Step 1: Make sure your endpoint is ready to process Amazon SNS messages \(p. 135\)](#)
- [Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic \(p. 138\)](#)
- [Step 3: Confirm the subscription \(p. 139\)](#)
- [Step 4: Set the delivery retry policy for the subscription \(optional\) \(p. 139\)](#)
- [Step 5: Give users permissions to publish to the topic \(optional\) \(p. 139\)](#)
- [Step 6: Send messages to the HTTP/HTTPS endpoint \(p. 140\)](#)

### Step 1: Make sure your endpoint is ready to process Amazon SNS messages

Before you subscribe your HTTP or HTTPS endpoint to a topic, you must make sure that the HTTP or HTTPS endpoint has the capability to handle the HTTP POST requests that Amazon SNS uses to send the subscription confirmation and notification messages. Usually, this means creating and deploying

a web application (for example, a Java servlet if your endpoint host is running Linux with Apache and Tomcat) that processes the HTTP requests from Amazon SNS. When you subscribe an HTTP endpoint, Amazon SNS sends it a subscription confirmation request. Your endpoint must be prepared to receive and process this request when you create the subscription because Amazon SNS sends this request at that time. Amazon SNS will not send notifications to the endpoint until you confirm the subscription. Once you confirm the subscription, Amazon SNS will send notifications to the endpoint when a publish action is performed on the subscribed topic.

### To set up your endpoint to process subscription confirmation and notification messages

1. Your code should read the HTTP headers of the HTTP POST requests that Amazon SNS sends to your endpoint. Your code should look for the header field `x-amz-sns-message-type`, which tells you the type of message that Amazon SNS has sent to you. By looking at the header, you can determine the message type without having to parse the body of the HTTP request. There are two types that you need to handle: `SubscriptionConfirmation` and `Notification`. The `UnsubscribeConfirmation` message is used only when the subscription is deleted from the topic.

For details about the HTTP header, see [HTTP/HTTPS headers \(p. 143\)](#). The following HTTP POST request is an example of a subscription confirmation message.

```
POST / HTTP/1.1
    x-amz-sns-message-type: SubscriptionConfirmation
    x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
    x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
    Content-Length: 1336
    Content-Type: text/plain; charset=UTF-8
    Host: example.com
    Connection: Keep-Alive
    User-Agent: Amazon Simple Notification Service Agent

{
    "Type" : "SubscriptionConfirmation",
    "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
    "Token" : "2336412f37f...",
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
    "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL included in this message.",
    "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37...",
    "Timestamp" : "2012-04-26T20:45:04.751Z",
    "SignatureVersion" : "1",
    "Signature" : "EXAMPLEPfH+...",
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

2. Your code should parse the JSON document in the body of the HTTP POST request to read the name-value pairs that make up the Amazon SNS message. Use a JSON parser that handles converting the escaped representation of control characters back to their ASCII character values (for example, converting `\n` to a newline character). You can use an existing JSON parser such as the [Jackson JSON Processor](#) or write your own. In order to send the text in the subject and message fields as valid JSON, Amazon SNS must convert some control characters to escaped representations that can be included in the JSON document. When you receive the JSON document in the body of the POST request sent to your endpoint, you must convert the escaped characters back to their original character values if you want an exact representation of the original subject and messages published to the topic. This is critical if you want to verify the signature of a notification because the signature uses the message and subject in their original forms as part of the string to sign.
3. Your code should verify the authenticity of a notification, subscription confirmation, or unsubscribe confirmation message sent by Amazon SNS. Using information contained in the Amazon SNS

message, your endpoint can recreate the signature so that you can verify the contents of the message by matching your signature with the signature that Amazon SNS sent with the message. For more information about verifying the signature of a message, see [Verifying the signatures of Amazon SNS messages \(p. 140\)](#).

4. Based on the type specified by the header field `x-amz-sns-message-type`, your code should read the JSON document contained in the body of the HTTP request and process the message. Here are the guidelines for handling the two primary types of messages:

### SubscriptionConfirmation

Read the value for `SubscribeURL` and visit that URL. To confirm the subscription and start receiving notifications at the endpoint, you must visit the `SubscribeURL` (for example, by sending an HTTP GET request to the URL). See the example HTTP request in the previous step to see what the `SubscribeURL` looks like. For more information about the format of the `SubscriptionConfirmation` message, see [HTTP/HTTPS subscription confirmation JSON format \(p. 143\)](#). When you visit the URL, you will get back a response that looks like the following XML document. The document returns the subscription ARN for the endpoint within the `ConfirmSubscriptionResult` element.

```
<ConfirmSubscriptionResponse xmlns="http://sns.amazonaws.com/doc/2010-03-31/">
  <ConfirmSubscriptionResult>
    <SubscriptionArn>arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55</SubscriptionArn>
  </ConfirmSubscriptionResult>
  <ResponseMetadata>
    <RequestId>075ecce8-8dac-11e1-bf80-f781d96e9307</RequestId>
  </ResponseMetadata>
</ConfirmSubscriptionResponse>
```

As an alternative to visiting the `SubscribeURL`, you can confirm the subscription using the `ConfirmSubscription` action with the `Token` set to its corresponding value in the `SubscriptionConfirmation` message. If you want to allow only the topic owner and subscription owner to be able to unsubscribe the endpoint, you call the `ConfirmSubscription` action with an AWS signature.

### Notification

Read the values for `Subject` and `Message` to get the notification information that was published to the topic.

For details about the format of the `Notification` message, see [HTTP/HTTPS headers \(p. 143\)](#). The following HTTP POST request is an example of a notification message sent to the endpoint `example.com`.

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
```

```
"Subject" : "My First Message",
"Message" : "Hello world!",
"Timestamp" : "2012-05-02T00:54:06.655Z",
"SignatureVersion" : "1",
"Signature" : "EXAMPLEw6JRN...",
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
"UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

5. Make sure that your endpoint responds to the HTTP POST message from Amazon SNS with the appropriate status code. The connection will time out in 15 seconds. If your endpoint does not respond before the connection times out or if your endpoint returns a status code outside the range of 200–4xx, Amazon SNS will consider the delivery of the message as a failed attempt.
6. Make sure that your code can handle message delivery retries from Amazon SNS. If Amazon SNS doesn't receive a successful response from your endpoint, it attempts to deliver the message again. This applies to all messages, including the subscription confirmation message. By default, if the initial delivery of the message fails, Amazon SNS attempts up to three retries with a delay between failed attempts set at 20 seconds.

**Note**

The message request times out after 15 seconds. This means that, if the message delivery failure is caused by a timeout, Amazon SNS retries for approximately 35 seconds after the previous delivery attempt. You can set a different delivery policy for the endpoint.

To be clear, Amazon SNS attempts to retry only after a delivery `x-amz-sns-message-id` header field. By comparing the IDs of the messages you have processed with incoming messages, you can determine whether the message is a retry attempt.

7. If you are subscribing an HTTPS endpoint, make sure that your endpoint has a server certificate from a trusted Certificate Authority (CA). Amazon SNS will only send messages to HTTPS endpoints that have a server certificate signed by a CA trusted by Amazon SNS.
8. Deploy the code that you have created to receive Amazon SNS messages. When you subscribe the endpoint, the endpoint must be ready to receive at least the subscription confirmation message.

## Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic

To send messages to an HTTP or HTTPS endpoint through a topic, you must subscribe the endpoint to the Amazon SNS topic. You specify the endpoint using its URL. To subscribe to a topic, you can use the Amazon SNS console, the `sns-subscribe` command, or the `Subscribe` API action. Before you start, make sure you have the URL for the endpoint that you want to subscribe and that your endpoint is prepared to receive the confirmation and notification messages as described in Step 1.

### To subscribe an HTTP or HTTPS endpoint to a topic using the Amazon SNS console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics** and then choose the topic.
3. Choose the **Other actions** drop-down list and select **Subscribe to topic**.
4. In the **Protocol** drop-down list, select **HTTP** or **HTTPS**.
5. In the **Endpoint** box, paste in the URL for the endpoint that you want the topic to send messages to and then choose **Create subscription**.
6. For the **Subscription request received!** message, choose **Close**.

Your new subscription's **Subscription ID** displays PendingConfirmation. When you confirm the subscription, **Subscription ID** will display the subscription ID.

## Step 3: Confirm the subscription

After you subscribe your endpoint, Amazon SNS will send a subscription confirmation message to the endpoint. You should already have code that performs the actions described in [Step 1 \(p. 135\)](#) deployed to your endpoint. Specifically, the code at the endpoint must retrieve the `SubscribeURL` value from the subscription confirmation message and either visit the location specified by `SubscribeURL` itself or make it available to you so that you can manually visit the `SubscribeURL`, for example, using a web browser. Amazon SNS will not send messages to the endpoint until the subscription has been confirmed. When you visit the `SubscribeURL`, the response will contain an XML document containing an element `SubscriptionArn` that specifies the ARN for the subscription. You can also use the Amazon SNS console to verify that the subscription is confirmed: The **Subscription ID** will display the ARN for the subscription instead of the `PendingConfirmation` value that you saw when you first added the subscription.

## Step 4: Set the delivery retry policy for the subscription (optional)

By default, if the initial delivery of the message fails, Amazon SNS attempts up to three retries with a delay between failed attempts set at 20 seconds. As discussed in [Step 1 \(p. 135\)](#), your endpoint should have code that can handle retried messages. By setting the delivery policy on a topic or subscription, you can control the frequency and interval that Amazon SNS will retry failed messages. You can set a delivery policy on a topic or on a particular subscription.

## Step 5: Give users permissions to publish to the topic (optional)

By default, the topic owner has permissions to publish the topic. To enable other users or applications to publish to the topic, you should use AWS Identity and Access Management (IAM) to give publish permission to the topic. For more information about giving permissions for Amazon SNS actions to IAM users, see [Using identity-based policies with Amazon SNS \(p. 380\)](#).

There are two ways to control access to a topic:

- Add a policy to an IAM user or group. The simplest way to give users permissions to topics is to create a group and add the appropriate policy to the group and then add users to that group. It's much easier to add and remove users from a group than to keep track of which policies you set on individual users.
- Add a policy to the topic. If you want to give permissions to a topic to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

You should use the first method for most cases (apply policies to groups and manage permissions for users by adding or removing the appropriate users to the groups). If you need to give permissions to a user in another account, use the second method.

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sns:Publish` action on the topic `MyTopic`.

```
{  
  "Statement": [  
    {  
      "Sid": "AllowPublishToMyTopic",  
      "Effect": "Allow",  
      "Action": "sns:Publish",  
      "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"  
    }  
  ]  
}
```

```
        "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
    }]
}
```

The following example policy shows how to give another account permissions to a topic.

**Note**

When you give another AWS account access to a resource in your account, you are also giving IAM users who have admin-level access (wildcard access) permissions to that resource. All other IAM users in the other account are automatically denied access to your resource. If you want to give specific IAM users in that AWS account access to your resource, the account or an IAM user with admin-level access must delegate permissions for the resource to those IAM users. For more information about cross-account delegation, see [Enabling Cross-Account Access](#) in the *Using IAM Guide*.

If you added the following policy to a topic MyTopic in account 123456789012, you would give account 111122223333 permission to perform the sns:Publish action on that topic.

```
{
  "Statement": [
    {
      "Sid": "Allow-publish-to-topic",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
    }
  ]
}
```

## Step 6: Send messages to the HTTP/HTTPS endpoint

You can send a message to a topic's subscriptions by publishing to the topic. To publish to a topic, you can use the Amazon SNS console, the `sns-publish` CLI command, or the [Publish API](#).

If you followed [Step 1 \(p. 135\)](#), the code that you deployed at your endpoint should process the notification.

### To publish to a topic using the Amazon SNS console

1. Using the credentials of the AWS account or IAM user with permission to publish to the topic, sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. On the navigation panel, choose **Topics** and then choose a topic.
3. Choose the **Publish message** button.
4. In the **Subject** box, enter a subject (for example, `Testing publish to my endpoint`).
5. In the **Message** box, enter some text (for example, `Hello world!`), and choose **Publish message**.

The following message appears: Your message has been successfully published.

## Verifying the signatures of Amazon SNS messages

You should verify the authenticity of a notification, subscription confirmation, or unsubscribe confirmation message sent by Amazon SNS. Using information contained in the Amazon SNS message, your endpoint can recreate the string to sign and the signature so that you can verify the contents of the

message by matching the signature you recreated from the message contents with the signature that Amazon SNS sent with the message.

To help prevent spoofing attacks, you should do the following when verifying messages sent by Amazon SNS:

- Always use HTTPS when getting the certificate from Amazon SNS.
- Validate the authenticity of the certificate.
- Verify the certificate was received from Amazon SNS.
- When possible, use one of the supported AWS SDKs for Amazon SNS to validate and verify messages.

### To verify the signature of an Amazon SNS message when using HTTP query-based requests

1. Extract the name-value pairs from the JSON document in the body of the HTTP POST request that Amazon SNS sent to your endpoint. You'll be using the values of some of the name-value pairs to create the string to sign. When you are verifying the signature of an Amazon SNS message, it is critical that you convert the escaped control characters to their original character representations in the Message and Subject values. These values must be in their original forms when you use them as part of the string to sign. For information about how to parse the JSON document, see [Step 1: Make sure your endpoint is ready to process Amazon SNS messages \(p. 135\)](#).

The `SignatureVersion` tells you the signature version. From the signature version, you can determine the requirements for how to generate the signature. For Amazon SNS notifications, Amazon SNS currently supports signature version 1. This section provides the steps for creating a signature using signature version 1.

2. Get the X509 certificate that Amazon SNS used to sign the message. The `SigningCertURL` value points to the location of the X509 certificate used to create the digital signature for the message. Retrieve the certificate from this location.
3. Extract the public key from the certificate. The public key from the certificate specified by `SigningCertURL` is used to verify the authenticity and integrity of the message.
4. Determine the message type. The format of the string to sign depends on the message type, which is specified by the `Type` value.
5. Create the string to sign. The string to sign is a newline character-delimited list of specific name-value pairs from the message. Each name-value pair is represented with the name first followed by a newline character, followed by the value, and ending with a newline character. The name-value pairs must be listed in byte-sort order.

Depending on the message type, the string to sign must have the following name-value pairs.

#### Notification

Notification messages must contain the following name-value pairs:

```
Message
MessageId
Subject (if included in the message)
Timestamp
TopicArn
Type
```

The following example is a string to sign for a Notification.

```
Message
My Test Message
MessageId
4d4dc071-ddbf-465d-bba8-08f81c89da64
```

```
Subject
My subject
Timestamp
2019-01-31T04:37:04.321Z
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXTCOP
Type
Notification
```

### SubscriptionConfirmation and UnsubscribeConfirmation

SubscriptionConfirmation and UnsubscribeConfirmation messages must contain the following name-value pairs:

```
Message
MessageId
SubscribeURL
Timestamp
Token
TopicArn
Type
```

The following example is a string to sign for a SubscriptionConfirmation.

```
Message
My Test Message
MessageId
3d891288-136d-417f-bc05-901c108273ee
SubscribeURL
https://sns.us-east-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-2:123456789012:s4-
MySNSTopic-1G1WEFCOXTCOP&Token=233...
Timestamp
2019-01-31T19:25:13.719Z
Token
233...
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXTCOP
Type
SubscriptionConfirmation
```

6. Decode the Signature value from Base64 format. The message delivers the signature in the Signature value, which is encoded as Base64. Before you compare the signature value with the signature you have calculated, make sure that you decode the Signature value from Base64 so that you compare the values using the same format.
7. Generate the derived hash value of the Amazon SNS message. Submit the Amazon SNS message, in canonical format, to the same hash function used to generate the signature.
8. Generate the asserted hash value of the Amazon SNS message. The asserted hash value is the result of using the public key value (from step 3) to decrypt the signature delivered with the Amazon SNS message.
9. Verify the authenticity and integrity of the Amazon SNS message. Compare the derived hash value (from step 7) to the asserted hash value (from step 8). If the values are identical, then the receiver is assured that the message has not been modified while in transit and the message must have originated from Amazon SNS. If the values are not identical, it should not be trusted by the receiver.

## Parsing message formats

Amazon SNS uses the following formats.

## Topics

- [HTTP/HTTPS headers \(p. 143\)](#)
- [HTTP/HTTPS subscription confirmation JSON format \(p. 143\)](#)
- [HTTP/HTTPS notification JSON format \(p. 145\)](#)
- [HTTP/HTTPS unsubscribe confirmation JSON format \(p. 146\)](#)
- [SetSubscriptionAttributes delivery policy JSON format \(p. 147\)](#)
- [SetTopicAttributes delivery policy JSON format \(p. 148\)](#)

## HTTP/HTTPS headers

When Amazon SNS sends a subscription confirmation, notification, or unsubscribe confirmation message to HTTP/HTTPS endpoints, it sends a POST message with a number of Amazon SNS-specific header values. You can use these header values to do things such as identify the type of message without having to parse the JSON message body to read the `Type` value.

### **x-amz-sns-message-type**

The type of message. The possible values are `SubscriptionConfirmation`, `Notification`, and `UnsubscribeConfirmation`.

### **x-amz-sns-message-id**

A Universally Unique Identifier, unique for each message published. For a notification that Amazon SNS resends during a retry, the message ID of the original message is used.

### **x-amz-sns-topic-arn**

The Amazon Resource Name (ARN) for the topic that this message was published to.

### **x-amz-sns-subscription-arn**

The ARN for the subscription to this endpoint.

The following HTTP POST header is an example of a header for a Notification message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
```

## HTTP/HTTPS subscription confirmation JSON format

After you subscribe an HTTP/HTTPS endpoint, Amazon SNS sends a subscription confirmation message to the HTTP/HTTPS endpoint. This message contains a `SubscribeURL` value that you must visit to confirm the subscription (alternatively, you can use the `Token` value with the [ConfirmSubscription](#)).

### Note

Amazon SNS doesn't send notifications to this endpoint until the subscription is confirmed

The subscription confirmation message is a POST message with a message body that contains a JSON document with the following name-value pairs.

### **Message**

A string that describes the message. For subscription confirmation, this string looks like this:

```
You have chosen to subscribe to the topic arn:aws:sns:us-east-2:123456789012:MyTopic.  
\nTo confirm the subscription, visit the SubscribeURL included in this message.
```

### **MessageId**

A Universally Unique Identifier, unique for each message published. For a message that Amazon SNS resends during a retry, the message ID of the original message is used.

### **Signature**

Base64-encoded "SHA1withRSA" signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

### **SignatureVersion**

Version of the Amazon SNS signature used.

### **SigningCertURL**

The URL to the certificate that was used to sign the message.

### **SubscribeURL**

The URL that you must visit in order to confirm the subscription. Alternatively, you can instead use the Token with the [ConfirmSubscription](#) action to confirm the subscription.

### **Timestamp**

The time (GMT) when the subscription confirmation was sent.

### **Token**

A value you can use with the [ConfirmSubscription](#) action to confirm the subscription. Alternatively, you can simply visit the SubscribeURL.

### **TopicArn**

The Amazon Resource Name (ARN) for the topic that this endpoint is subscribed to.

### **Type**

The type of message. For a subscription confirmation, the type is SubscriptionConfirmation.

The following HTTP POST message is an example of a SubscriptionConfirmation message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
    "Type" : "SubscriptionConfirmation",
    "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
    "Token" : "2336412f37...",
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
    "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL included in this message.",
```

```
"SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37...",  
"Timestamp" : "2012-04-26T20:45:04.751Z",  
"SignatureVersion" : "1",  
"Signature" : "EXAMPLEPh+DcEwjAPg809mY8dReBSwksfg2S7WKQcikcNKWLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDVrY0Ad8L10Hs3zH8f3ecfb7224c7233fe7bb5f59f96de52f.pem"  
}
```

## HTTP/HTTPS notification JSON format

When Amazon SNS sends a notification to a subscribed HTTP or HTTPS endpoint, the POST message sent to the endpoint has a message body that contains a JSON document with the following name-value pairs.

### **Message**

The Message value specified when the notification was published to the topic.

### **MessageId**

A Universally Unique Identifier, unique for each message published. For a notification that Amazon SNS resends during a retry, the message ID of the original message is used.

### **Signature**

Base64-encoded SHA1withRSA signature of the Message, MessageId, Subject (if present), Type, Timestamp, and TopicArn values.

### **SignatureVersion**

Version of the Amazon SNS signature used.

### **SigningCertURL**

The URL to the certificate that was used to sign the message.

### **Subject**

The Subject parameter specified when the notification was published to the topic.

#### **Note**

This is an optional parameter. If no Subject was specified, then this name-value pair does not appear in this JSON document.

### **Timestamp**

The time (GMT) when the notification was published.

### **TopicArn**

The Amazon Resource Name (ARN) for the topic that this message was published to.

### **Type**

The type of message. For a notification, the type is Notification.

### **UnsubscribeURL**

A URL that you can use to unsubscribe the endpoint from this topic. If you visit this URL, Amazon SNS unsubscribes the endpoint and stops sending notifications to this endpoint.

The following HTTP POST message is an example of a Notification message to an HTTP endpoint.

```
POST / HTTP/1.1  
x-amz-sns-message-type: Notification
```

```
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
    "Type" : "Notification",
    "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
    "Subject" : "My First Message",
    "Message" : "Hello world!",
    "Timestamp" : "2012-05-02T00:54:06.655Z",
    "SignatureVersion" : "1",
    "Signature" : "EXAMPLEwJRN...",
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
    "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

## HTTP/HTTPS unsubscribe confirmation JSON format

After an HTTP/HTTPS endpoint is unsubscribed from a topic, Amazon SNS sends an unsubscribe confirmation message to the endpoint.

The unsubscribe confirmation message is a POST message with a message body that contains a JSON document with the following name-value pairs.

### **Message**

A string that describes the message. For unsubscribe confirmation, this string looks like this:

```
You have chosen to deactivate subscription arn:aws:sns:us-
east-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this
operation and restore the subscription, visit the SubscribeURL included in this
message.
```

### **MessageId**

A Universally Unique Identifier, unique for each message published. For a message that Amazon SNS resends during a retry, the message ID of the original message is used.

### **Signature**

Base64-encoded "SHA1withRSA" signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

### **SignatureVersion**

Version of the Amazon SNS signature used.

### **SigningCertURL**

The URL to the certificate that was used to sign the message.

### **SubscribeURL**

The URL that you must visit in order to re-confirm the subscription. Alternatively, you can instead use the Token with the [ConfirmSubscription](#) action to re-confirm the subscription.

**Timestamp**

The time (GMT) when the unsubscribe confirmation was sent.

**Token**

A value you can use with the [ConfirmSubscription](#) action to re-confirm the subscription.  
Alternatively, you can simply visit the [SubscribeURL](#).

**TopicArn**

The Amazon Resource Name (ARN) for the topic that this endpoint has been unsubscribed from.

**Type**

The type of message. For a unsubscribe confirmation, the type is `UnsubscribeConfirmation`.

The following HTTP POST message is an example of a `UnsubscribeConfirmation` message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: UnsubscribeConfirmation
x-amz-sns-message-id: 47138184-6831-46b8-8f7c-afc488602d7d
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1399
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
    "Type" : "UnsubscribeConfirmation",
    "MessageId" : "47138184-6831-46b8-8f7c-afc488602d7d",
    "Token" : "2336412f37...",
    "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
    "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this operation and restore the subscription, visit the SubscribeURL included in this message.",
    "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37fb6...",
    "Timestamp" : "2012-04-26T20:06:41.581Z",
    "SignatureVersion" : "1",
    "Signature" : "EXAMPLEHXgJm...",
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

## SetSubscriptionAttributes delivery policy JSON format

If you send a request to the `SetSubscriptionAttributes` action and set the `AttributeName` parameter to a value of `DeliveryPolicy`, the value of the `AttributeValue` parameter must be a valid JSON object. For example, the following example sets the delivery policy to 5 total retries.

```
http://sns.us-east-2.amazonaws.com/
?Action=SetSubscriptionAttributes
&SubscriptionArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic
%3A80289ba6-0fd4-4079-afb4-ce8c8260f0ca
&AttributeName=DeliveryPolicy
&AttributeValue={"healthyRetryPolicy":{"numRetries":5}}
...
```

Use the following JSON format for the value of the AttributeValue parameter.

```
{  
    "healthyRetryPolicy" : {  
        "minDelayTarget" : int,  
        "maxDelayTarget" : int,  
        "numRetries" : int,  
        "numMaxDelayRetries" : int,  
        "backoffFunction" : "linear/arithmetic/geometric/exponential"  
    },  
    "throttlePolicy" : {  
        "maxReceivesPerSecond" : int  
    }  
}
```

For more information about the SetSubscriptionAttribute action, go to [SetSubscriptionAttributes](#) in the *Amazon Simple Notification Service API Reference*.

## SetTopicAttributes delivery policy JSON format

If you send a request to the SetTopicAttributes action and set the AttributeName parameter to a value of *DeliveryPolicy*, the value of the AttributeValue parameter must be a valid JSON object. For example, the following example sets the delivery policy to 5 total retries.

```
http://sns.us-east-2.amazonaws.com/  
?Action=SetTopicAttributes  
&TopicArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic  
&AttributeName=DeliveryPolicy  
&AttributeValue={"http":{"defaultHealthyRetryPolicy":{"numRetries":5}}}  
...
```

Use the following JSON format for the value of the AttributeValue parameter.

```
{  
    "http" : {  
        "defaultHealthyRetryPolicy" : {  
            "minDelayTarget": int,  
            "maxDelayTarget": int,  
            "numRetries": int,  
            "numMaxDelayRetries": int,  
            "backoffFunction": "linear/arithmetic/geometric/exponential"  
        },  
        "disableSubscriptionOverrides" : Boolean,  
        "defaultThrottlePolicy" : {  
            "maxReceivesPerSecond" : int  
        }  
    }  
}
```

For more information about the SetTopicAttribute action, go to [SetTopicAttributes](#) in the *Amazon Simple Notification Service API Reference*.

# Fanout to AWS Event Fork Pipelines

For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Kinesis Data Firehose. You can subscribe Kinesis Data Firehose delivery streams to SNS

topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Kinesis Data Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#).

You can use Amazon SNS to build event-driven applications which use subscriber services to perform work automatically in response to events triggered by publisher services. This architectural pattern can make services more reusable, interoperable, and scalable. However, it can be labor-intensive to fork the processing of events into pipelines that address common event handling requirements, such as event storage, backup, search, analytics, and replay.

To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source [nested applications](#), based on the [AWS Serverless Application Model \(AWS SAM\)](#), which you can deploy directly from the [AWS Event Fork Pipelines suite](#) (choose **Show apps that create custom IAM roles or resource policies**) into your AWS account.

For an AWS Event Fork Pipelines use case, see [Deploying and testing the AWS Event Fork Pipelines sample application \(p. 153\)](#).

#### Topics

- [How AWS Event Fork Pipelines works \(p. 149\)](#)
- [Deploying AWS Event Fork Pipelines \(p. 152\)](#)
- [Deploying and testing the AWS Event Fork Pipelines sample application \(p. 153\)](#)
- [Subscribing an AWS Event Fork Pipelines to an Amazon SNS topic \(p. 159\)](#)

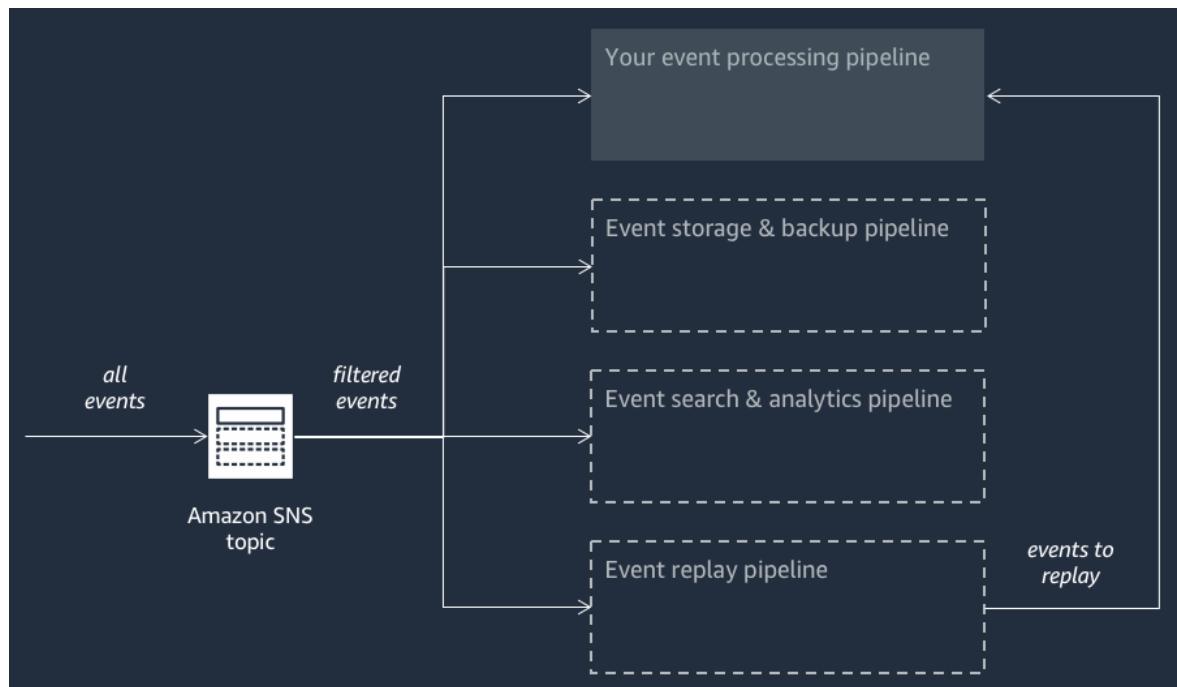
## How AWS Event Fork Pipelines works

AWS Event Fork Pipelines is a serverless design pattern. However, it is also a suite of nested serverless applications based on AWS SAM (which you can deploy directly from the AWS Serverless Application Repository (AWS SAR) to your AWS account in order to enrich your event-driven platforms). You can deploy these nested applications individually, as your architecture requires.

#### Topics

- [The event storage and backup pipeline \(p. 150\)](#)
- [The event search and analytics pipeline \(p. 151\)](#)
- [The event replay pipeline \(p. 151\)](#)

The following diagram shows an AWS Event Fork Pipelines application supplemented by three nested applications. You can deploy any of the pipelines from the AWS Event Fork Pipelines suite on the AWS SAR independently, as your architecture requires.



Each pipeline is subscribed to the same Amazon SNS topic, allowing itself to process events in parallel as these events are published to the topic. Each pipeline is independent and can set its own [Subscription Filter Policy \(p. 71\)](#). This allows a pipeline to process only a subset of the events that it is interested in (rather than all events published to the topic).

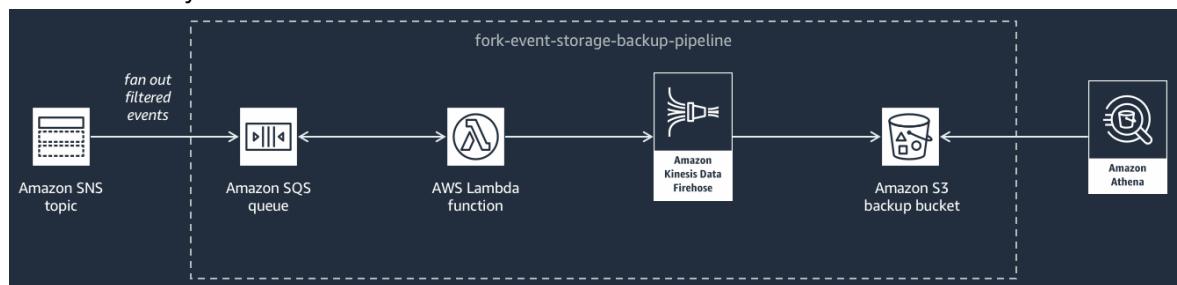
#### Note

Because you place the three AWS Event Fork Pipelines alongside your regular event processing pipelines (possibly already subscribed to your Amazon SNS topic), you don't need to change any portion of your current message publisher to take advantage of AWS Event Fork Pipelines in your existing workloads.

## The event storage and backup pipeline

The following diagram shows the [Event Storage and Backup Pipeline](#). You can subscribe this pipeline to your Amazon SNS topic to automatically back up the events flowing through your system.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, an AWS Lambda function that automatically polls for these events in the queue and pushes them into an Amazon Kinesis Data Firehose stream, and an Amazon S3 bucket that durably backs up the events loaded by the stream.



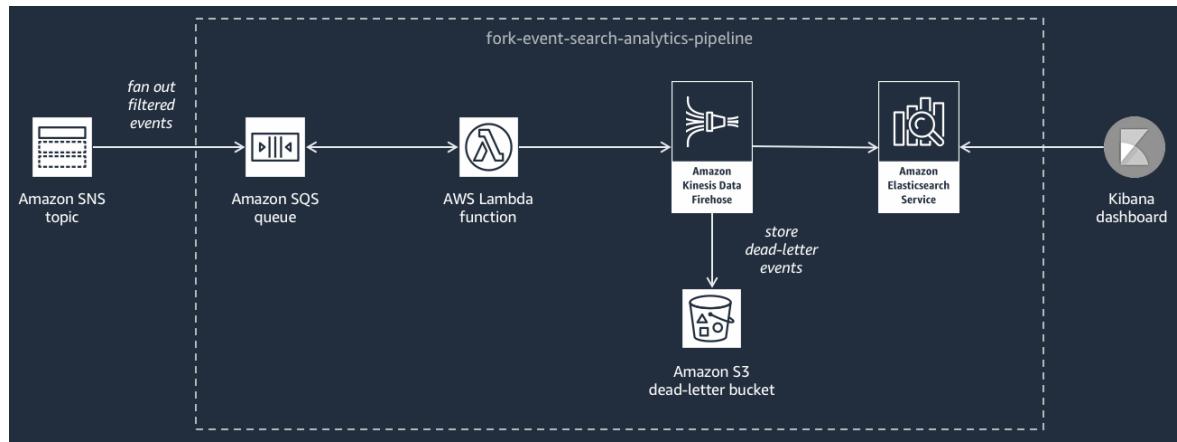
To fine-tune the behavior of your Firehose stream, you can configure it to buffer, transform, and compress your events prior to loading them into the bucket. As events are loaded, you can use Amazon

Athena to query the bucket using standard SQL queries. You can also configure the pipeline to reuse an existing Amazon S3 bucket or create a new one.

## The event search and analytics pipeline

The following diagram shows the [Event Search and Analytics Pipeline](#). You can subscribe this pipeline to your Amazon SNS topic to index the events that flow through your system in a search domain and then run analytics on them.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, an AWS Lambda function that polls events from the queue and pushes them into an Amazon Kinesis Data Firehose stream, an Amazon OpenSearch Service domain that indexes the events loaded by the Firehose stream, and an Amazon S3 bucket that stores the dead-letter events that can't be indexed in the search domain.



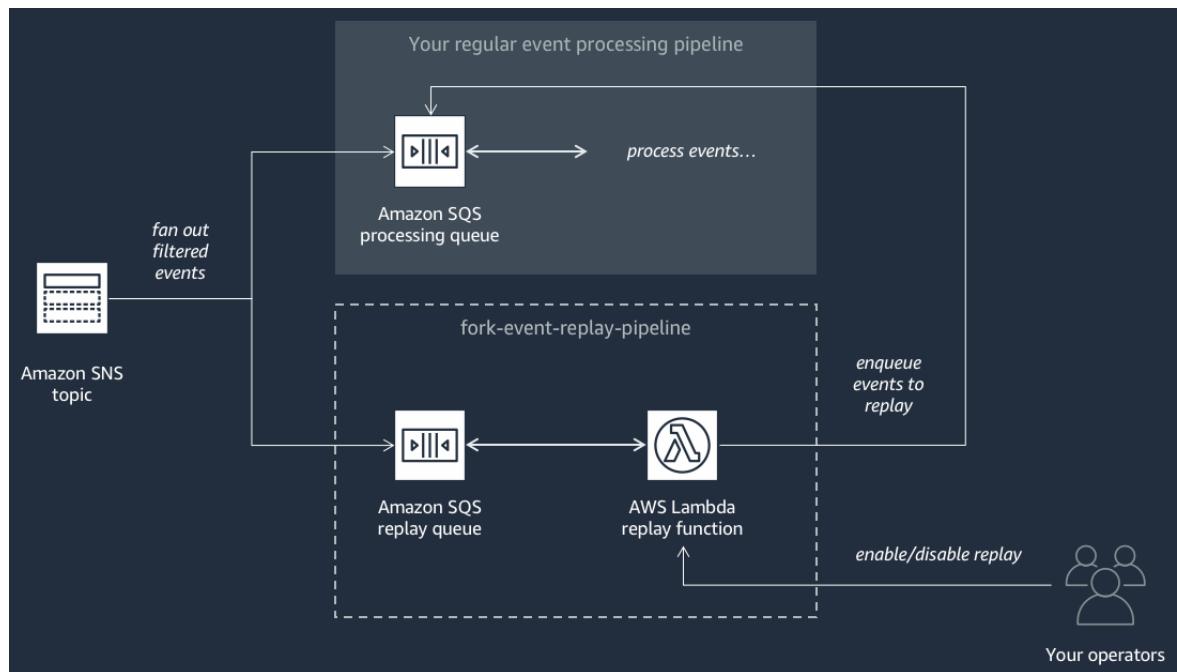
To fine-tune your Firehose stream in terms of event buffering, transformation, and compression, you can configure this pipeline.

You can also configure whether the pipeline should reuse an existing OpenSearch domain in your AWS account or create a new one for you. As events are indexed in the search domain, you can use Kibana to run analytics on your events and update visual dashboards in real-time.

## The event replay pipeline

The following diagram shows the [Event Replay Pipeline](#). To record the events that have been processed by your system for the past 14 days (for example when your platform needs to recover from failure), you can subscribe this pipeline to your Amazon SNS topic and then reprocess the events.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, and an AWS Lambda function that polls events from the queue and redrives them into your regular event processing pipeline, which is also subscribed to your topic.



#### Note

By default, the replay function is disabled, not redriving your events. If you need to reprocess events, you must enable the Amazon SQS replay queue as an event source for the AWS Lambda replay function.

## Deploying AWS Event Fork Pipelines

The [AWS Event Fork Pipelines suite](#) (choose **Show apps that create custom IAM roles or resource policies**) is available as a group of public applications in the AWS Serverless Application Repository, from where you can deploy and test them manually using the [AWS Lambda console](#). For information about deploying pipelines using the AWS Lambda console, see [Subscribing an AWS Event Fork Pipelines to an Amazon SNS topic \(p. 159\)](#).

In a production scenario, we recommend embedding AWS Event Fork Pipelines within your overall application's AWS SAM template. The nested-application feature lets you do this by adding the resource [AWS::Serverless::Application](#) to your AWS SAM template, referencing the AWS SAR `ApplicationId` and the `SemanticVersion` of the nested application.

For example, you can use the Event Storage and Backup Pipeline as a nested application by adding the following YAML snippet to the `Resources` section of your AWS SAM template.

```
Backup:
  Type: AWS::Serverless::Application
  Properties:
    Location:
      ApplicationId: arn:aws:serverlessrepo:us-east-2:123456789012:applications/fork-event-
storage-backup-pipeline
    SemanticVersion: 1.0.0
  Parameters:
    #The ARN of the Amazon SNS topic whose messages should be backed up to the Amazon S3
    bucket.
    TopicArn: !Ref MySNSTopic
```

When you specify parameter values, you can use AWS CloudFormation intrinsic functions to reference other resources in your template. For example, in the YAML snippet above, the `TopicArn` parameter

references the [AWS::SNS::Topic](#) resource `MySNSTopic`, defined elsewhere in the AWS SAM template. For more information, see the [Intrinsic Function Reference](#) in the *AWS CloudFormation User Guide*.

**Note**

The AWS Lambda console page for your AWS SAR application includes the **Copy as SAM Resource** button, which copies the YAML required for nesting an AWS SAR application to the clipboard.

## Deploying and testing the AWS Event Fork Pipelines sample application

To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source [nested applications](#), based on the [AWS Serverless Application Model](#) (AWS SAM), which you can deploy directly from the [AWS Event Fork Pipelines suite](#) (choose **Show apps that create custom IAM roles or resource policies**) into your AWS account. For more information, see [How AWS Event Fork Pipelines works \(p. 149\)](#).

This page shows how you can use the AWS Management Console to deploy and test the AWS Event Fork Pipelines sample application.

**Important**

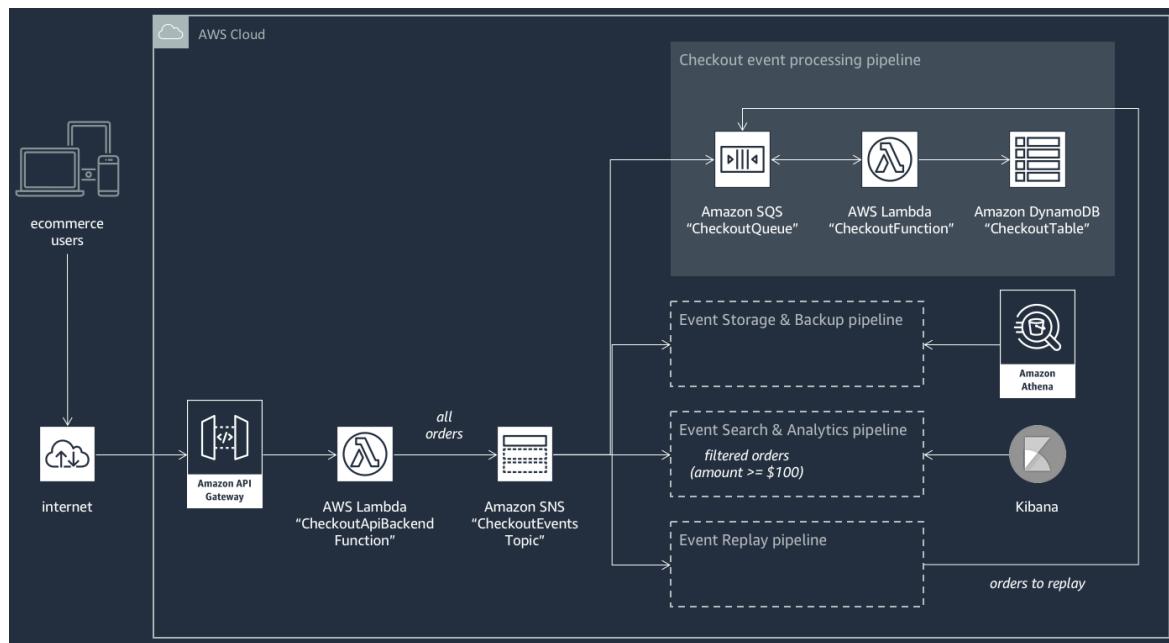
To avoid incurring unwanted costs after you finish deploying the AWS Event Fork Pipelines sample application, delete its AWS CloudFormation stack. For more information, see [Deleting a Stack on the AWS CloudFormation Console](#) in the *AWS CloudFormation User Guide*.

**Topics**

- [Example AWS Event Fork Pipelines use case \(p. 153\)](#)
- [Step 1: To deploy the sample application \(p. 155\)](#)
- [Step 2: To execute the sample application \(p. 156\)](#)
- [Step 3: To verify the execution of the sample application and its pipelines \(p. 157\)](#)
- [Step 4: To simulate an issue and replay events for recovery \(p. 158\)](#)

## Example AWS Event Fork Pipelines use case

The following scenario describes an event-driven, serverless e-commerce application that uses AWS Event Fork Pipelines. You can use this [example e-commerce application](#) in the AWS Serverless Application Repository and then deploy it in your AWS account using the AWS Lambda console, where you can test it and examine its source code in GitHub.



This e-commerce application takes orders from buyers through a RESTful API hosted by API Gateway and backed by the AWS Lambda function `CheckoutApiBackendFunction`. This function publishes all received orders to an Amazon SNS topic named `CheckoutEventsTopic` which, in turn, fans out the orders to four different pipelines.

The first pipeline is the regular checkout-processing pipeline designed and implemented by the owner of the e-commerce application. This pipeline has the Amazon SQS queue `CheckoutQueue` that buffers all received orders, an AWS Lambda function named `CheckoutFunction` that polls the queue to process these orders, and the DynamoDB table `CheckoutTable` that securely saves all placed orders.

## Applying AWS Event Fork Pipelines

The components of the e-commerce application handle the core business logic. However, the e-commerce application owner also needs to address the following:

- **Compliance**—secure, compressed backups encrypted at rest and sanitization of sensitive information
- **Resiliency**—replay of most recent orders in case of the disruption of the fulfillment process
- **Searchability**—running analytics and generating metrics on placed orders

Instead of implementing this event processing logic, the application owner can subscribe AWS Event Fork Pipelines to the `CheckoutEventsTopic` Amazon SNS topic

- **The event storage and backup pipeline (p. 150)** is configured to transform data to remove credit card details, buffer data for 60 seconds, compress it using GZIP, and encrypt it using the default Customer Master Key (CMK) for Amazon S3. This CMK is managed by AWS and powered by the AWS Key Management Service (AWS KMS).

For more information, see [Choose Amazon S3 For Your Destination](#), [Amazon Kinesis Data Firehose Data Transformation](#), and [Configure Settings](#) in the *Amazon Kinesis Data Firehose Developer Guide*.

- **The event search and analytics pipeline (p. 151)** is configured with an index retry duration of 30 seconds, a bucket for storing orders that fail to be indexed in the search domain, and a filter policy to restrict the set of indexed orders.

For more information, see [Choose OpenSearch Service for your Destination](#) in the *Amazon Kinesis Data Firehose Developer Guide*.

- The event replay pipeline (p. 151) is configured with the Amazon SQS queue part of the regular order-processing pipeline designed and implemented by the e-commerce application owner.

For more information, see [Queue Name and URL](#) in the *Amazon Simple Queue Service Developer Guide*.

The following JSON filter policy is set in the configuration for the Event Search and Analytics Pipeline. It matches only incoming orders in which the total amount is \$100 or higher. For more information, see [Amazon SNS message filtering](#) (p. 71).

```
{  
    "amount": [{ "numeric": [ ">=", 100 ] }]  
}
```

Using the AWS Event Fork Pipelines pattern, the e-commerce application owner can avoid the development overhead that often follows coding undifferentiating logic for event handling. Instead, she can deploy AWS Event Fork Pipelines directly from the AWS Serverless Application Repository into her AWS account.

## Step 1: To deploy the sample application

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:
  - a. Choose **Browse serverless app repository**, **Public applications**, **Show apps that create custom IAM roles or resource policies**.
  - b. Search for `fork-example-ecommerce-checkout-api` and then choose the application.
4. On the `fork-example-ecommerce-checkout-api` page, do the following:
  - a. In the **Application settings** section, enter an **Application name** (for example, `fork-example-ecommerce-my-app`).

### Note

- To find your resources easily later, keep the prefix `fork-example-ecommerce`.
  - For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).
- b. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
    - DEBUG
    - ERROR
    - INFO (default)
    - WARNING
  5. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications**. and then, at the bottom of the page, choose **Deploy**.

On the **Deployment status for fork-example-ecommerce-my-app** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE\_IN\_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

**Note**

It might take 20-30 minutes for all resources to be deployed.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

## Step 2: To execute the sample application

1. In the AWS Lambda console, on the navigation panel, choose **Applications**.
2. On the **Applications** page, in the search field, search for `serverlessrepo-fork-example-ecommerce-my-app` and then choose the application.
3. In the **Resources** section, do the following:
  - a. To find the resource whose type is **ApiGateway RestApi**, sort the resources by **Type**, for example `ServerlessRestApi`, and then expand the resource.
  - b. Two nested resources are displayed, of types **ApiGateway Deployment** and **ApiGateway Stage**.
  - c. Copy the link **Prod API endpoint** and append `/checkout` to it, for example:

```
https://abcdefgij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

4. Copy the following JSON to a file named `test_event.json`.

```
{
    "id": 15311,
    "date": "2019-03-25T23:41:11-08:00",
    "status": "confirmed",
    "customer": {
        "id": 65144,
        "name": "John Doe",
        "email": "john.doe@example.com"
    },
    "payment": {
        "id": 2509,
        "amount": 450.00,
        "currency": "usd",
        "method": "credit",
        "card-network": "visa",
        "card-number": "1234 5678 9012 3456",
        "card-expiry": "10/2022",
        "card-owner": "John Doe",
        "card-cvv": "123"
    },
    "shipping": {
        "id": 7600,
        "time": 2,
        "unit": "days",
        "method": "courier"
    },
    "items": [
        {
            "id": 6512,
            "product": 8711,
            "name": "Hockey Jersey - Large",
            "quantity": 1,
            "price": 400.00,
            "subtotal": 400.00
        },
        {
            "id": 9954,
            "product": 7600,
            "name": "Hockey Jersey - Large",
            "quantity": 1,
            "price": 400.00,
            "subtotal": 400.00
        }
    ]
}
```

```
        "name": "Hockey Puck",
        "quantity": 2,
        "price": 25.00,
        "subtotal": 50.00
    }]
}
```

5. To send an HTTPS request to your API endpoint, pass the sample event payload as input by executing a `curl` command, for example:

```
curl -d "$(cat test_event.json)" https://abcdefgij.execute-api.us-
east-2.amazonaws.com/Prod/checkout
```

The API returns the following empty response, indicating a successful execution:

```
{ }
```

## Step 3: To verify the execution of the sample application and its pipelines

### Step 1: To verify the execution of the sample checkout pipeline

1. Sign in to the [Amazon DynamoDB console](#).
2. On the navigation panel, choose **Tables**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutTable`.
4. On the table details page, choose **Items** and then choose the created item.

The stored attributes are displayed.

### Step 2: To verify the execution of the event storage and backup pipeline

1. Sign in to the [Amazon S3 console](#).
2. On the navigation panel, choose **Buckets**.
3. Search for `serverlessrepo-fork-example` and then choose `CheckoutBucket`.
4. Navigate the directory hierarchy until you find a file with the extension `.gz`.
5. To download the file, choose **Actions, Open**.
6. The pipeline is configured with a Lambda function that sanitizes credit card information for compliance reasons.

To verify that the stored JSON payload doesn't contain any credit card information, decompress the file.

### Step 3: To verify the execution of the event search and analytics pipeline

1. Sign in to the [Amazon OpenSearch Service console](#).
2. On the navigation panel, under **My domains**, choose the domain prefixed with `server1-analyt`.
3. The pipeline is configured with an Amazon SNS subscription filter policy that sets a numeric matching condition.

To verify that the event is indexed because it refers to an order whose value is higher than USD \$100, on the `server1-analyt-abcdefghijklm` page, choose **Indices, checkout\_events**.

## Step 4: To verify the execution of the event replay pipeline

1. Sign in to the [Amazon SQS console](#).
2. In the list of queues, search for `serverlessrepo-fork-example` and choose `ReplayQueue`.
3. Choose **Queue Actions, View/Delete Messages**.
4. In the **View/Delete Messages in fork-example-ecommerce-my-app...ReplayP-ReplayQueue-123ABCD4E5F6** dialog box, choose **Start Polling for Messages**.
5. To verify that the event is enqueued, choose **More Details** next to the message that appears in the queue.

## Step 4: To simulate an issue and replay events for recovery

### Step 1: To enable the simulated issue and send a second API request

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutFunction`.
4. On the **fork-example-ecommerce-my-app-CheckoutFunction-ABCDEF**... page, in the **Environment variables** section, set the **BUG\_ENABLED** variable to **true** and then choose **Save**.
5. Copy the following JSON to a file named `test_event_2.json`.

```
{  
    "id": 9917,  
    "date": "2019-03-26T21:11:10-08:00",  
    "status": "confirmed",  
    "customer": {  
        "id": 56999,  
        "name": "Marcia Oliveira",  
        "email": "marcia.oliveira@example.com"  
    },  
    "payment": {  
        "id": 3311,  
        "amount": 75.00,  
        "currency": "usd",  
        "method": "credit",  
        "card-network": "mastercard",  
        "card-number": "1234 5678 9012 3456",  
        "card-expiry": "12/2025",  
        "card-owner": "Marcia Oliveira",  
        "card-cvv": "321"  
    },  
    "shipping": {  
        "id": 9900,  
        "time": 20,  
        "unit": "days",  
        "method": "plane"  
    },  
    "items": [  
        {  
            "id": 9993,  
            "product": 3120,  
            "name": "Hockey Stick",  
            "quantity": 1,  
            "price": 75.00,  
            "subtotal": 75.00  
        }  
    ]  
}
```

6. To send an HTTPS request to your API endpoint, pass the sample event payload as input by executing a `curl` command, for example:

```
curl -d "$(cat test_event_2.json)" https://abcdefgij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

The API returns the following empty response, indicating a successful execution:

```
{ }
```

## Step 2: To verify simulated data corruption

1. Sign in to the [Amazon DynamoDB console](#).
2. On the navigation panel, choose **Tables**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutTable`.
4. On the table details page, choose **Items** and then choose the created item.

The stored attributes are displayed, some marked as **CORRUPTED!**

## Step 3: To disable the simulated issue

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutFunction`.
4. On the `fork-example-ecommerce-my-app-CheckoutFunction-ABCDEF...` page, in the **Environment variables** section, set the `BUG_ENABLED` variable to `false` and then choose **Save**.

## Step 4: To enable replay to recover from the issue

1. In the AWS Lambda console, on the navigation panel, choose **Functions**.
2. Search for `serverlessrepo-fork-example` and choose `ReplayFunction`.
3. Expand the **Designer** section, choose the **SQS** tile and then, in the **SQS** section, choose **Enabled**.  
**Note**  
It takes approximately 1 minute for the Amazon SQS event source trigger to become enabled.
4. Choose **Save**.
5. To view the recovered attributes, return to the Amazon DynamoDB console.
6. To disable replay, return to the AWS Lambda console and disable the Amazon SQS event source trigger for `ReplayFunction`.

## Subscribing an AWS Event Fork Pipelines to an Amazon SNS topic

To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source [nested applications](#), based on the [AWS Serverless Application Model](#) (AWS SAM), which you can deploy directly from the [AWS Event Fork Pipelines suite](#) (choose **Show apps that create**

custom IAM roles or resource policies) into your AWS account. For more information, see [How AWS Event Fork Pipelines works \(p. 149\)](#).

This section shows how you can use the AWS Management Console to deploy a pipeline and then subscribe AWS Event Fork Pipelines to an Amazon SNS topic. Before you begin, [create an Amazon SNS topic \(p. 24\)](#).

To delete the resources that comprise a pipeline, find the pipeline on the **Applications** page of the AWS Lambda console, expand the **SAM template section**, choose **CloudFormation stack**, and then choose **Other Actions, Delete Stack**.

#### Topics

- [To deploy and subscribe the event storage and backup pipeline \(p. 160\)](#)
- [To deploy and subscribe the event search and analytics pipeline \(p. 162\)](#)
- [To deploy and subscribe the event replay pipeline \(p. 164\)](#)

## To deploy and subscribe the event storage and backup pipeline

For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Kinesis Data Firehose. You can subscribe Kinesis Data Firehose delivery streams to SNS topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Kinesis Data Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#).

This page shows how to deploy the [Event Storage and Backup Pipeline \(p. 150\)](#) and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Storage and Backup Pipeline, including the following:

- Amazon SQS queue
- Lambda function
- Kinesis Data Firehose delivery stream
- Amazon S3 backup bucket

For more information about configuring a stream with an S3 bucket as a destination, see [S3DestinationConfiguration](#) in the [Amazon Kinesis Data Firehose API Reference](#).

For more information about transforming events and about configuring event buffering, event compression, and event encryption, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#) in the [Amazon Kinesis Data Firehose Developer Guide](#).

For more information about filtering events, see [Amazon SNS subscription filter policies \(p. 71\)](#) in this guide.

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:
  - a. Choose **Browse serverless app repository, Public applications, Show apps that create custom IAM roles or resource policies**.

- b. Search for **fork-event-storage-backup-pipeline** and then choose the application.
4. On the **fork-event-storage-backup-pipeline** page, do the following:
  - a. In the **Application settings** section, enter an **Application name** (for example, `my-app-backup`).

**Note**

- For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).
- b. (Optional) For **BucketArn**, enter the ARN of the S3 bucket into which incoming events are loaded. If you don't enter a value, a new S3 bucket is created in your AWS account.
- c. (Optional) For **DataTransformationFunctionArn**, enter the ARN of the Lambda function through which the incoming events are transformed. If you don't enter a value, data transformation is disabled.
- d. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
  - DEBUG
  - ERROR
  - INFO (default)
  - WARNING
- e. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.
- f. (Optional) For **StreamBufferingIntervalInSeconds** and **StreamBufferingSizeInMBs**, enter the values for configuring the buffering of incoming events. If you don't enter any values, 300 seconds and 5 MB are used.
- g. (Optional) Enter one of the following **StreamCompressionFormat** settings for compressing incoming events:
  - GZIP
  - SNAPPY
  - UNCOMPRESSED (default)
  - ZIP
- h. (Optional) For **StreamPrefix**, enter the string prefix to name files stored in the S3 backup bucket. If you don't enter a value, no prefix is used.
- i. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are stored in the S3 backup bucket. If you don't enter a value, no filtering is used (all events are stored).
- j. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status for `my-app`** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE\_IN\_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are stored in the S3 backup bucket provisioned by the Event Storage and Backup pipeline automatically.

## To deploy and subscribe the event search and analytics pipeline

For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Kinesis Data Firehose. You can subscribe Kinesis Data Firehose delivery streams to SNS topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Kinesis Data Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see [Fanout to Kinesis Data Firehose delivery streams \(p. 103\)](#).

This page shows how to deploy the [Event Search and Analytics Pipeline \(p. 151\)](#) and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Search and Analytics Pipeline, including the following:

- Amazon SQS queue
- Lambda function
- Kinesis Data Firehose delivery stream
- Amazon OpenSearch Service domain
- Amazon S3 dead-letter bucket

For more information about configuring a stream with an index as a destination, see [ElasticsearchDestinationConfiguration](#) in the *Amazon Kinesis Data Firehose API Reference*.

For more information about transforming events and about configuring event buffering, event compression, and event encryption, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#) in the *Amazon Kinesis Data Firehose Developer Guide*.

For more information about filtering events, see [Amazon SNS subscription filter policies \(p. 71\)](#) in this guide.

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:
  - a. Choose **Browse serverless app repository**, **Public applications**, **Show apps that create custom IAM roles or resource policies**.
  - b. Search for **fork-event-search-analytics-pipeline** and then choose the application.
4. On the **fork-event-search-analytics-pipeline** page, do the following:
  - a. In the **Application settings** section, enter an **Application name** (for example, `my-app-search`).

### Note

For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

- b. (Optional) For **DataTransformationFunctionArn**, enter the ARN of the Lambda function used for transforming incoming events. If you don't enter a value, data transformation is disabled.
- c. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
  - DEBUG

- **ERROR**
  - **INFO (default)**
  - **WARNING**
- d. (Optional) For **SearchDomainArn**, enter the ARN of the OpenSearch Service domain, a cluster that configures the needed compute and storage functionality. If you don't enter a value, a new domain is created with the default configuration.
- e. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.
- f. For **SearchIndexName**, enter the name of the OpenSearch Service index for event search and analytics.

**Note**

The following quotas apply to index names:

- Can't include uppercase letters
  - Can't include the following characters: \ / \* ? " < > | ^ , #
  - Can't begin with the following characters: - + \_
  - Can't be the following: . ..
  - Can't be longer than 80 characters
  - Can't be longer than 255 bytes
  - Can't contain a colon (from OpenSearch Service 7.0)
- g. (Optional) Enter one of the following **SearchIndexRotationPeriod** settings for the rotation period of the OpenSearch Service index:
- **NoRotation (default)**
  - **OneDay**
  - **OneHour**
  - **OneMonth**
  - **OneWeek**

Index rotation appends a timestamp to the index name, facilitating the expiration of old data.

- h. For **SearchTypeName**, enter the name of the OpenSearch Service type for organizing the events in an index.

**Note**

- OpenSearch Service type names can contain any character (except null bytes) but can't begin with \_.
  - For OpenSearch Service 6.x, there can be only one type per index. If you specify a new type for an existing index that already has another type, Kinesis Data Firehose returns a runtime error.
- i. (Optional) For **StreamBufferingIntervalInSeconds** and **StreamBufferingSizeInMBs**, enter the values for configuring the buffering of incoming events. If you don't enter any values, 300 seconds and 5 MB are used.
- j. (Optional) Enter one of the following **StreamCompressionFormat** settings for compressing incoming events:
- **GZIP**
  - **SNAPPY**
  - **UNCOMPRESSED (default)**
  - **ZIP**

- k. (Optional) For **StreamPrefix**, enter the string prefix to name files stored in the S3 dead-letter bucket. If you don't enter a value, no prefix is used.
- l. (Optional) For **StreamRetryDurationInSeconds**, enter the retry duration for cases when Kinesis Data Firehose can't index events in the OpenSearch Service index. If you don't enter a value, then 300 seconds is used.
- m. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are indexed in the OpenSearch Service index. If you don't enter a value, no filtering is used (all events are indexed).
- n. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status for *my-app-search*** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE\_IN\_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are indexed in the OpenSearch Service index provisioned by the Event Search and Analytics pipeline automatically. If the pipeline can't index an event, it stores it in a S3 dead-letter bucket.

## To deploy and subscribe the event replay pipeline

This page shows how to deploy the [Event Replay Pipeline \(p. 151\)](#) and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Replay Pipeline, including an Amazon SQS queue and a Lambda function.

For more information about filtering events, see [Amazon SNS subscription filter policies \(p. 71\)](#) in this guide.

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:
  - a. Choose **Browse serverless app repository, Public applications, Show apps that create custom IAM roles or resource policies.**
  - b. Search for **fork-event-replay-pipeline** and then choose the application.
4. On the **fork-event-replay-pipeline** page, do the following:
  - a. In the **Application settings** section, enter an **Application name** (for example, *my-app-replay*).

### Note

For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

- b. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
  - DEBUG
  - ERROR

- **INFO** (default)
  - **WARNING**
- c. (Optional) For **ReplayQueueRetentionPeriodInSeconds**, enter the amount of time, in seconds, for which the Amazon SQS replay queue keeps the message. If you don't enter a value, 1,209,600 seconds (14 days) is used.
  - d. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.
  - e. For **DestinationQueueName**, enter the name of the Amazon SQS queue to which the Lambda replay function forwards messages.
  - f. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are buffered for replay. If you don't enter a value, no filtering is used (all events are buffered for replay).
  - g. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status** for *my-app-replay* page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE\_IN\_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are buffered for replay in the Amazon SQS queue provisioned by the Event Replay Pipeline automatically.

**Note**

By default, replay is disabled. To enable replay, navigate to the function's page on the Lambda console, expand the **Designer** section, choose the **SQS** tile and then, in the **SQS** section, choose **Enabled**.

# Using Amazon SNS for application-to-person (A2P) messaging

This section provides information about using Amazon SNS for user notifications with subscribers such as mobile applications, mobile phone numbers, and email addresses.

## Topics

- [Mobile text messaging \(SMS\) \(p. 166\)](#)
- [Mobile push notifications \(p. 234\)](#)
- [Email notifications \(p. 260\)](#)

## Mobile text messaging (SMS)

You can use Amazon SNS to send text messages, or *SMS messages*, to SMS-enabled devices. You can [send a message directly to a phone number \(p. 196\)](#), or you can [send a message to multiple phone numbers \(p. 192\)](#) at once by subscribing those phone numbers to a topic and sending your message to the topic.

You can [set SMS preferences \(p. 189\)](#) for your AWS account to tailor your SMS deliveries for your use cases and budget. For example, you can choose whether your messages are optimized for cost or reliable delivery. You can also specify spending quotas for individual message deliveries and monthly spending quotas for your AWS account.

Where required by local laws and regulations (such as the US and Canada), SMS recipients can [opt out \(p. 207\)](#), which means that they choose to stop receiving SMS messages from your AWS account. After a recipient opts out, you can, with limitations, opt in the phone number again so that you can resume sending messages to it.

Amazon SNS supports SMS messaging in several regions, and you can send messages to more than 200 countries and regions. For more information, see [Supported Regions and countries \(p. 219\)](#).

## Topics

- [SMS sandbox \(p. 167\)](#)
- [Origination identities for SMS messages \(p. 169\)](#)
- [Requesting support for SMS messaging with Amazon SNS \(p. 181\)](#)
- [Setting SMS messaging preferences \(p. 189\)](#)
- [Sending SMS messages \(p. 192\)](#)
- [Monitoring SMS activity \(p. 202\)](#)
- [Managing phone numbers and SMS subscriptions \(p. 207\)](#)
- [Supported Regions and countries \(p. 219\)](#)
- [SMS best practices \(p. 228\)](#)
- [Special requirements for sending SMS messages to US destinations \(p. 231\)](#)

- [Special requirements for sending SMS messages to recipients in India \(p. 231\)](#)

## SMS sandbox

When you start using Amazon SNS to send SMS messages, your AWS account is in the *SMS sandbox*. The SMS sandbox provides a safe environment for you to try Amazon SNS features without risking your reputation as an SMS sender. While your account is in the SMS sandbox, you can use all of the features of Amazon SNS, with the following restrictions:

- You can send SMS messages only to verified destination phone numbers.
- You can have up to 10 verified destination phone numbers.
- You can delete destination phone numbers only after 24 or more hours have passed since verification or the last verification attempt.

When your account is moved out of the sandbox, these restrictions are removed, and you can send SMS messages to any recipient.

### Topics

- [Adding and verifying phone numbers in the SMS sandbox \(p. 167\)](#)
- [Deleting phone numbers from the SMS sandbox \(p. 168\)](#)
- [Moving out of the SMS sandbox \(p. 168\)](#)

## Adding and verifying phone numbers in the SMS sandbox

To get started with sending SMS messages while your AWS account is in the [SMS sandbox \(p. 167\)](#), first add destination phone numbers, and then verify them.

### Note

As with accounts that aren't in the SMS sandbox, an [origination identity \(p. 169\)](#) is required before you can send SMS messages to recipients in some countries or regions. For more information, see [Supported Regions and countries \(p. 219\)](#). Origination IDs include [sender IDs \(p. 169\)](#) and different types of [origination numbers \(p. 170\)](#). To view your existing origination numbers, in the navigation pane of the [Amazon SNS console](#), choose **Origination numbers**. Currently, sender IDs don't appear in this list.

### To add and verify destination phone numbers

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, choose an [AWS Region that supports SMS messaging \(p. 219\)](#).
3. In the navigation pane, choose **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, under **Sandbox destination phone numbers**, choose **Add phone number**.
5. Under **Destination details**, enter the country code and phone number, specify what language to use for the verification message, and then choose **Add phone number**.

Amazon SNS sends a one-time password (OTP) to the destination phone number. If the destination phone number doesn't receive the OTP within 15 minutes, choose **Resend verification code**. You can send the OTP to the same destination phone number up to five times every 24 hours.

6. In the **Verification code** box, enter the OTP sent to the destination phone number, and then choose **Verify phone number**.

The destination phone number and its verification status appear in the **Sandbox destination phone numbers** section. If the verification status is **Pending**, verification was unsuccessful. This can happen, for example, if you didn't enter the country code with the phone number. You can delete pending or verified destination phone numbers only after 24 hours or more have passed since verification or the last verification attempt.

7. Repeat these steps in each Region where you want to use this destination phone number.

## Deleting phone numbers from the SMS sandbox

You can delete pending or verified destination phone numbers from the [SMS sandbox \(p. 167\)](#).

### To delete destination phone numbers from the SMS sandbox

1. Wait 24 hours after [verifying the phone number \(p. 167\)](#), or 24 hours after your last verification attempt.
2. Sign in to the [Amazon SNS console](#).
3. In the console menu, choose an [AWS Region that supports SMS messaging \(p. 219\)](#) where you added a destination phone number.
4. In the navigation pane, choose **Text messaging (SMS)**.
5. On the **Mobile text messaging (SMS)** page, under **Sandbox destination phone numbers**, choose the phone number to delete, and then choose **Delete phone number**.
6. To confirm that you want to delete the phone number, enter **delete me**, and then choose **Delete**.

If 24 hours or more have passed since you verified or attempted to verify the destination phone number, it is deleted, and Amazon SNS updates the list of your destination phone numbers.

7. Repeat these steps in each Region where you added the destination phone number and no longer plan to use it.

## Moving out of the SMS sandbox

Moving your AWS account out of the [SMS sandbox \(p. 167\)](#) requires that you first add, verify, and test destination phone numbers. Then, you must create a case with AWS Support.

### To request that your AWS account is moved out of the SMS sandbox

1. While your AWS account is in the SMS sandbox, [add and verify \(p. 167\)](#) one or more destination phone numbers.
2. Confirm that you can publish and receive messages to at least one verified destination phone number.

For instructions on publishing SMS messages to one or more phone numbers, see [Publishing to a mobile phone \(p. 196\)](#).

3. On the Amazon SNS console's **Mobile text messaging (SMS)** page, under **Account information**, choose **Exit SMS sandbox**.

The [AWS Support Center](#) opens and creates a case. The **Service limit increase** option is already selected and the **Limit type** is set to **SNS Text Messaging**.

4. Under **Case details**, provide the following information:
  - A link to the site or the name of the app that you plan to send SMS messages from
  - The type of messages that you plan to send: one-time passwords, promotional, or transactional
  - The AWS Region that you plan to send SMS messages from

- The countries or regions that you plan to send SMS messages to
  - A description of how your customers opt in to receive messages from you
  - Any message template that you plan to use
5. Under **Requests**, do the following:
- a. For **Region**, choose the AWS Region where you want to move your AWS account out of the SMS sandbox.
  - b. For **Resource Type**, choose **General Limits**.
  - c. For **Limit**, choose **Exit SMS Sandbox**.
  - d. (Optional) To make additional requests, such as a limit increase, choose **Add another request**. Then, choose the options for your request.
6. (Optional) Under **Case description**, enter any additional relevant details about your request.
7. Under **Contact options**, choose your **Preferred contact language**.
8. Choose **Submit**.

The AWS Support team provides an initial response to your request within 24 hours.

To prevent our systems from being used to send unsolicited or malicious content, we consider each request carefully. If we can, we will grant your request within this 24-hour period. However, if we need additional information from you, it might take longer to resolve your request.

If your use case doesn't align with our policies, we might be unable to grant your request.

## Origination identities for SMS messages

When you send SMS messages using Amazon SNS, you can identify yourself to your recipients using the following types of *originating identities*:

- [Sender IDs \(p. 169\)](#)
- [Origination numbers \(p. 170\)](#)

### Note

Amazon SNS SMS messaging is available in Regions where Amazon Pinpoint is not currently supported. If you operate in Europe (Stockholm), Middle East (Bahrain), Europe (Paris), South America (São Paulo), or US West (N. California), open the Amazon Pinpoint console in the US East (N. Virginia) Region to register your 10DLC company and campaign, but *do not* request a 10DLC number. Instead, open a case in the [AWS Support Center](#) to request a 10DLC number in the AWS Region where you use or plan to use Amazon SNS to send SMS messages. To learn more about how to request origination identities, see [Requesting support for SMS messaging with Amazon SNS \(p. 181\)](#).

## Sender IDs

A sender ID is an alphabetic name that identifies the sender of an SMS message. When you send an SMS message using a sender ID, and the recipient is in an area where sender ID authentication is supported, your sender ID appears on the recipient's device instead of a phone number. A sender ID provides SMS recipients with more information about the sender than a phone number, long code, or short code provides.

### Important

AWS prohibits SMS [SMS spoofing](#), where the sender ID is used to impersonate another person, company, or product. Only use a sender ID that represents a brand or trademark that you own.

There's no additional charge for using sender IDs. However, support and requirements for sender ID authentication varies. Several major markets (including Canada, China, and the United States) don't support using sender IDs. Some areas require that companies who send SMS messages to individual customers must use a sender ID that's pre-registered with a regulatory agency or industry group.

## Origination numbers

An *origination number* is a numeric string that identifies an SMS message sender's phone number. When you send an SMS message using an origination number, the recipient's device shows the origination number as the sender's phone number. You can specify different origination numbers by use case.

**Tip**

To view a list of all existing origination numbers in your AWS account, in the navigation pane of the [Amazon SNS console](#), choose **Origination numbers**.

Support for origination numbers is not available in countries where local laws require the use of [sender IDs \(p. 169\)](#) instead of origination numbers.

### Topics

- [10DLC \(p. 170\)](#)
- [Toll-free numbers \(p. 178\)](#)
- [Short codes \(p. 179\)](#)
- [Person-to-person \(P2P\) long codes \(p. 180\)](#)
- [U.S. product number comparison \(p. 180\)](#)

## 10DLC

US carriers no longer support using application-to-person (A2P) SMS messaging over local, unregistered long codes. For high-volume A2P SMS messaging, US carriers instead offer a new type of long code called 10-digit long codes (10DLC).

### What is 10DLC?

10DLC is a type of long code that is registered with carriers to support high volume A2P SMS messaging using the 10-digit phone number format. Amazon SNS no longer offers local long codes as an SMS product and instead offers 10DLC. 10DLC doesn't impact you if you use only short codes and toll-free numbers.

10DLC is a 10-digit phone number used only in the United States. Messages sent from a 10DLC to recipients show a 10-digit number as the sender. Unlike toll-free numbers, 10DLC supports both transactional *and* promotional messaging, and can include any US area code.

If you have existing local long codes, you can request that their local long codes be enabled for 10DLC. To do so, complete the 10DLC registration process and then submit a support ticket. In the event that there's a problem with enabling your long code for 10DLC, you're notified and instructed to request a new 10DLC through the Amazon Pinpoint (not Amazon SNS) console. For information about how to file a support ticket to convert a long code, see [Associating a long code with a 10DLC campaign \(p. 176\)](#).

In order to use a 10DLC number, first register your company and create a 10DLC campaign using the Amazon Pinpoint (not Amazon SNS) console. AWS shares this information with The Campaign Registry, a third party that approves or rejects your registration based on the information. In some cases, registration occurs immediately. For example, if you've previously registered with The Campaign Registry, they might already have your information. However, some campaigns might take one week or longer for approval. After your company and 10DLC campaign are approved, you can purchase a 10DLC number and associate it with your campaign. Requesting a 10DLC might also take up to a week for approval.

Although you can associate multiple 10DLCs with a single campaign, you can't use the same 10DLC across multiple campaigns. For each campaign you create, you need to have a unique 10DLC.

**Important**

If you use local long codes or a shared pool of numbers, you must switch over to 10DLC. There's no impact on you if you use only short codes and toll-free numbers. For more information about how The Campaign Registry uses your information, see their [FAQ](#) at [campaignregistry.com](http://campaignregistry.com).

### 10DLC capabilities

The capabilities of 10DLC phone numbers depend on the mobile carriers of your recipients. AT&T provides a throughput limit based on the number of message parts that can be sent each minute. T-Mobile and Sprint provide a daily limit, with no limitation on the number of message parts that can be sent per minute. As of February 15, 2021, Verizon hasn't announced its 10DLC throughput policy.

When you set up 10DLC, you have to register your company. Your company information is used by The Campaign Registry to calculate a trust score. Your trust score determines the capabilities of your 10DLC phone number. The following table shows the 10DLC trust score tiers and the capabilities of 10DLC numbers that fall into those tiers. We're working with The Campaign Registry to provide additional information about finding the trust score for your company.

Tier	Message parts per minute (AT&T)	Maximum daily messages (T-Mobile)
High	3,600	200,000
Medium-high	600	40,000
Medium-low	30	10,000
Basic	12	2,000

For a comparison of the different phone number types see [U.S. product number comparison \(p. 180\)](#).

### Getting started with 10DLC

Use the Amazon Pinpoint (not Amazon SNS) console to request your 10DLC. Follow these steps to set up 10DLC for use with your 10DLC campaigns.

#### 1. Register your company.

Before you can request a 10DLC, your company must be registered with The Campaign Registry; for information, see [Registering a company \(p. 172\)](#). Registration is typically instantaneous unless The Campaign Registry requires more information. There is a one-time registration fee to register your company, displayed on the registration page. This one-time fee is paid separately from your monthly charges for the campaign and 10DLC.

**Note**

Amazon SNS SMS messaging is available in Regions where Amazon Pinpoint is not currently supported. In these cases, open the Amazon Pinpoint console in the US East (N. Virginia) Region to register your 10DLC company and campaign, but *do not* request a 10DLC number. Instead, open a case in the [AWS Support Center](#) to request a 10DLC number in the AWS Region where you use or plan to use Amazon SNS to send SMS messages. For information on Regions where Amazon Pinpoint is available, see [Amazon Pinpoint endpoints and quotas](#) in the [AWS General Reference](#).

#### 2. Register your campaign.

After your company is registered, create a 10DLC campaign and associate it with one of your registered companies. This campaign is submitted to The Campaign Registry for approval. In most

cases, 10DLC campaign approval is instantaneous unless The Campaign Registry requires more information. For more information, see [Registering a 10DLC campaign \(p. 174\)](#).

### 3. Request your 10DLC number.

After your 10DLC campaign is approved, you can request a 10DLC and associate that number with the approved campaign. Your 10DLC campaign can only use a number approved for it. See [Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS \(p. 184\)](#).

## 10DLC registration and monthly fees

There are registration and monthly fees associated with using 10DLC, such as registering your company and 10DLC campaign. These are separate from any other monthly fees charged by AWS. For more information, see the [Amazon SNS Worldwide SMS Pricing](#) page.

## Registering a company

Before you can request a 10DLC, you need to register your company with The Campaign Registry.

### Note

Amazon SNS SMS messaging is available in Regions where Amazon Pinpoint is not currently supported. In these cases, open the Amazon Pinpoint console in the US East (N. Virginia) Region to register your 10DLC company and campaign, but *do not* request a 10DLC number. Instead, open a case in the [AWS Support Center](#) to request a 10DLC number in the AWS Region where you use or plan to use Amazon SNS to send SMS messages. For information on Regions where Amazon Pinpoint is available, see [Amazon Pinpoint endpoints and quotas](#) in the [AWS General Reference](#).

## Register the company

You need to register your company only once. After it's registered, you won't be able to make any direct changes to company information other than through filing a support request, described later in this section.

### Important

You can only register a company in a single AWS account. If you have multiple accounts, such as Test, Development, and Production accounts, you should only register in the Production account. You can then use cross-account access to send messages through the 10DLC phone number in your other accounts. Do not enter fictitious or placeholder information in the registration process. For more information on setting up cross-account access, see [10DLC cross-account access \(p. 177\)](#).

## To register a company

The first step before using a 10DLC is to register your company and provide contact information. You do this using the Amazon Pinpoint (not Amazon SNS) console.

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Account Settings**, and then under **SMS and voice**, choose **10DLC campaigns**.
3. Choose **Register company**.
4. On the **Register your company** page, you'll see **Registration fee**. This is a one-time fee associated with registering your company. This cost is separate from any other monthly costs or fees, and is charged to you only when your company has been registered.
5. In the **Company info** section, provide the following information:
  - **Legal company name** – This is the name under which the company is registered. You must use the company's legal name. Do not modify or change this name.

- **What type of legal form is this organization** – Choose one of the following from the dropdown list: **Private profit**, **Public for profit**, or **Not for profit**.
  - If you choose **Public for profit**, you're prompted to supply the company's stock symbol and the stock exchange on which it's listed.
  - Choose the country where the company is registered from the **Country of registration** list.
  - If the company is known by any name other than its registered name, enter that name in the **Doing Business As (DBA) or brand name** field.
  - Enter the business's **Tax ID**. In the United States, this is a 9-digit number. A tax ID is required, so you must provide an actual tax ID. Do not leave this blank or complete with false information.
  - For **Vertical**, choose the vertical market that most closely resembles the company you're registering.
6. In the **Contact info** section, provide contact information for contacting the company.
    - **Address/Street** – The physical street address.
    - **City** – The city in which the physical address is located.
    - **State or region** – The state or region where the address is located.
    - **Zip Code /Postal Code** – The associated ZIP or postal code for the address.
    - **Company website** – Provide the full URL, including HTTP or HTTPS.
    - **Support email** – The full email address where your company support can be contacted.
    - **Support phone number** – The phone number, using an E.164 format, where your company support can be contacted. For example, **12065550101**. You don't need to supply the leading +.
  7. When you're finished, choose **Create**. This submits your company registration to The Campaign Registry. Typically, approval is instantaneous.
  8. After your company is registered, you can create a 10DLC campaign. For information about creating a 10DLC campaign, see [Registering a 10DLC campaign \(p. 174\)](#).

### Editing or deleting a company

After a company is registered, you can't change or delete it except through filing a support request.

#### To edit or delete a company

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Account Settings**, and then under **SMS and voice**, choose **10DLC campaigns**.
3. To open the **Company info page**, choose the link of the company you want to modify.

#### Note

On the **Company info** page you can also create a new 10DLC campaign.

4. To open Support Center, choose **Request to delete/edit**.
5. On the Support page, choose **Create case**.
6. For the case type, choose **Service limit increase**.
7. For the **Limit type**, choose **Pinpoint**.
8. In the Requests section, choose the **Region**, and then for the **Limit**, choose **10DLC**.
9. For **Use case description**, enter the company name you want to delete or edit, and then provide details for what you want to be done.
10. Under **Contact options**, for **Preferred contact language**, choose the language that you prefer to use when communicating with the AWS Support team.
11. For **Contact method**, choose your preferred method of communicating with the AWS Support team.
12. Choose **Submit**.

## Registering a 10DLC campaign

A 10DLC campaign indicates how you'll be using the 10DLC number you're requesting. You do this using the Amazon Pinpoint (not Amazon SNS) console. Before you can create and register a 10DLC campaign, your company must be registered. For information on registering a company, see [Registering a company \(p. 172\)](#).

On this page, you first provide the details about the company you're creating the 10DLC campaign for and then provide the use case details of the campaign itself. The information on this page is then provided to The Campaign Registry for approval.

### Step 1: Choose the company associated with the 10DLC campaign

In this section, you'll choose the company you're creating the 10DLC campaign for and provide additional details.

#### Important

Carefully verify that all of the information that you enter is correct. Once you submit a campaign registration, you can only edit it by opening a ticket with AWS Support. Errors in the registration process could result in reduced throughput for your 10DLC phone numbers.

### To create a 10DLC campaign

#### Note

For each campaign you're registering, there is a carrier registration fee that won't be charged until your campaign is approved.

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Account Settings**, and then under **SMS and voice**, choose **10DLC campaigns**.
3. On the **Create a 10DLC campaign** page, the **Fees** section displays the one-time carrier 10DLC campaign registration fee, separate from any monthly fees charged by AWS, in addition to the monthly 10DLC campaign fee. This fee charged is per campaign.
4. For **Company name**, choose the company that you're creating this campaign for. If you haven't already registered the company, you must do that first.
5. Enter a **10DLC campaign name**.
6. For **Vertical**, choose the vertical market that most closely resembles your market.
7. In **Help message**, enter the message that your customers receive if they respond with a HELP keyword.
8. In **Stop message**, enter the message a customer receives from you if they respond with the STOP keyword.

### Step 2: Add the use case details

After you've provided the company information, you'll next supply the 10DLC campaign details.

### To provide use case details

1. Choose an option for the use case.
  - **Standard** – The purpose tells the mobile operators how you'll be using the 10DLC phone number. You'll be prompted to supply additional details about the proposed campaign.

#### Note

In some cases, for instance if you've been previously registered with the Campaign Registry, you might be automatically approved.

- **Special** – A special campaign might be an emergency or a time-sensitive announcement. For example, this might be a public safety warning or a sweepstakes announcement. Some of these use cases require additional documentation. If you choose one of those use cases, you're prompted to create a support case to justify the request.
2. For **Use case**, choose a use case that most closely resembles your campaign from the preset list of use cases. The monthly fee for each use case appears next to the use case name.
  3. Enter at least one **Sample message**. This is the sample message you plan to send to your customers. This message must be the exact message you'll be sending.
  4. If your use case supports multiple scenarios, enter up to four additional sample messages.
  5. The **Campaign and content attributes** section defines the characteristics of your 10DLC campaign. These are a series of **Yes** or **No** options indicating particular features of the campaign. Some attributes are mandatory, so you can't change the default value. These values appear grayed out.

**Important**

Make sure that the attributes you choose are applicable to your campaign. In some cases, choosing an attribute that is not applicable to your campaign can negatively affect your throughput per second (TPS).

- **Subscriber opt-in** – Subscribers can opt in to receive messages about this campaign.
  - **Subscriber opt-out** – Subscribers can opt out of receiving messages about this campaign.
  - **Subscriber help** – Subscribers can contact the message sender after sending the HELP keyword.
  - **Number pooling** – This 10DLC campaign uses more than 50 phone numbers.
  - **Direct lending or loan arrangement** – The campaign includes information about direct lending or other loan arrangements.
  - **Embedded link** – The 10DLC campaign includes an embedded link. URL shorteners, such as Tinyurl or Bitly, are not allowed.
  - **Embedded phone number** – The campaign includes an embedded phone number that is not a customer support number.
  - **Affiliate marketing** – The 10DLC campaign includes information from affiliate marketing.
  - **Age-gated content** – The 10DLC campaign includes age-gated content as defined by carrier and Cellular Telecommunications and Internet Association (CTIA) guidelines.
6. Choose **Create**. The SMS and voice page opens. A message appears indicating that your campaign was submitted and is under review. You can see the status of your request on the **10DLC campaigns** tab. You can check the status of your registration on the **10DLC** tab, which will be one of the following:
    - **Active** – Your 10DLC campaign was approved. You can request a 10DLC and assign that number to your campaign. See [Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS \(p. 184\)](#).
    - **Pending** – Approval for your 10DLC campaign has not yet been received by one or more carriers. In some cases, approval might take one week or more. If the status changes to any other status, the Amazon Pinpoint console reflects that change. We don't notify you of status changes.
    - **Rejected** – Your 10DLC campaign was rejected by one or more carriers. To get more information, submit a support request that includes the campaign ID of the rejected campaign. Amazon Pinpoint doesn't include rejection reasons on the console, and we don't notify you if your campaign is rejected.
    - **Suspended** – One or more carriers suspended your 10DLC campaign. To get more information, submit a support request that includes the campaign ID of the suspended campaign. Amazon Pinpoint doesn't include suspension reasons on the console, and we don't notify you if your campaign is suspended.
  7. If your 10DLC is approved, you can request a 10DLC number to associate with that campaign. For information about requesting a 10DLC number, see [Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS \(p. 184\)](#).

## Editing or deleting a 10DLC campaign

After a campaign is registered with the carriers, you can't change it or delete except through filing a support request. When filing the request, you must provide the **10DLC Campaign ID**.

### Important

If you have any 10DLC numbers associated with a campaign you want to delete, first remove those numbers from the campaign. You can submit this request at the same time you're deleting a campaign.

### To edit or delete a 10DLC campaign

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Account Settings**, and then under **SMS and voice**, choose **10DLC campaigns**.
3. Choose the 10DLC campaign you want to modify.
4. To open Support Center, choose **Request to delete/edit**.
5. On the support page, choose **Create case**.
6. For the case type, choose **Service limit increase**.
7. For the **Limit type**, choose **Pinpoint**.
8. In the **Requests** section, choose the **Region**, and then for the **Limit**, choose **10DLC**.
9. For **Use case description**, enter the campaign ID of the campaign you want to edit or delete. We recommend that you edit or delete only one campaign per each support request.
10. Under **Contact options**, for **Preferred contact language**, choose the language that you prefer to use when communicating with the AWS Support team.
11. For **Contact method**, choose your preferred method of communicating with the AWS Support team.
12. Choose **Submit**.

## Associating a long code with a 10DLC campaign

If you have an existing long code, you can associate that long code with one of your current 10DLC campaigns by filing a support request. You do this using the Amazon Pinpoint (not Amazon SNS) console.

The long code you associate with the 10DLC campaign can only be used with that campaign and can't be used for any other 10DLC campaign. While your long code is being migrated to 10DLC you'll still be able to use it. Until it's approved, however, you won't be able to use it for any 10DLC campaign.

When filing the request, you'll need:

- The long codes to associate with a 10DLC campaign
- The 10DLC campaign ID to associate with the long code

### Note

Before you can associate any long codes with a campaign, you need to have that 10DLC campaign registered. If you have not yet created and registered a 10DLC campaign, see [Registering a 10DLC campaign \(p. 174\)](#).

### To assign a long code to 10DLC

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Settings**, and then under **SMS and voice**, choose the **Phone numbers** tab.
3. Choose the long code that you want to convert to a 10DLC.
4. To open Support Center, choose **Assign to 10DLC campaign**.

5. For the case type, choose **Service limit increase**.
6. For **Limit type**, choose **Pinpoint**.
7. In the **Requests** section, choose the **Region**, and then for the **Limit**, choose **10DLC**.
8. Under **Case description**, for **Use case description**, be sure to include the 10DLC campaign ID and the long code numbers you want to associate that campaign. You can include multiple long codes in the request, but you should include only one campaign ID.
9. Under **Contact options**, for **Preferred contact language**, choose the language that you prefer to use when communicating with the AWS Support team.
10. For **Contact method**, choose your preferred method of communicating with the AWS Support team.
11. Choose **Submit**.

### 10DLC cross-account access

10DLC companies and campaigns all reside within a single AWS account. If you have multiple accounts, you can associate those other accounts with your main account in order to use your 10DLC numbers from any of those accounts. You do this using the Amazon Pinpoint (not Amazon SNS) console. Then you create an AWS Identity and Access Management (IAM) role, and delegate access permissions to your other accounts.

### To allow your multiple AWS accounts access to your 10DLC numbers

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Under **Account Settings**, and then under **SMS and voice**, choose the **10DLC** tab.
3. Register your company and campaign on the main account that will use the 10DLC phone number. For example, your main account might be a **Production** account, but you want your **Development** account to use a 10DLC number.
4. Create the IAM role in your main account, which allows another account to call the `SendMessage` operation of the Pinpoint API. See [Creating IAM roles](#) in the *IAM User Guide* for more information on creating roles.
5. Delegate and test access permission from your main account using IAM roles with any of other your accounts that need to use your 10DLC numbers. For example, you might delegate access permission from your **Production** account to your **Development** account. See [Delegate access across AWS accounts using IAM roles](#) in the *IAM User Guide* for more information about delegating and testing.
6. Using the new role, send a message using a 10DLC number from the main account. When sending the message, Amazon SNS assumes the main account role and sends the message. See [Using IAM roles](#) in the *IAM User Guide* for more information on using a role.

#### Important

A sent message from the secondary account is treated as if it were sent from your main account. Quotas and billing are counted against this account and not any secondary accounts.

### Rejected 10DLC companies and campaigns

If your company or 10DLC campaign are rejected, you'll need to submit a support request to get more information. You do this using the Amazon Pinpoint (not Amazon SNS) console.

Amazon Pinpoint does not display rejection reasons on the console.

### To submit a support request for a rejected company or 10DLC campaign

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
2. Choose **Support**, and then **Support Center**.

3. On the Support page, choose **Create case**.
4. For the case type, choose **Service limit increase**.
5. For the **Limit type**, choose **Pinpoint**.
6. In the **Requests** section,
  - Choose the **Region**.
  - For the **Resource Type**, choose **10DLC Registration**.
  - For the **Limit**, choose **Company or 10DLC Campaign Registration Rejection**.
7. For **Use case description**, enter the rejected 10DLC company name or 10DLC campaign ID for which you want to know the rejection reason.
8. Under **Contact options**, for **Preferred contact language**, choose the language that you prefer to use when communicating with the AWS Support team.
9. For **Contact method**, choose your preferred method of communicating with the AWS Support team.
10. Choose **Submit**.

## Toll-free numbers

A toll-free number is a 10-digit number that begins with one of the following area codes: 800, 888, 877, 866, 855, 844, or 833. You can use toll-free numbers to send transactional messages only.

To purchase toll-free numbers, use the Amazon Pinpoint console. No message template registration is required. For more information, see [Requesting a number](#) in the *Amazon Pinpoint User Guide*.

Currently, Amazon Pinpoint supports toll-free numbers for both voice and SMS messages, but Amazon SNS supports SMS messaging only.

### Guidelines for using toll-free numbers

When using a toll-free number as an origination number, follow these guidelines:

- Don't use shortened URLs created from third-party URL shorteners, as these messages are more likely to be filtered as spam.

If you need to use a shortened URL, consider using a [10DLC number \(p. 170\)](#) or [short code \(p. 179\)](#).

Using short codes and 10DLCs require that you register your message template, where you can specify a shortened URL.

- Be aware that keyword opt-out (STOP) and opt-in (UNSTOP) responses are set at the carrier level. You can't modify these keywords or other any other keywords, or the messages that are sent when users reply with STOP and UNSTOP.
- Don't send the same or similar message contents using multiple toll-free numbers. Carriers call this practice *snowshoeing* or *number pooling* and target these messages for filtering.
- Because toll-free numbers are subject to content filtering, be sure your messages don't contain any of the following types of restricted content:

Category	Examples
High-risk financial services	<ul style="list-style-type: none"> <li>• Payday loans</li> <li>• Short-term high-interest loans</li> <li>• Auto loans</li> <li>• Mortgage loans</li> <li>• Student loans</li> <li>• Debt collection</li> </ul>

Category	Examples
Debt forgiveness	<ul style="list-style-type: none"> <li>• Debt consolidation</li> <li>• Debt reduction</li> <li>• Credit repair programs</li> </ul>
Get-rich-quick schemes	<ul style="list-style-type: none"> <li>• Work-from-home programs</li> <li>• Risk-investment opportunities</li> <li>• Pyramid or multi-level marketing schemes</li> </ul>
Prohibited/controlled substances	<ul style="list-style-type: none"> <li>• Cannabis</li> </ul>
Phishing	<ul style="list-style-type: none"> <li>• Attempts to get users to reveal personal information or website login information.</li> </ul>
S.H.A.F.T.	<ul style="list-style-type: none"> <li>• Sex</li> <li>• Hate</li> <li>• Alcohol</li> <li>• Firearms</li> <li>• Tobacco</li> </ul>

## Short codes

Short codes are numeric sequences that are shorter than a regular phone number. For example, in the United States and Canada, standard phone numbers (long codes) contain 11 digits, while short codes contain five or six digits. Amazon SNS supports dedicated short codes.

### Dedicated short codes

If you send a large volume of SMS messages to recipients in the United States or Canada, you can purchase a dedicated short code. Unlike the short codes in the shared pool, dedicated short codes are reserved for your exclusive use.

Using a memorable short code can help build trust. If you need to send sensitive information, such as one-time passwords, it's a good idea to send it using a short code so that your customer can quickly determine whether a message is actually from you.

If you're running a new customer acquisition campaign, you can invite potential customers to send a keyword to your short code (for example, "Text 'FOOTBALL' to 10987 for football news and information"). Short codes are easier to remember than long codes, and it's easier for customers to enter short codes into their devices. By reducing the amount of difficulty that customers encounter when they sign up for your marketing programs, you can increase the effectiveness of your campaigns.

Because mobile carriers must approve new short codes before making them active, they are less likely to flag messages sent from short codes as unsolicited.

When you use dedicated short codes to send SMS messages, you can send a higher volume of messages per 24-hour period than you can when you use other types of origination identities. In other words, you have a much higher *sending quota*. You can also send a much higher volume of messages per second. That is, you have a much higher *sending rate*.

### Important

There are additional costs to acquire short codes, and they can take a long time to implement. For example, in the United States, there's a one-time setup fee of \$650.00 (USD) for each short code, plus an additional recurring charge of \$995.00 per month for each short code. It can take 8–12 weeks for short

codes to become active on all carrier networks. To find the price and provisioning time for a different country or region, complete the procedure described in [Requesting dedicated short codes for SMS messaging with Amazon SNS \(p. 182\)](#).

## Person-to-person (P2P) long codes

### Important

Effective June 1, 2021, US telecom providers no longer support using person-to-person (P2P) long codes for application-to-person (A2P) communications to US destinations. Instead, you need to use another type of origination ID for these messages. For more information, see [Special requirements for sending SMS messages to US destinations \(p. 231\)](#).

P2P long codes are phone numbers that use the number format of the country or region where your recipients are located. P2P long codes are also referred to as long numbers or virtual mobile numbers. For example, in the United States and Canada, P2P long codes contain 11 digits: the number 1 (the country code), a three-digit area code, and a seven-digit phone number.

For more information about requesting P2P long codes, see [Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS \(p. 184\)](#).

Dedicated P2P long codes are reserved for use by your Amazon SNS account only—they aren't shared with other users. When you use dedicated P2P long codes, you can specify which P2P long code you want to use when you send each message. If you send multiple messages to the same customer, you can ensure that each message appears to be sent from the same phone number. For this reason, dedicated P2P long codes can be helpful in establishing your brand or identity.

P2P long codes aren't supported for A2P communications to US destinations.

If you send several hundred messages per day from a dedicated P2P long code, mobile carriers might identify your number as one that sends unsolicited messages. If your P2P long code is flagged, your messages might not be delivered to your recipients.

P2P long codes also have limited throughput. The maximum sending rates varies by country. Contact AWS Support for more information. If you plan to send large volumes of SMS messages, or you plan to send at a rate greater than one message per second, you should purchase a dedicated short code.

Some carriers don't allow you to use P2P long codes to send A2P SMS messages, including in the US. An A2P SMS is a message that's sent to a customer's mobile device when that customer submits his or her mobile number to an application. A2P messages are one-way conversations, such as marketing messages, one-time passwords, and appointment reminders. If you plan to send A2P messages, you should purchase a dedicated short code (if your customers are in the United States or Canada), or use a sender ID (if your recipients are in a country or region where sender IDs are supported).

## U.S. product number comparison

This table shows the support comparison for U.S. phone number types.

Product feature	Short code	Toll-free number	10DLC	
Number format	5-6 digits	10-digit number	10-digit number	
Channel support	SMS	SMS	SMS	
SMS traffic type	Promotional and transactional	Transactional	Promotional and transactional	
Requires vetting	Yes	No	Yes	
Estimated provisioning time	12 weeks <sup>1</sup>	Immediately	1 week	

Product feature	Short code	Toll-free number	10DLC	
SMS throughput (number of SMS messages per second) <sup>2</sup>	100 message parts per second; higher throughput available for an additional fee.	3 message parts per second	Varies based on your 10DLC registration. Supports up to 100 message parts per second.	
Keywords required	Opt-in, opt-out, and HELP	STOP, UNSTOP. These are network-managed. You cannot change the opt-out and opt back in messages.	Opt-in, opt-out, and HELP	

<sup>1</sup> Provisioning estimate doesn't include approval time.

<sup>2</sup> For more information on the maximum size for SMS messages, see [Publishing to a mobile phone \(p. 196\)](#).

## Requesting support for SMS messaging with Amazon SNS

Certain SMS options with Amazon SNS aren't available for your AWS account until you contact AWS Support. Create a case in the [AWS Support Center](#) to request any of the following:

- An increase to your monthly SMS spending threshold

By default, the monthly spending threshold is \$1.00 (USD). Your spending threshold determines the volume of messages that you can send with Amazon SNS. You can request a spending threshold that meets the expected monthly message volume for your SMS use case.

- A move from the [SMS sandbox \(p. 167\)](#) so that you can send SMS messages without restrictions. For more information, see [Moving out of the SMS sandbox \(p. 168\)](#).
- A dedicated [origination number \(p. 170\)](#)
- A dedicated sender ID

A *sender ID* is a custom ID that is shown as the sender on the recipient's device. For example, you can use your business brand to make the message source easier to recognize. Support for sender IDs varies by country or region. For more information, see [Supported Regions and countries \(p. 219\)](#).

When you create your case in the AWS Support Center, be sure to include all the required information for the type of request that you're submitting. Otherwise, AWS Support must contact you to obtain this information before proceeding. By submitting a detailed case, you help ensure that your case is fulfilled without delays. For the required details for specific types of SMS requests, see the following topics.

### Topics

- [Requesting dedicated short codes for SMS messaging with Amazon SNS \(p. 182\)](#)
- [Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS \(p. 184\)](#)
- [Requesting sender IDs for SMS messaging with Amazon SNS \(p. 185\)](#)
- [Requesting increases to your monthly SMS spending quota for Amazon SNS \(p. 187\)](#)

## Requesting dedicated short codes for SMS messaging with Amazon SNS

A short code is a number that you can use for high-volume SMS message sending. Short codes are often used for application-to-person (A2P) messaging, two-factor authentication (2FA), and marketing. A short code typically contains between three and seven digits, depending on the country or region that it's based in.

You can only use short codes to send messages to recipients in the same country where the short code is based. If your use case requires you to use short codes in more than one country, you have to request a separate short code for each country that your recipients are located in.

For information about short code pricing, see [Amazon SNS pricing](#).

### Important

If you're new to SMS messaging with Amazon SNS, request a monthly SMS spending threshold that meets the expected demands of your SMS use case. By default, your monthly spending threshold is \$1.00 (USD). You can request to increase your spending threshold in the same support case that includes your request for a short code. Or, you can use a separate case. For more information, see [Requesting increases to your monthly SMS spending quota for Amazon SNS \(p. 187\)](#).

In addition, if you're requesting a dedicated short code to send messages that will or may contain Protected Health Information (PHI), you should identify this purpose in your **Case description** when you open a support case, as detailed below.

### Step 1: Open a support case

Open a case with AWS Support by completing the following steps.

#### To request a dedicated short code

1. Go to the [AWS Support Center](#).
2. Choose **Create case**.
3. For **Regarding**, choose **Service Limit Increase**.
4. For **Limit Type**, choose **SNS Text Messaging**.
5. For **Resource Type**, choose **Dedicated SMS Short Codes**.
6. For **Limit**, choose the option that most closely resembles your use case.
7. For **New limit value**, specify how many short codes you want to reserve (typically, this value is 1).
8. For **Use Case Description**, summarize your use case, and summarize how your recipients will sign up for messages sent with your short code and provide the following information:

#### Company information:

- Company name.
- Company mailing address.
- Name and phone number for the primary contact for your request.
- Email address and toll-free number for support at your company.
- Company tax ID.
- Name of your product or service.

#### User sign-up process:

- Company website, or the website that your customers will sign up on to receive messages from your short code.

- How users will sign up to receive messages from your short code. Specify one or more of the following options:
  - **Text messages.**
  - **Website.**
  - **Mobile app.**
  - **Other.** If other, explain.
- The text for the option to sign up for messages on your website, app, or elsewhere.
- The sequence of messages that you plan to use for double opt-in. Provide all of the following:
  1. The SMS message that you plan to send when a user signs up. This message asks for the user's consent for recurring messages. For example: *ExampleCorp: Reply YES to receive account transaction alerts. Msg&data rates may apply.*
  2. The opt-in response that you expect from the user. This is typically a keyword, such as *YES*.
  3. The confirmation message that you want to send when customers send this keyword to your short code. For example: *You are now registered for account alerts from ExampleCorp. Msg&data rates may apply. Txt STOP to cancel or HELP for info.*

#### **The purpose of your messages:**

- The purpose of the messages that you plan to send with your short code. Specify one of the following options:
  - **Promotions and marketing.**
  - **Location-based services.**
  - **Notifications.**
  - **Information on demand.**
  - **Group chat.**
  - **Two-factor authentication (2FA).**
  - **Polling and surveys.**
  - **Sweepstakes or contests.**
  - **Other.** If other, explain.
- Whether you plan to use your short code to send promotional or marketing messages for a business other than your own.
- Whether you plan to use your short code to send messages that will or may contain Protected Health Information (PHI), as defined by the Health Insurance Portability and Accountability Act (HIPAA) and associated legislation and regulations.

#### **Message content:**

- The message that you plan to send when customers opt in to your messages by sending you a specific keyword. Be careful when you specify this keyword and message—it may take several weeks to change this message. When we create your short code, we register the keyword and message with the mobile phone carriers in the country where you use the short code. Your message might resemble the following example: *Welcome to ProductName alerts! Msg&data rates apply. 2 msgs per month. Reply HELP for help, STOP to cancel.*
- The response that you want to send when customers reply to your messages with the keyword *HELP*. This message has to include customer support contact information. For example: *ProductName Alerts: Help at example.com/help or (800) 555-0199. Msg&data rates apply. 2 msgs per month. Reply STOP to cancel.*
- The response that you want to send when customers reply to your messages with the keyword *STOP*. This message has to confirm that the user will no longer receive messages from you. For

example: You are unsubscribed from *ProductName* Alerts. No more messages will be sent. Reply HELP for help or (800) 555-0199.

- The text that you plan to send as a periodic reminder that the user is subscribed to your messages. For example: *Reminder: You're subscribed to account alerts from ExampleCorp. Msg&data rates may apply. Txt STOP to cancel or HELP for info.*
- An example of each type of message that you plan to send using your short code. Provide at least three examples. If you plan to send more than three types of messages, provide examples for all of them.

**Important**

Mobile carriers require us to provide all of the information listed above in order to provision short codes. We can't process your request until you provide all of this information.

9. Under **Contact options**, for **Preferred contact language**, choose whether you want to receive communications for this case in **English** or **Japanese**.
10. When you finish, choose **Submit**.

After we receive your request, we provide an initial response within 24 hours. We might contact you to request additional information. If we're able to provide you with a short code, we send you information about the costs associated with obtaining a short code in the country or region that you specified in your request. We also provide an estimate of the amount of time that's required to provision a short code in your country or region. It usually takes several weeks to provision a short code, although this delay can be much shorter or much longer depending on the country or region where the short code is based.

**Note**

The fees associated with using short codes begin immediately after we initiate your short code request with carriers. You're responsible for paying these charges, even if the short code hasn't been completely provisioned yet.

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. We might not be able to grant your request if your use case doesn't align with our policies.

## Next steps

You've registered a short code with wireless carriers and reviewed your settings in the Amazon SNS console. Now you can use Amazon SNS to send SMS messages with your short code as the origination number.

## Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS

**Important**

Effective June 1, 2021, US telecom providers no longer support using person-to-person (P2P) long codes for application-to-person (A2P) communications to US destinations. Instead, you need to use another type of origination ID for these messages. For more information, see [Special requirements for sending SMS messages to US destinations \(p. 231\)](#).

To request [10DLC numbers \(p. 170\)](#), [toll-free numbers \(p. 178\)](#), and [P2P long codes \(p. 180\)](#), use the Amazon Pinpoint console. For detailed instructions, see [Requesting a number](#) in the [Amazon Pinpoint User Guide](#).

Amazon SNS SMS messaging is available in Regions where Amazon Pinpoint is not currently supported. In these cases, open the Amazon Pinpoint console in the US East (N. Virginia) Region to register your 10DLC company and campaign, but do not request a 10DLC number. Instead, open a case in the [AWS Support Center](#) to request a 10DLC number in the AWS Region where you use or plan to use Amazon

SNS to send SMS messages. For information on Regions where Amazon Pinpoint is available, see [Amazon Pinpoint endpoints and quotas](#) in the *AWS General Reference*.

**Note**

If you're new to SMS messaging with Amazon SNS, you should also request a monthly SMS spending threshold that meets the expected demands of your SMS use case. By default, your monthly spending threshold is \$1.00 (USD). For more information, see [Requesting increases to your monthly SMS spending quota for Amazon SNS \(p. 187\)](#).

## Requesting sender IDs for SMS messaging with Amazon SNS

**Important**

If you're new to SMS messaging with Amazon SNS, request a monthly SMS spending threshold that meets the expected demands of your SMS use case. By default, your monthly spending threshold is \$1.00 (USD). You can request to increase your spending threshold in the same support case that includes your request for a sender ID. Or, if you prefer, you can open a separate case. For more information, see [Requesting increases to your monthly SMS spending quota for Amazon SNS \(p. 187\)](#).

In SMS messaging, a *sender ID* is a name that appears as the message sender on recipients' devices. Sender IDs are a useful way to identify yourself to the recipients of your messages.

Support for sender IDs varies by country. For example, carriers in the United States don't support sender IDs at all, but carriers in India require senders to use sender IDs. For a complete list of countries that support sender IDs, see [Supported Regions and countries \(p. 219\)](#).

**Important**

Some countries require you to register sender IDs before you use them to send messages. Depending on the country, this registration process might take several weeks. The countries that require pre-registered sender IDs are indicated in the table on the [Supported Countries \(p. 219\)](#) page.

If you have enterprise support and are registering multiple templates across multiple accounts, follow the steps below, and work with your Technical Account Manager to ensure that your onboarding experience is coordinated.

If you're sending messages to recipients in a country where sender IDs are supported, and that country doesn't require you to register your sender ID, you don't have to perform any additional steps. You can start sending messages that include sender ID values immediately.

You only need to complete the procedures on this page if you plan to send messages to a country where registration of sender IDs is required.

### Step 1: Open an Amazon SNS SMS case

If you plan to send messages to recipients in a country where sender IDs are required, you can request a sender ID by creating a new case in the AWS Support Center.

**Note**

If you plan to send messages to recipients in a country where sender IDs are allowed but not required, you don't need to open a case in the Support Center. You can start sending messages that use sender IDs immediately.

#### To request a sender ID

1. Sign in to the AWS Management Console at <https://console.aws.amazon.com/>.
2. On the **Support** menu, choose **Support Center**.
3. On the **My support cases** tab, choose **Create case**.
4. Choose **Service quota increase**.

5. Under **Case classification**, do the following:
  - a. For **Quota type**, choose **Pinpoint SMS**.
  - b. For **Provide a link to the site or app which will be sending SMS messages**, identify the website or application where your audience members opt in to receive your SMS messages.
  - c. For **What type of messages do you plan to send**, choose the type of message that you plan to send using your sender ID:
    - **One Time Password** – Messages that provide passwords that your customers use to authenticate with your website or application.
    - **Promotional** – Noncritical messages that promote your business or service, such as special offers or announcements.
    - **Transactional** – Important informational messages that support customer transactions, such as order confirmations or account alerts. Transactional messages must not contain promotional or marketing content.
  - d. For **Which countries do you plan to send messages to**, specify the countries where you want to register a sender ID. Support for sender IDs and sender ID registration requirements vary by country. For more information, see [Supported Regions and countries \(p. 219\)](#).
- If the list of countries exceeds the number of characters allowed by this text box, you can instead list the countries in the **Case description** section.
6. Under **Requests**, do the following:
  - a. For **Resource Type**, choose **General Quotas**.
  - b. For **Quota**, choose **SenderId Registration**.
  - c. For **New quota value**, enter the number of sender IDs that you're requesting. Typically, this value is 1.
7. Under **Case description**, for **Use case description**, provide the following information:
  - The sender ID that you want to register.
  - The template that you plan to use for your SMS messages.
  - The number of messages that you plan to send to each recipient per month.
  - Information about how your customers opt in to receiving messages from you.
  - The name of your company or organization.
  - The address that's associated with your company or organization.
  - The country where your company or organization is based.
  - A phone number for your company or organization.
  - The URL of the website for your company or organization.

After we receive your request, we provide an initial response within 24 hours. We might contact you to request additional information. If we're able to provide you with a Sender ID, we send you an estimate of the amount of time that's required to provision it.

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. We might not be able to grant your request if your use case doesn't align with our policies.

## Step 2: Update your SMS settings in the Amazon SNS console

When we complete the process of obtaining your sender ID, we respond to your case. When you receive this notification, complete the steps in this section to configure Amazon SNS to use your sender ID as the default sender ID for all messages sent using your account. Alternatively, you can choose to specify which sender ID to use when [publishing the message](#).

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose Mobile, Text messaging (SMS).
3. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
4. On the **Edit text messaging preferences page**, in the **Details** section, do the following:
5. For **Default sender ID**, enter the provided sender ID to be used as the default for all messages from your account.
6. Choose **Save changes**.

## Next steps

You've registered a sender ID and updated your settings in the Amazon SNS console. Now you can use Amazon SNS to send SMS messages with your sender ID. SMS recipients in supported countries will see your sender ID as the message sender on their devices. If a different sender ID is used when publishing messages, it will override the default ID configured here.

## Requesting increases to your monthly SMS spending quota for Amazon SNS

Your spending quota determines how much money you can spend sending SMS messages through Amazon SNS each month. When Amazon SNS determines that sending an SMS message would incur a cost that exceeds your spending quota for the current month, it stops publishing SMS messages within minutes.

### Important

Because Amazon SNS is a distributed system, it stops sending SMS messages within minutes of the spending quota being exceeded. During this period, if you continue to send SMS messages, you might incur costs that exceed your quota.

We set the spending quota for all new accounts at \$1.00 (USD) per month. This quota is intended to let you test the message-sending capabilities of Amazon SNS. This quota also helps to reduce the risk of sending large campaigns before you're actually ready to use Amazon SNS for your production workloads. Finally, this quota is necessary to prevent malicious users from abusing Amazon SNS.

To request an increase to the SMS spending quota for your account, open a quota increase case in the AWS Support Center.

## Step 1: Open an Amazon SNS SMS case

You can request an increase to your monthly spending quota by opening a quota increase case in the AWS Support Center.

### Note

Some of the fields on the request form are marked as "optional." However, AWS Support requires all of the information that's mentioned in the following steps in order to process your request. If you don't provide all of the required information, you may experience delays in processing your request.

### To request a spending quota increase

1. Go to the [AWS Support Center](#).
2. Choose **Create case**.
3. Choose **Service limit increase**, and then under **Case classification**, perform the following steps:
4. For **Limit type**, choose **SNS Text Messaging**.

5. For **Link to site or app which will be sending SMS - optional**, enter the URL of your website or application.
6. For **Type of messages - optional**, choose **One Time Password**, **Promotional**, or **Transactional**, depending on what you plan to send.
7. For **Targeted Countries** - optional, choose **General Limits**.
8. Under **Requests**, for Request 1, do the following:
9. For **Resource Type**, choose **General Limits**.
10. For **New limit**, enter the needed spend limit that you calculated earlier.
11. Under **Case description**, for Use case description, enter the description that you wrote earlier.
12. Expand **Contact options**, and then choose your preferred contact language.
13. Choose **Submit**.
14. When you finish, choose **Submit**.

The AWS Support team provides an initial response to your request within 24 hours.

To prevent our systems from being used to send unsolicited or malicious content, we consider each request carefully. If we can, we will grant your request within this 24-hour period. However, if we need additional information from you, it might take longer to resolve your request.

If your use case doesn't align with our policies, we might be unable to grant your request.

## Step 2: Update your SMS settings on the Amazon SNS console

After we notify you that your monthly spending quota has been increased, you have to adjust the spending quota for your account on the Amazon SNS console.

### Important

Important: If you skip this step, your SMS spend limit won't increase.

### To adjust your spending quota on the console

1. Open the Amazon SNS console.
2. Open the left navigation menu, expand Mobile, and then choose Text messaging (SMS).
3. On the Mobile text messaging (SMS) page, next to Text messaging preferences, choose Edit.
4. On the Edit text messaging preferences page, under Details, enter your new SMS spend limit for Account spend limit.

### Note

You might receive a warning that the entered value is larger than the default spend limit.  
You can ignore this.

5. Choose Save changes.

If you get an "Invalid Parameter" error, check the contact from AWS Support and confirm that you entered the correct new SMS spend limit. If you still experience a problem, open a case in the AWS Support Center.

6. Open the Amazon Pinpoint console at <https://console.aws.amazon.com/pinpoint/>.
7. On the All projects page, choose a project that uses the SMS channel.
8. In the navigation pane, under **Settings**, choose **SMS and voice**.
9. In the **SMS and voice** section, choose **Edit**.
10. Under **Account-level settings**, for **Account spending limit**, enter the maximum amount, in US Dollars, that you want to spend on SMS messages each calendar month. You can specify a value

that's less than or equal to the total monthly spending quota provided by AWS Support. By setting a lower value, you can control your monthly spending while still retaining the capacity to scale up if necessary.

11. Choose **Save changes**.

## Setting SMS messaging preferences

Use Amazon SNS to specify preferences for SMS messaging. For example, you can specify whether to optimize deliveries for cost or reliability, your monthly spending limit, how deliveries are logged, and whether to subscribe to daily SMS usage reports.

These preferences take effect for every SMS message that you send from your account, but you can override some of them when you send an individual message. For more information, see [Publishing to a mobile phone \(p. 196\)](#).

### Topics

- [Setting SMS messaging preferences using the AWS Management Console \(p. 189\)](#)
- [Setting preferences \(AWS SDKs\) \(p. 190\)](#)

## Setting SMS messaging preferences using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. Choose a [region that supports SMS messaging \(p. 219\)](#).
3. On the navigation panel, choose **Mobile, Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
5. On the **Edit text messaging preferences** page, in the **Details** section, do the following:
  - a. For **Default message type**, choose one of the following:
    - **Promotional** (default) – Non-critical messages (for example, marketing). Amazon SNS optimizes message delivery to incur the lowest cost.
    - **Transactional** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication. Amazon SNS optimizes message delivery to achieve the highest reliability.

For pricing information for promotional and transactional messages, see [Global SMS Pricing](#).

- b. (Optional) For **Account spend limit**, enter the amount (in USD) that you want to spend on SMS messages each calendar month.

### Important

- By default, the spend quota is set to 1.00 USD. If you want to raise the service quota, [submit a request](#).
- If the amount set in the console exceeds your service quota, Amazon SNS stops publishing SMS messages.
- Because Amazon SNS is a distributed system, it stops sending SMS messages within minutes of the spend quota being exceeded. During this interval, if you continue to send SMS messages, you might incur costs that exceed your quota.

6. (Optional) For **Default sender ID**, enter a custom ID, such as your business brand, which is displayed as the sender of the receiving device.

**Note**

Support for sender IDs varies by country.

7. (Optional) Enter the name of the **Amazon S3 bucket name for usage reports**.

**Note**

The S3 bucket policy must grant write access to Amazon SNS.

8. Choose **Save changes**.

## Setting preferences (AWS SDKs)

To set your SMS preferences using one of the AWS SDKs, use the action in that SDK that corresponds to the `SetSMSAttributes` request in the Amazon SNS API. With this request, you assign values to the different SMS attributes, such as your monthly spend quota and your default SMS type (promotional or transactional). For all SMS attributes, see [SetSMSAttributes](#) in the *Amazon Simple Notification Service API Reference*.

The following code examples show how to set the default settings for sending SMS messages using Amazon SNS.

C++

### SDK for C++

How to use Amazon SNS to set the `DefaultSMSType` attribute.

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String sms_type = argv[1];

    Aws::SNS::Model::SetSMSAttributesRequest ssmst_req;
    ssmst_req.AddAttributes("DefaultSMSType", sms_type);

    auto ssmst_out = sns.SetSMSAttributes(ssmst_req);

    if (ssmst_out.IsSuccess())
    {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else
    {
        std::cout << "Error while setting SMS Type: '" <<
        ssmst_out.GetError().GetMessage()
        << "'" << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSmsAttributes](#) in *AWS SDK for C++ API Reference*.

Java

### SDK for Java 2.x

```
public static void setSNSAttributes( SnsClient snsClient, HashMap<String, String> attributes ) {  
  
    try {  
        SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()  
            .attributes(attributes)  
            .build();  
  
        SetSmsAttributesResponse result = snsClient.setSMSAttributes(request);  
        System.out.println("Set default Attributes to " + attributes + ".  
Status was " + result.sdkHttpResponse().statusCode());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSmsAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create SNS service object.  
const snsClient = new SNSClient({ region: REGION });  
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {SetSMSAttributesCommand} from "@aws-sdk/client-sns";  
import {snsClient} from "./libs/snsClient.js";  
  
// Set the parameters  
const params = {  
    attributes: {  
        /* required */  
        DefaultSMSType: "Transactional" /* highest reliability */,  
        //DefaultSMSType: 'Promotional' /* lowest cost */  
    },  
};  
  
const run = async () => {  
    try {  
        const data = await snsClient.send(new SetSMSAttributesCommand(params));  
        console.log("Success.", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err.stack);  
    }  
};  
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [SetSmsAttributes](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->SetSMSAttributes([
        'attributes' => [
            'DefaultSMSType' => 'Transactional',
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [SetSmsAttributes](#) in [AWS SDK for PHP API Reference](#).

## Sending SMS messages

This section describes how to send SMS messages.

### Topics

- [Publishing to a topic \(p. 192\)](#)
- [Publishing to a mobile phone \(p. 196\)](#)

## Publishing to a topic

You can publish a single SMS message to many phone numbers at once by subscribing those phone numbers to an Amazon SNS topic. An SNS topic is a communication channel to which you can add subscribers and then publish messages to all of those subscribers. A subscriber receives all messages published to the topic until you cancel the subscription, or until the subscriber opts out of receiving SMS messages from your AWS account.

### Topics

- [Sending a message to a topic \(console\) \(p. 193\)](#)
- [Sending a message to a topic \(AWS SDKs\) \(p. 194\)](#)

## Sending a message to a topic (console)

### To create a topic

Complete the following steps if you don't already have a topic to which you want to send SMS messages.

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, choose an [AWS Region that supports SMS messaging \(p. 219\)](#).
3. In the navigation pane, choose **Topics**.
4. On the **Topics** page, choose **Create topic**.
5. On the **Create topic** page, under **Details**, do the following:
  - a. For **Type**, choose **Standard**.
  - b. For **Name**, enter a topic name.
  - c. (Optional) For **Display name**, enter a custom prefix for your SMS messages. When you send a message to the topic, Amazon SNS prepends the display name followed by a right angle bracket (>) and a space. Display names are not case sensitive, and Amazon SNS converts display names to uppercase characters. For example, if the display name of a topic is **MyTopic** and the message is **Hello World!**, the message appears as:

```
MYTOPIC> Hello World!
```

6. Choose **Create topic**. The topic's name and Amazon Resource Name (ARN) appear on the **Topics** page.

### To create an SMS subscription

You can use subscriptions to send an SMS message to multiple recipients by publishing the message only once to your topic.

#### Note

When you start using Amazon SNS to send SMS messages, your AWS account is in the *SMS sandbox*. The SMS sandbox provides a safe environment for you to try Amazon SNS features without risking your reputation as an SMS sender. While your account is in the SMS sandbox, you can use all of the features of Amazon SNS, but you can send SMS messages only to verified destination phone numbers. For more information, see [SMS sandbox \(p. 167\)](#).

1. Sign in to the [Amazon SNS console](#).
2. In the navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, under **Details**, do the following:
  - a. For **Topic ARN**, enter or choose the Amazon Resource Name (ARN) of the topic to which you want to send SMS messages.
  - b. For **Protocol**, choose **SMS**.
  - c. For **Endpoint**, enter the phone number that you want to subscribe to your topic.
5. Choose **Create subscription**. The subscription information appears on the **Subscriptions** page.

To add more phone numbers, repeat these steps. You can also add other types of subscriptions, such as email.

### To send a message

When you publish a message to a topic, Amazon SNS attempts to deliver that message to every phone number that is subscribed to the topic.

1. In the [Amazon SNS console](#), on the **Topics** page, choose the name of the topic to which you want to send SMS messages.
  2. On the topic details page, choose **Publish message**.
  3. On the **Publish message to topic** page, under **Message details**, do the following:
    - a. For **Subject**, keep the field blank unless your topic contains email subscriptions and you want to publish to both email and SMS subscriptions. Amazon SNS uses the **Subject** that you enter as the email subject line.
    - b. (Optional) For **Time to Live (TTL)**, enter a number of seconds that Amazon SNS has to send your SMS message to any mobile application endpoint subscribers.
  4. Under **Message body**, do the following:
    - a. For **Message structure**, choose **Identical payload for all delivery protocols** to send the same message to all protocol types subscribed to your topic. Or, choose **Custom payload for each delivery protocol** to customize the message for subscribers of different protocol types. For example, you can enter a default message for phone number subscribers and a custom message for email subscribers.
    - b. For **Message body to send to the endpoint**, enter your message, or your custom messages per delivery protocol.
- If your topic has a display name, Amazon SNS adds it to the message, which increases the message length. The display name length is the number of characters in the name plus two characters for the right angle bracket (>) and the space that Amazon SNS adds.
- For information about the size quotas for SMS messages, see [Publishing to a mobile phone \(p. 196\)](#).
5. (Optional) For **Message attributes**, add message metadata such as timestamps, signatures, and IDs.
  6. Choose **Publish message**. Amazon SNS sends the SMS message and displays a success message.

## Sending a message to a topic (AWS SDKs)

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code example shows how to:

- Create an Amazon SNS topic.
- Subscribe phone numbers to the topic.
- Publish SMS messages to the topic so that all subscribed phone numbers receive the message at once.

Java

### SDK for Java 1.x

Create a topic and return its ARN.

```
public static String createSNSTopic(AmazonSNSClient snsClient) {  
    CreateTopicRequest createTopic = new CreateTopicRequest("mySNSTopic");  
    CreateTopicResult result = snsClient.createTopic(createTopic);  
    System.out.println("Create topic request: " +  
        snsClient.getCachedResponseMetadata(createTopic));  
    System.out.println("Create topic result: " + result);  
    return result.getTopicArn();  
}
```

Subscribe an endpoint to a topic.

```
public static void subscribeToTopic(AmazonSNSClient snsClient, String topicArn,
    String protocol, String endpoint) {
    SubscribeRequest subscribe = new SubscribeRequest(topicArn, protocol,
    endpoint);
    SubscribeResult subscribeResult = snsClient.subscribe(subscribe);
    System.out.println("Subscribe request: " +
        snsClient.getCachedResponseMetadata(subscribe));
    System.out.println("Subscribe result: " + subscribeResult);
}
```

Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```
public static void addMessageAttributes(Map<String, MessageAttributeValue>
    smsAttributes) {
    smsAttributes.put("AWS.SNS.SMS.SenderID", new MessageAttributeValue()
        .withStringValue("mySenderId") //The sender ID shown on the device.
        .withDataType("String"));
    smsAttributes.put("AWS.SNS.SMS.MaxPrice", new MessageAttributeValue()
        .withStringValue("0.50") //Sets the max price to 0.50 USD.
        .withDataType("Number"));
    smsAttributes.put("AWS.SNS.SMS.SMSType", new MessageAttributeValue()
        .withStringValue("Promotional") //Sets the type to promotional.
        .withDataType("String"));
}
```

Publish a message to a topic. The message is sent to every subscriber.

```
public static void sendSMSMessageToTopic(AmazonSNSClient snsClient, String
    topicArn,
    String message, Map<String, MessageAttributeValue> smsAttributes) {
    PublishResult result = snsClient.publish(new PublishRequest()
        .withTopicArn(topicArn)
        .withMessage(message)
        .withMessageAttributes(smsAttributes));
    System.out.println(result);
}
```

Call the previous functions to create a topic, subscribe a phone number, set message attributes, and publish a message to the topic.

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();

    String topicArn = createSNSTopic(snsClient);
    String phoneNumber = "+1XXX5550100";
    // Specify a protocol of "sms" when subscribing a phone number.
    subscribeToTopic(snsClient, topicArn, "sms", phoneNumber);

    String message = "My SMS message";
    Map<String, MessageAttributeValue> smsAttributes =
        new HashMap<String, MessageAttributeValue>();
    addMessageAttributes(smsAttributes)
    sendSMSMessageToTopic(snsClient, topicArn, message, smsAttributes);
}
```

- Find instructions and more code on [GitHub](#).

## Publishing to a mobile phone

You can use Amazon SNS to send SMS messages directly to a mobile phone without subscribing the phone number to an Amazon SNS topic.

### Note

Subscribing phone numbers to a topic is useful if you want to send one message to multiple phone numbers at once. For instructions on publishing an SMS message to a topic, see [Publishing to a topic \(p. 192\)](#).

When you send a message, you can control whether the message is optimized for cost or reliable delivery. You can also specify a [sender ID or origination number \(p. 169\)](#). If you send the message programmatically using the Amazon SNS API or the AWS SDKs, you can specify a maximum price for the message delivery.

Each SMS message can contain up to 140 bytes, and the character quota depends on the encoding scheme. For example, an SMS message can contain:

- 160 GSM characters
- 140 ASCII characters
- 70 UCS-2 characters

If you publish a message that exceeds the size quota, Amazon SNS sends it as multiple messages, each fitting within the size quota. Messages are not cut off in the middle of a word, but instead on whole-word boundaries. The total size quota for a single SMS publish action is 1,600 bytes.

When you send an SMS message, you specify the phone number using the E.164 format, a standard phone numbering structure used for international telecommunication. Phone numbers that follow this format can have a maximum of 15 digits along with the prefix of a plus sign (+) and the country code. For example, a US phone number in E.164 format appears as +1XXX5550100.

### Topics

- [Sending a message \(console\) \(p. 196\)](#)
- [Sending a message \(AWS SDKs\) \(p. 197\)](#)

## Sending a message (console)

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, choose an [AWS Region that supports SMS messaging \(p. 219\)](#).
3. In the navigation pane, choose **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, choose **Publish text message**.
5. On the **Publish SMS message** page, for **Message type**, choose one of the following:
  - **Promotional** – Non-critical messages, such as marketing messages.
  - **Transactional** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication.

### Note

This message-level setting overrides your account-level default message type. You can set an account-level default message type from the **Text messaging preferences** section of the **Mobile text messaging (SMS)** page.

For pricing information for promotional and transactional messages, see [Worldwide SMS Pricing](#).

6. For **Destination phone number**, enter the phone number to which you want to send the message.
7. For **Message**, enter the message to send.
8. (Optional) Under **Origination identities**, specify how to identify yourself to your recipients:
  - To specify a **Sender ID**, type a custom ID that contains 3-11 alphanumeric characters, including at least one letter and no spaces. The sender ID is displayed as the message sender on the receiving device. For example, you can use your business brand to make the message source easier to recognize.

Support for sender IDs varies by country and/or region. For example, messages delivered to U.S. phone numbers will not display the sender ID. For the countries and regions that support sender IDs, see [Supported Regions and countries \(p. 219\)](#).

If you do not specify a sender ID, one of the following is displayed as the originating identity:

- In countries that support long codes, the long code is shown.
- In countries where only sender IDs are supported, *NOTICE* is shown.

This message-level sender ID overrides your default sender ID, which you set on the **Text messaging preferences** page.

- To specify an **Origination number**, enter a string of 5-14 numbers to display as the sender's phone number on the receiver's device. This string must match an origination number that is configured in your AWS account for the destination country. The origination number can be a 10DLC number, toll-free number, person-to-person long code, or short codes. For more information, see [Origination identities for SMS messages \(p. 169\)](#).

If you don't specify an origination number, Amazon SNS selects an origination number to use for the SMS text message, based on your AWS account configuration.

9. If you're sending SMS messages to recipients in India, expand **Country-specific attributes**, and specify the following attributes:

- **Entity ID** – The entity ID or principal entity (PE) ID for sending SMS messages to recipients in India. This ID is a unique string of 1–50 characters that the Telecom Regulatory Authority of India (TRAI) provides to identify the entity that you registered with the TRAI.
- **Template ID** – The template ID for sending SMS messages to recipients in India. This ID is a unique, TRAI-provided string of 1–50 characters that identifies the template that you registered with the TRAI. The template ID must be associated with the sender ID that you specified for the message.

For more information on sending SMS messages to recipients in India, see [Special requirements for sending SMS messages to recipients in India \(p. 231\)](#).

10. Choose **Publish message**.

#### Tip

To send SMS messages from an origination number, you can also choose **Origination numbers** in the Amazon SNS console navigation panel. Choose an origination number that includes **SMS** in the **Capabilities** column, and then choose **Publish text message**.

## Sending a message (AWS SDKs)

To send an SMS message using one of the AWS SDKs, use the API operation in that SDK that corresponds to the `Publish` request in the Amazon SNS API. With this request, you can send an SMS message directly to a phone number. You can also use the `MessageAttributes` parameter to set values for the following attribute names:

#### AWS.SNS.SMS.SenderID

A custom ID that contains 3–11 alphanumeric characters or hyphen (-) characters, including at least one letter and no spaces. The sender ID appears as the message sender on the receiving device. For example, you can use your business brand to help make the message source easier to recognize.

Support for sender IDs varies by country or region. For example, messages delivered to US phone numbers don't display the sender ID. For a list of the countries or regions that support sender IDs, see [Supported Regions and countries \(p. 219\)](#).

If you don't specify a sender ID, a [long code \(p. 180\)](#) appears as the sender ID in supported countries or regions. For countries or regions that require an alphabetic sender ID, *NOTICE* appears as the sender ID.

This message-level attribute overrides the account-level attribute `DefaultSenderId`, which you can set using the `SetSMSAttributes` request.

#### AWS.MM.SMS.OriginationNumber

A custom string of 5–14 numbers, which can include an optional leading plus sign (+). This string of numbers appears as the sender's phone number on the receiving device. The string must match an origination number that's configured in your AWS account for the destination country. The origination number can be a 10DLC number, toll-free number, person-to-person (P2P) long code, or short code. For more information, see [Origination numbers \(p. 170\)](#).

If you don't specify an origination number, Amazon SNS chooses an origination number based on your AWS account configuration.

#### AWS.SNS.SMS.MaxPrice

The maximum price in USD that you're willing to spend to send the SMS message. If Amazon SNS determines that sending the message would incur a cost that exceeds your maximum price, it doesn't send the message.

This attribute has no effect if your month-to-date SMS costs have already exceeded the quota set for the `MonthlySpendLimit` attribute. You can set the `MonthlySpendLimit` attribute using the `SetSMSAttributes` request.

If you're sending the message to an Amazon SNS topic, the maximum price applies to each message delivery to each phone number that is subscribed to the topic.

#### AWS.SNS.SMS.SMSType

The type of message that you're sending:

- **Promotional** (default) – Non-critical messages, such as marketing messages.
- **Transactional** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication.

This message-level attribute overrides the account-level attribute `DefaultSMSType`, which you can set using the `SetSMSAttributes` request.

#### AWS.MM.SMS.EntityId

This attribute is required only for sending SMS messages to recipients in India.

This is your entity ID or principal entity (PE) ID for sending SMS messages to recipients in India. This ID is a unique string of 1–50 characters that the Telecom Regulatory Authority of India (TRAI) provides to identify the entity that you registered with the TRAI.

#### AWS.MM.SMS.TemplateId

This attribute is required only for sending SMS messages to recipients in India.

This is your template for sending SMS messages to recipients in India. This ID is a unique, TRAI-provided string of 1–50 characters that identifies the template that you registered with the TRAI. The template ID must be associated with the sender ID that you specified for the message.

## Sending a message

The following code examples show how to publish SMS messages using Amazon SNS.

C++

### SDK for C++

```
/**  
 * Publish SMS: use Amazon SNS to send an SMS text message to a phone number.  
 * Note: This requires additional AWS configuration prior to running example.  
 *  
 * NOTE: When you start using Amazon SNS to send SMS messages, your AWS account is  
 in the SMS sandbox and you can only  
 use verified destination phone numbers. See https://docs.aws.amazon.com/sns/latest/dg/sns-sms-sandbox.html.  
 * NOTE: If destination is in the US, you also have an additional restriction that  
 you have use a dedicated  
 origination ID (phone number). You can request an origination number using  
 Amazon Pinpoint for a fee.  
 * See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-origination-numbers-with-amazon-sns/  
 * for more information.  
 *  
 * <phone_number_value> input parameter uses E.164 format.  
 * For example, in United States, this input value should be of the form:  
 +12223334444  
 */  
int main(int argc, char ** argv)  
{  
    if (argc != 3)  
    {  
        std::cout << "Usage: publish_sms <message_value> <phone_number_value> " <<  
        std::endl;  
        return 1;  
    }  
  
    Aws::SDKOptions options;  
    Aws::InitAPI(options);  
    {  
        Aws::SNS::SNSClient sns;  
        Aws::String message = argv[1];  
        Aws::String phone_number = argv[2];  
  
        Aws::SNS::Model::PublishRequest psms_req;  
        psms_req.SetMessage(message);  
        psms_req.SetPhoneNumber(phone_number);  
  
        auto psms_out = sns.Publish(psms_req);  
  
        if (psms_out.IsSuccess())  
        {  
            std::cout << "Message published successfully " <<  
            psms_out.GetResult().GetMessageId()  
            << std::endl;  
        }  
        else  
        {  
    }
```

```
    std::cout << "Error while publishing message " <<
psms_out.GetError().GetMessage()
    << std::endl;
}
}

Aws::ShutdownAPI(options);
return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for C++ API Reference*.

## Java

### SDK for Java 2.x

```
public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .phoneNumber(phoneNumber)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out.println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Java 2.x API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Sends a a text message (SMS message) directly to a phone number using Amazon
 * SNS.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    // ...
]);
```

```
'profile' => 'default',
'region' => 'us-east-1',
'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';

try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'PhoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def publish_text_message(self, phone_number, message):
        """
        Publishes a text message directly to a phone number without need for a
        subscription.

        :param phone_number: The phone number that receives the message. This must
        be
                in E.164 format. For example, a United States phone
                number might be +12065550101.
        :param message: The message to send.
        :return: The ID of the message.
        """
        try:
            response = self.sns_resource.meta.client.publish(
                PhoneNumber=phone_number, Message=message)
            message_id = response['MessageId']
            logger.info("Published message to %s.", phone_number)
        except ClientError:
            logger.exception("Couldn't publish message to %s.", phone_number)
            raise
        else:
            return message_id
```

- Find instructions and more code on [GitHub](#).

- For API details, see [Publish](#) in *AWS SDK for Python (Boto3) API Reference*.

## Monitoring SMS activity

By monitoring your SMS activity, you can keep track of destination phone numbers, successful or failed deliveries, reasons for failure, costs, and other information. Amazon SNS helps by summarizing statistics in the console, sending information to Amazon CloudWatch, and sending daily SMS usage reports to an Amazon S3 bucket that you specify.

### Topics

- [Viewing SMS delivery statistics \(p. 202\)](#)
- [Viewing Amazon CloudWatch metrics and logs for SMS deliveries \(p. 202\)](#)
- [Viewing daily SMS usage reports \(p. 204\)](#)

## Viewing SMS delivery statistics

You can use the Amazon SNS console to view statistics about your recent SMS deliveries.

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, set the region selector to a [region that supports SMS messaging \(p. 219\)](#).
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Text messaging (SMS)** page, in the **Account stats** section, view the charts for your transactional and promotional SMS message deliveries. Each chart shows the following data for the preceding 15 days:
  - Delivery rate (percentage of successful deliveries)
  - Sent (number of delivery attempts)
  - Failed (number of delivery failures)

On this page, you can also choose the **Usage** button to go to the Amazon S3 bucket where you store your daily usage reports. For more information, see [Viewing daily SMS usage reports \(p. 204\)](#).

## Viewing Amazon CloudWatch metrics and logs for SMS deliveries

You can use Amazon CloudWatch and Amazon CloudWatch Logs to monitor your SMS message deliveries.

### Topics

- [Viewing Amazon CloudWatch metrics \(p. 202\)](#)
- [Viewing CloudWatch Logs \(p. 203\)](#)
- [Example log for successful SMS delivery \(p. 203\)](#)
- [Example log for failed SMS delivery \(p. 204\)](#)
- [SMS delivery failure reasons \(p. 204\)](#)

## Viewing Amazon CloudWatch metrics

Amazon SNS automatically collects metrics about your SMS message deliveries and pushes them to Amazon CloudWatch. You can use CloudWatch to monitor these metrics and create alarms to alert you

when a metric crosses a threshold. For example, you can monitor CloudWatch metrics to learn your SMS delivery rate and your month-to-date SMS charges.

For information about monitoring CloudWatch metrics, setting CloudWatch alarms, and the types of metrics available, see [Monitoring Amazon SNS topics using CloudWatch \(p. 390\)](#).

## Viewing CloudWatch Logs

You can collect information about successful and unsuccessful SMS message deliveries by enabling Amazon SNS to write to Amazon CloudWatch Logs. For each SMS message that you send, Amazon SNS writes a log that includes the message price, the success or failure status, the reason for failure (if the message failed), the message dwell time, and other information.

### To enable and view CloudWatch Logs for your SMS messages

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, set the region selector to a [region that supports SMS messaging \(p. 219\)](#).
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
5. On the next page, expand the **Delivery status logging** section.
6. For **Success sample rate**, specify the percentage of successful SMS deliveries for which Amazon SNS will write logs in CloudWatch Logs. For example:
  - To write logs only for failed deliveries, set this value to 0.
  - To write logs for 10% of your successful deliveries, set it to 10.

If you don't specify a percentage, Amazon SNS writes logs for all successful deliveries.

7. To provide the required permissions, do one of the following:
  - To create a new service role, choose **Create new service role** and then **Create new roles**. On the next page, choose **Allow** to give Amazon SNS write access to your account's resources.
  - To use an existing service role, choose **Use existing service role** and then paste the ARN name in the **IAM role for successful and failed deliveries** box.
8. Choose **Save changes**.
9. Back on the **Mobile text messaging (SMS)** page, go to the **Delivery status logs** section to view any available logs.

#### Note

Depending on the destination phone number's carrier, it can take up to 72 hours for delivery logs to appear in the Amazon SNS console.

## Example log for successful SMS delivery

The delivery status log for a successful SMS delivery will resemble the following example:

```
{  
    "notification": {  
        "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",  
        "timestamp": "2016-06-28 00:40:34.558"  
    },  
    "delivery": {  
        "phoneCarrier": "My Phone Carrier",  
        "mnc": 270,  
        "status": "Success",  
        "price": 0.01,  
        "dwellTime": 1000000000000000000,  
        "error": null  
    }  
}
```

```
        "destination": "+1XXX5550100",
        "priceInUSD": 0.00645,
        "smsType": "Transactional",
        "mcc": 310,
        "providerResponse": "Message has been accepted by phone carrier",
        "dwellTimeMs": 599,
        "dwellTimeMsUntilDeviceAck": 1344
    },
    "status": "SUCCESS"
}
```

## Example log for failed SMS delivery

The delivery status log for a failed SMS delivery will resemble the following example:

```
{
    "notification": {
        "messageId": "1077257a-92f3-5ca3-bc97-6a915b310625",
        "timestamp": "2016-06-28 00:40:34.559"
    },
    "delivery": {
        "mnc": 0,
        "destination": "+1XXX5550100",
        "priceInUSD": 0.00645,
        "smsType": "Transactional",
        "mcc": 0,
        "providerResponse": "Unknown error attempting to reach phone",
        "dwellTimeMs": 1420,
        "dwellTimeMsUntilDeviceAck": 1692
    },
    "status": "FAILURE"
}
```

## SMS delivery failure reasons

The reason for a failure is provided with the providerResponse attribute. SMS messages might fail to deliver for the following reasons:

- Blocked as spam by phone carrier
- Destination is on a blocked list
- Invalid phone number
- Message body is invalid
- Phone carrier has blocked this message
- Phone carrier is currently unreachable/unavailable
- Phone has blocked SMS
- Phone is on a blocked list
- Phone is currently unreachable/unavailable
- Phone number is opted out
- This delivery would exceed max price
- Unknown error attempting to reach phone

## Viewing daily SMS usage reports

You can monitor your SMS deliveries by subscribing to daily usage reports from Amazon SNS. For each day that you send at least one SMS message, Amazon SNS delivers a usage report as a CSV file to the specified Amazon S3 bucket. It takes 24 hours for the SMS usage report to be available in the S3 bucket.

## Topics

- [Daily usage report information \(p. 205\)](#)
- [Subscribing to daily usage reports \(p. 205\)](#)

## Daily usage report information

The usage report includes the following information for each SMS message that you send from your account.

Note that the report does not include messages that are sent to recipients who have opted out.

- Time of publication for message (in UTC)
- Message ID
- Destination phone number
- Message type
- Delivery status
- Message price (in USD)
- Part number (a message is split into multiple parts if it is too long for a single message)
- Total number of parts

### Note

If Amazon SNS did not receive the part number, we set its value to zero.

## Subscribing to daily usage reports

To subscribe to daily usage reports, you must create an Amazon S3 bucket with the appropriate permissions.

### To create an Amazon S3 bucket for your daily usage reports

1. From the AWS account that sends SMS messages, sign in to the [Amazon S3 console](#).
2. Choose **Create Bucket**.
3. For **Bucket Name**, we recommend that you enter a name that is unique for your account and your organization. For example, use the pattern <my-bucket-prefix>-<account\_id>-<org-id>.

For information about conventions and restrictions for bucket names, see [Rules for Bucket Naming](#) in the *Amazon Simple Storage Service User Guide*.

4. Choose **Create**.
5. In the **All Buckets** table, choose the bucket.
6. In the **Permissions** tab, choose **Bucket policy**.
7. In the **Bucket Policy Editor** window, provide a policy that allows the Amazon SNS service principal to write to your bucket. For an example, see [Example bucket policy \(p. 206\)](#).

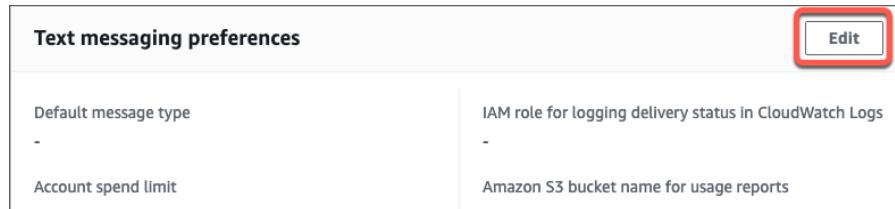
If you use the example policy, remember to replace *my-s3-bucket* with the bucket name that you chose in Step 3.

8. Choose **Save**.

### To subscribe to daily usage reports

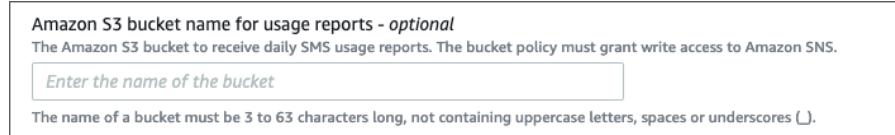
1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Text messaging (SMS)**.

3. On the **Text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.



Text messaging preferences	
Default message type	IAM role for logging delivery status in CloudWatch Logs -
Account spend limit	Amazon S3 bucket name for usage reports

4. On the **Edit text messaging preferences** page, in the **Details** section, specify the **Amazon S3 bucket name for usage reports**.



Amazon S3 bucket name for usage reports - *optional*  
The Amazon S3 bucket to receive daily SMS usage reports. The bucket policy must grant write access to Amazon SNS.

The name of a bucket must be 3 to 63 characters long, not containing uppercase letters, spaces or underscores ( ).

5. Choose **Save changes**.

### Example bucket policy

The following policy allows the Amazon SNS service principal to perform the `s3:PutObject`, `s3:GetBucketLocation`, and `s3>ListBucket` actions. You can use this example when you create an Amazon S3 bucket to receive daily SMS usage reports from Amazon SNS.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPutObject",
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my-s3-bucket/*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        }
      }
    },
    {
      "Sid": "AllowGetBucketLocation",
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "s3:GetBucketLocation",
      "Resource": "arn:aws:s3:::my-s3-bucket",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        }
      }
    },
    {
      "Sid": "AllowListBucket",
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
    }
  ]
}
```

```
        "Action": "s3>ListBucket",
        "Resource": "arn:aws:s3:::my-s3-bucket"
    }
}
```

#### Note

You can publish usage reports to Amazon S3 buckets that are owned by the AWS account that's specified in the Condition element in the Amazon S3 policy. To publish usage reports to an Amazon S3 bucket that another AWS account owns, see [How can I copy S3 objects from another AWS account?](#).

#### Example daily usage report

After you subscribe to daily usage reports, each day, Amazon SNS puts a CSV file with usage data in the following location:

```
<my-s3-bucket>/SMSUsageReports/<region>/YYYY/MM/DD/00x.csv.gz
```

Each file can contain up to 50,000 records. If the records for a day exceed this quota, Amazon SNS will add multiple files.

The following shows an example report:

```
PublishTimeUTC,MessageId,DestinationPhoneNumber,MessageType,DeliveryStatus,PriceInUSD,PartNumber,TotalP
2016-05-10T03:00:29.476Z,96a298ac-1458-4825-
a7eb-7330e0720b72,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.90084,0,1
2016-05-10T03:00:29.561Z,1e29d394-
d7f4-4dc9-996e-26412032c344,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.34322,0,1
2016-05-10T03:00:30.769Z,98ba941c-afc7-4c51-
ba2c-56c6570a6c08,1XXX5550100,Transactional,Message has been accepted by phone
carrier,0.27815,0,1
```

## Managing phone numbers and SMS subscriptions

Amazon SNS provides several options for managing who receives SMS messages from your account. With a limited frequency, you can opt in phone numbers that have opted out of receiving SMS messages from your account. To stop sending messages to SMS subscriptions, you can remove subscriptions or the topics that publish to them.

#### Topics

- [Opting out of receiving SMS messages \(p. 207\)](#)
- [Managing phone numbers and subscriptions \(console\) \(p. 208\)](#)
- [Managing phone numbers and subscriptions \(AWS SDKs\) \(p. 209\)](#)

## Opting out of receiving SMS messages

Where required by local laws and regulations (such as the US and Canada), SMS recipients can use their devices to opt out by replying to the message with any of the following:

- ARRET (French)
- CANCEL
- END

- OPT-OUT
- OPTOUT
- QUIT
- REMOVE
- STOP
- TD
- UNSUBSCRIBE

To opt out, the recipient must reply to the same [origination number \(p. 170\)](#) that Amazon SNS used to deliver the message. After opting out, the recipient will no longer receive SMS messages delivered from your AWS account unless you opt in the phone number.

If the phone number is subscribed to an Amazon SNS topic, opting out does not remove the subscription, but SMS messages will fail to deliver to that subscription unless you opt in the phone number.

## Managing phone numbers and subscriptions (console)

You can use the Amazon SNS console to control which phone numbers receive SMS messages from your account.

### Opting in a phone number that has been opted out

You can view which phone numbers have been opted out of receiving SMS messages from your account, and you can opt in these phone numbers to resume sending messages to them.

You can opt in a phone number only once every 30 days.

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, set the region selector to a [region that supports SMS messaging \(p. 219\)](#).
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Text messaging (SMS)** page, choose **View opted out phone numbers**. The **Opted out phone numbers** page displays the opted out phone numbers.
5. Select the check box for the phone number that you want to opt in, and choose **Opt in**. The phone number is no longer opted out and will receive SMS messages that you send to it.

### Deleting an SMS subscription

Delete an SMS subscription to stop sending SMS messages to that phone number when you publish to your topics.

1. On the navigation panel, choose **Subscriptions**.
2. Select the check boxes for the subscriptions that you want to delete. Then choose **Actions**, and choose **Delete Subscriptions**.
3. In the **Delete** window, choose **Delete**. Amazon SNS deletes the subscription and displays a success message.

### Deleting a topic

Delete a topic when you no longer want to publish messages to its subscribed endpoints.

1. On the navigation panel, choose **Topics**.
2. Select the check boxes for the topics that you want to delete. Then choose **Actions**, and choose **Delete Topics**.

3. In the **Delete** window, choose **Delete**. Amazon SNS deletes the topic and displays a success message.

## Managing phone numbers and subscriptions (AWS SDKs)

You can use the AWS SDKs to make programmatic requests to Amazon SNS and manage which phone numbers can receive SMS messages from your account.

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

### Viewing all opted out phone numbers

To view all opted out phone numbers, submit a `ListPhoneNumbersOptedOut` request with the Amazon SNS API.

The following code examples show how to list phone numbers that are opted out of receiving Amazon SNS messages.

Java

#### SDK for Java 2.x

```
public static void listOpts( SnsClient snsClient) {  
  
    try {  
  
        ListPhoneNumbersOptedOutRequest request =  
ListPhoneNumbersOptedOutRequest.builder().build();  
        ListPhoneNumbersOptedOutResponse result =  
snsClient.listPhoneNumbersOptedOut(request);  
        System.out.println("Status is " + result.sdkHttpResponse().statusCode()  
+ "\n\nPhone Numbers: \n\n" + result.phoneNumbers());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListPhoneNumbersOptedOut](#) in *AWS SDK for Java 2.x API Reference*.

PHP

#### SDK for PHP

```
require 'vendor/autoload.php';  
  
use Aws\Sns\SnsClient;  
use Aws\Exception\AwsException;  
  
/**  
 * Returns a list of phone numbers that are opted out of receiving SMS messages  
from your AWS SNS account.  
*  
* This code expects that you have AWS credentials set up per:
```

```
* https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
*/
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
try {
    $result = $SnSclient->listPhoneNumbersOptedOut([
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [ListPhoneNumbersOptedOut](#) in [AWS SDK for PHP API Reference](#).

## Checking whether a phone number is opted out

To check whether a phone number is opted out, submit a `CheckIfPhoneNumberIsOptedOut` request with the Amazon SNS API.

The following code examples show how to check whether a phone number is opted out of receiving Amazon SNS messages.

.NET

### AWS SDK for .NET

```
/// <summary>
/// Checks to see if the supplied phone number has been opted out.
/// </summary>
/// <param name="client">The initialized Amazon SNS Client object used
/// to check if the phone number has been opted out.</param>
/// <param name="phoneNumber">A string representing the phone number
/// to check.</param>
public static async Task<br/>
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)<br/>
{
    var request = new CheckIfPhoneNumberIsOptedOutRequest<br/>
    {<br/>
        PhoneNumber = phoneNumber,<br/>
    };<br/>
    try<br/>
    {
        var response = await<br/>
client.CheckIfPhoneNumberIsOptedOutAsync(request);<br/>
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)<br/>
        {
            string optOutStatus = response.IsOptedOut ? "opted out" : "not<br/>
opted out.";
```

```
        Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
    }
}
catch (AuthorizationErrorException ex)
{
    Console.WriteLine($"{ex.Message}");
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in *AWS SDK for .NET API Reference*.

## Java

### SDK for Java 2.x

```
public static void checkPhone(SnsClient snsClient, String phoneNumber) {

    try {
        CheckIfPhoneNumberIsOptedOutRequest request =
CheckIfPhoneNumberIsOptedOutRequest.builder()
            .phoneNumber(phoneNumber)
            .build();

        CheckIfPhoneNumberIsOptedOutResponse result =
snsClient.checkIfPhoneNumberIsOptedOut(request);

        System.out.println(result.isOptedOut() + "Phone Number " + phoneNumber
+ " has Opted Out of receiving sns messages." +
"\n\nStatus was " + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";
import {snsClient } from "./libs/snsClient.js";

// Set the parameters
const params = { phoneNumber: "353861230764" }; //PHONE_NUMBER, in the E.164 phone
number structure

const run = async () => {
  try {
    const data = await snsClient.send(
      new CheckIfPhoneNumberIsOptedOutCommand(params)
    );
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Indicates whether the phone number owner has opted out of receiving SMS messages
from your AWS SNS account.
*
* This code expects that you have AWS credentials set up per:
* https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
*/

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$phone = '+1XXX5550100';

try {
    $result = $SnSclient->checkIfPhoneNumberIsOptedOut([
        'phoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in [AWS SDK for PHP API Reference](#).

## Opting in a phone number that has been opted out

To opt in a phone number, submit an `OptInPhoneNumber` request with the Amazon SNS API.

You can opt in a phone number only once every 30 days.

## Deleting an SMS subscription

To delete an SMS subscription from an Amazon SNS topic, get the subscription ARN by submitting a `ListSubscriptions` request with the Amazon SNS API, and then pass the ARN to an `Unsubscribe` request.

The following code examples show how to delete an Amazon SNS subscription.

C++

### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String subscription_arn = argv[1];

    Aws::SNS::Model::UnsubscribeRequest s_req;
    s_req.SetSubscriptionArn(subscription_arn);

    auto s_out = sns.Unsubscribe(s_req);

    if (s_out.IsSuccess())
    {
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else
    {
        std::cout << "Error while unsubscribing " << s_out.GetError().GetMessage()
            << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe](#) in [AWS SDK for C++ API Reference](#).

Java

### SDK for Java 2.x

```
public static void unSub(SnsClient snsClient, String subscriptionArn) {  
  
    try {  
        UnsubscribeRequest request = UnsubscribeRequest.builder()  
            .subscriptionArn(subscriptionArn)  
            .build();  
  
        UnsubscribeResponse result = snsClient.unsubscribe(request);  
  
        System.out.println("\n\nStatus was " +  
result.sdkHttpResponse().statusCode()  
            + "\n\nSubscription was removed for " + request.subscriptionArn());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe in AWS SDK for Java 2.x API Reference](#).

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create SNS service object.  
const snsClient = new SNSClient({ region: REGION });  
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {UnsubscribeCommand} from "@aws-sdk/client-sns";  
import {snsClient} from "./libs/snsClient.js";  
  
// Set the parameters  
const params = { SubscriptionArn: "TOPIC_SUBSCRIPTION_ARN" }; //  
TOPIC_SUBSCRIPTION_ARN  
  
const run = async () => {  
    try {  
        const data = await snsClient.send(new UnsubscribeCommand(params));  
        console.log("Success.", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err.stack);  
    }  
};  
run();
```

- Find instructions and more code on [GitHub](#).

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Unsubscribe in AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Deletes a subscription to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $SnSclient->unsubscribe([
        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Unsubscribe in AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def delete_subscription(subscription):
        """
        Unsubscribes and deletes a subscription.
        
```

```
"""
try:
    subscription.delete()
    logger.info("Deleted subscription %s.", subscription.arn)
except ClientError:
    logger.exception("Couldn't delete subscription %s.", subscription.arn)
    raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe in AWS SDK for Python \(Boto3\) API Reference](#).

## Deleting a topic

To delete a topic and all of its subscriptions, get the topic ARN by submitting a `ListTopics` request with the Amazon SNS API, and then pass the ARN to the `DeleteTopic` request.

The following code examples show how to delete an Amazon SNS topic and all subscriptions to that topic.

.NET

### AWS SDK for .NET

```
/// <summary>
/// This example deletes an existing Amazon Simple Notification Service
/// (Amazon SNS) topic. The example was created using the AWS SDK for .NET
/// version 3.7 and .NET Core 5.0.
/// </summary>
public class DeleteSNSTopic
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-east-2:704825161248:ExampleSNSTopic";
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var response = await client.DeleteTopicAsync(topicArn);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic in AWS SDK for .NET API Reference](#).

C++

### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::String topic_arn = argv[1];
    Aws::SNS::SNSClient sns;
```

```
Aws::SNS::Model::DeleteTopicRequest dt_req;
dt_req.SetTopicArn(topic_arn);

auto dt_out = sns.DeleteTopic(dt_req);

if (dt_out.IsSuccess())
{
    std::cout << "Successfully deleted topic " << topic_arn << std::endl;
}
else
{
    std::cout << "Error deleting topic " << topic_arn << ":" <<
        dt_out.GetError().GetMessage() << std::endl;
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for C++ API Reference*.

## Java

### SDK for Java 2.x

```
public static void deleteSNSTopic(SnsClient snsClient, String topicArn ) {

    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js

// Import required AWS SDK clients and commands for Node.js
import {DeleteTopicCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { TopicArn: "TOPIC_ARN" }; //TOPIC_ARN

const run = async () => {
    try {
        const data = await snsClient.send(new DeleteTopicCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};

run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteTopic in AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Deletes a SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
```

```
// output error message if fails
error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for Python (Boto3) API Reference*.

## Supported Regions and countries

Currently, Amazon SNS supports SMS messaging in the following AWS Regions:

Region name	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	sns.us-east-2.amazonaws.com	HTTP and HTTPS
US East (N. Virginia)	us-east-1	sns.us-east-1.amazonaws.com	HTTP and HTTPS
US West (N. California)	us-west-1	sns.us-west-1.amazonaws.com	HTTP and HTTPS
US West (Oregon)	us-west-2	sns.us-west-2.amazonaws.com	HTTP and HTTPS
Asia Pacific (Mumbai)	ap-south-1	sns.ap-south-1.amazonaws.com	HTTP and HTTPS

Region name	Region	Endpoint	Protocol
Asia Pacific (Singapore)	ap-southeast-1	sns.ap-southeast-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Sydney)	ap-southeast-2	sns.ap-southeast-2.amazonaws.com	HTTP and HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	sns.ap-northeast-1.amazonaws.com	HTTP and HTTPS
Canada (Central)	ca-central-1	sns.ca-central-1.amazonaws.com	HTTP and HTTPS
Europe (Frankfurt)	eu-central-1	sns.eu-central-1.amazonaws.com	HTTP and HTTPS
Europe (Ireland)	eu-west-1	sns.eu-west-1.amazonaws.com	HTTP and HTTPS
Europe (London)	eu-west-2	sns.eu-west-2.amazonaws.com	HTTP and HTTPS
Europe (Paris)	eu-west-3	sns.eu-west-3.amazonaws.com	HTTP and HTTPS
Europe (Stockholm)	eu-north-1	sns.eu-north-1.amazonaws.com	HTTP and HTTPS
Middle East (Bahrain)	me-south-1	sns.me-south-1.amazonaws.com	HTTP and HTTPS
South America (São Paulo)	sa-east-1	sns.sa-east-1.amazonaws.com	HTTP and HTTPS
AWS GovCloud (US-West)	us-gov-west-1	sns.us-gov-west-1.amazonaws.com	HTTP and HTTPS

You can use Amazon SNS to send SMS messages to the following countries and regions:

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Afghanistan	AF		
Albania	AL	Yes	
Algeria	DZ		
Andorra	AD	Yes	
Angola	AO	Yes	
Anguilla	AI	Yes	
Antigua and Barbuda	AG	Yes	
Argentina	AR		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Armenia	AM	Yes	
Aruba	AW	Yes	
Australia	AU	Yes	Yes
Austria	AT	Yes	Yes
Azerbaijan	AZ		
Bahamas	BS	Yes	
Bahrain	BH	Yes	Yes
Bangladesh	BD		
Barbados	BB	Yes	
Belarus	BY	Yes <sup>1 (p. 227)</sup>	
Belgium	BE		Yes
Belize	BZ	Yes	
Benin	BJ	Yes	
Bermuda	BM	Yes	
Bhutan	BT	Yes	
Bolivia	BO	Yes	
Bosnia and Herzegovina	BA	Yes	
Botswana	BW	Yes	
Brazil	BR		Yes
Brunei	BN	Yes	
Bulgaria	BG	Yes	
Burkina Faso	BF	Yes	
Burundi	BI	Yes	
Cambodia	KH	Yes	
Cameroon	CM	Yes	
Canada	CA		Yes
Cape Verde	CV	Yes	
Cayman Islands	KY		
Central African Republic	CF	Yes	
Chad	TD	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Chile	CL		Yes
China <sup>2</sup> (p. 227)	CN		For support, contact sales.
Colombia	CO		
Comoros	KM	Yes	
Cook Islands	CK	Yes	
Costa Rica	CR		
Croatia	HR		Yes
Cyprus	CY	Yes	
Czechia	CZ		Yes
Democratic Republic of the Congo	CD		
Denmark	DK	Yes	Yes
Djibouti	DJ	Yes	
Dominica	DM	Yes	
Dominican Republic	DO		
Ecuador	EC		
Egypt	EG	Yes <sup>1</sup> (p. 227)	
El Salvador	SV		
Equatorial Guinea	GQ	Yes	
Estonia	EE	Yes	Yes
Ethiopia	ET		
Faroe Islands	FO	Yes	
Fiji	FJ	Yes	
Finland	FI	Yes	Yes
France	FR	Yes	Yes
French Guiana	GF		
Gabon	GA	Yes	
Gambia	GM	Yes	
Georgia	GE	Yes	
Germany	DE	Yes	Yes

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Ghana	GH		
Gibraltar	GI	Yes	
Greece	GR	Yes	
Greenland	GL	Yes	
Grenada	GD	Yes	
Guadeloupe	GP	Yes	
Guam	GU		
Guatemala	GT		Yes
Guinea	GN	Yes	
Guinea-Bissau	GW	Yes	
Guyana	GY	Yes	
Haiti	HT	Yes	
Honduras	HN		Yes
Hong Kong	HK	Yes	Yes
Hungary	HU		Yes
Iceland	IS	Yes	
India	IN	Yes <sup>3</sup> ( <a href="#">p. 228</a> )	Yes
Indonesia	ID		
Iraq	IQ		
Ireland	IE	Yes	Yes
Israel	IL	Yes	Yes
Italy	IT	Yes	Yes
Ivory Coast	CI		
Jamaica	JM	Yes	
Japan	JP		Yes
Jordan	JO	Yes <sup>1</sup> ( <a href="#">p. 227</a> )	
Kazakhstan	KZ		
Kenya	KE		
Kiribati	KI		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Kuwait	KW	Yes <sup>1</sup> ( <a href="#">p. 227</a> )	
Kyrgyzstan	KG		
Laos	LA		
Latvia	LV	Yes	Yes
Lebanon	LB	Yes	
Lesotho	LS	Yes	
Liberia	LR	Yes	
Libya	LY	Yes	
Liechtenstein	LI	Yes	
Lithuania	LT	Yes	Yes
Luxembourg	LU	Yes	
Macau	MO	Yes	
Madagascar	MG	Yes	
Malawi	MW	Yes	
Malaysia	MY		Yes
Maldives	MV	Yes	
Mali	ML		
Malta	MT	Yes	
Martinique	MQ	Yes	
Mauritania	MR	Yes	
Mauritius	MU	Yes	
Mexico	MX		Yes
Moldova	MD	Yes	
Monaco	MC		
Mongolia	MN	Yes	
Montenegro	ME	Yes	
Montserrat	MS	Yes	
Morocco	MA		
Mozambique	MZ		
Myanmar	MM		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Namibia	NA		
Nepal	NP		
Netherlands	NL	Yes	Yes
Netherlands Antilles	AN	Yes	
New Caledonia	NC	Yes	
New Zealand	NZ		Yes
Nicaragua	NI		
Niger	NE	Yes	
Nigeria	NG	Yes	
North Macedonia	MK	Yes	
Norway	NO	Yes	Yes
Oman	OM	Yes	
Pakistan	PK		
Palau	PW		
Palestinian Territories	PS		
Panama	PA		
Papua New Guinea	PG	Yes	
Paraguay	PY	Yes	
Peru	PE		
Philippines	PH	Yes <sup>1</sup> (p. 227)	Yes
Poland	PL	Yes	Yes
Portugal	PT	Yes	Yes
Puerto Rico	PR		Yes
Qatar	QA	Yes <sup>1</sup> (p. 227)	
Republic of the Congo	CG		
Reunion Island	RE	Yes	
Romania	RO		Yes
Russia	RU	Yes <sup>1</sup> (p. 227)	Yes
Rwanda	RW	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Saint Kitts and Nevis	KN		
Saint Lucia	LC		
Saint Vincent and the Grenadines	VC		
Samoa	WS	Yes	
Sao Tome and Principe	ST	Yes	
Saudi Arabia	SA	Yes <sup>1 (p. 227)</sup>	
Senegal	SN	Yes	
Serbia	RS	Yes	
Seychelles	SC	Yes	
Sierra Leone	SL	Yes	
Singapore	SG	Yes	Yes
Slovakia	SK	Yes	Yes
Slovenia	SI	Yes	Yes
Solomon Islands	SB	Yes	
Somalia	SO	Yes	
South Africa	ZA		Yes
South Korea	KR		
South Sudan	SS	Yes	
Spain	ES	Yes	Yes
Sri Lanka	LK	Yes <sup>1 (p. 227)</sup>	
Suriname	SR	Yes	
Swaziland	SZ	Yes	
Sweden	SE	Yes	Yes
Switzerland	CH	Yes	Yes
Taiwan	TW		Yes
Tajikistan	TJ	Yes	
Tanzania	TZ		
Thailand	TH	Yes <sup>1 (p. 227)</sup>	Yes
Timor-Leste	TL		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS (Amazon Pinpoint only)
Togo	TG	Yes	
Tonga	TO	Yes	
Trinidad and Tobago	TT	Yes	
Tunisia	TN	Yes	
Turkey	TR	Yes <sup>1</sup> (p. 227)	Yes
Turkmenistan	TM	Yes	
Turks and Caicos Islands	TC	Yes	
Uganda	UG	Yes	
Ukraine	UA	Yes	Yes
United Arab Emirates	AE	Yes <sup>1</sup> (p. 227)	
United Kingdom	GB	Yes	Yes
United States	US		Yes
Uruguay	UY		
Uzbekistan	UZ	Yes	
Vanuatu	VU	Yes	
Venezuela	VE		
Vietnam	VN	Yes <sup>1</sup> (p. 227)	
Virgin Islands, British	VG	Yes	
Virgin Islands, US	VI	Yes	
Yemen	YE	Yes	
Zambia	ZM	Yes	
Zimbabwe	ZW	Yes	

## Notes

1. Senders are required to use a pre-registered alphabetic sender ID. To request a sender ID from AWS Support, file a support request. Some countries require senders to meet specific requirements or abide by certain restrictions in order to obtain approval. In these cases, AWS Support might contact you for additional information after you submit your sender ID request.
2. Senders are required to use a pre-registered template for each type of message that they plan to send. If a sender doesn't meet this requirement, their messages will be blocked. To register a template, file a support request. Some countries require senders to meet additional, specific requirements or

abide by certain restrictions in order to obtain approval. In these cases, AWS Support might ask you for additional information.

3. Senders are required to use a pre-registered alphabetic sender ID. Additional registration steps are required. For more information, see [Special requirements for sending SMS messages to recipients in India \(p. 231\)](#).

## SMS best practices

Mobile phone users tend to have a very low tolerance for unsolicited SMS messages. Response rates for unsolicited SMS campaigns will almost always be low, and therefore the return on your investment will be poor.

Additionally, mobile phone carriers continuously audit bulk SMS senders. They throttle or block messages from numbers that they determine to be sending unsolicited messages.

Sending unsolicited content is also a violation of the [AWS acceptable use policy](#). The Amazon SNS team routinely audits SMS campaigns, and might throttle or block your ability to send messages if it appears that you're sending unsolicited messages.

Finally, in many countries, regions, and jurisdictions, there are severe penalties for sending unsolicited SMS messages. For example, in the United States, the Telephone Consumer Protection Act (TCPA) states that consumers are entitled to \$500–\$1,500 in damages (paid by the sender) for each unsolicited message that they receive.

This section describes several best practices that might help you improve your customer engagement and avoid costly penalties. However, note that this section doesn't contain legal advice. Always consult an attorney to obtain legal advice.

### Topics

- [Comply with laws and regulations \(p. 228\)](#)
- [Obtain permission \(p. 229\)](#)
- [Audit your customer lists \(p. 229\)](#)
- [Keep records \(p. 230\)](#)
- [Respond appropriately \(p. 230\)](#)
- [Adjust your sending based on engagement \(p. 230\)](#)
- [Send at appropriate times \(p. 230\)](#)
- [Avoid cross-channel fatigue \(p. 230\)](#)
- [Maintain independent lists \(p. 231\)](#)
- [Use dedicated short codes \(p. 231\)](#)

## Comply with laws and regulations

You can face significant fines and penalties if you violate the laws and regulations of the places where your customers reside. For this reason, it's vital to understand the laws related to SMS messaging in each country or region where you do business.

The following list includes links to key laws that apply to SMS communications in major markets around the world.

- **United states:** The Telephone Consumer Protection Act of 1991, also known as TCPA, applies to certain types of SMS messages. For more information, see the [rules and regulations](#) at the Federal Communications Commission website.

- **United kingdom:** The Privacy and Electronic Communications (EC Directive) Regulations 2003, also known as PECR, applies to certain types of SMS messages. For more information, see [What are PECR?](#) at the website of the UK Information Commissioner's Office.
- **European union:** The Privacy and Electronic Communications Directive 2002, sometimes known as the ePrivacy Directive, applies to some types of SMS messages. For more information, see the [full text of the law](#) at the Europa.eu website.
- **Canada:** The Fighting Internet and Wireless Spam Act, more commonly known as Canada's Anti-Spam Law or CASL, applies to certain types of SMS messages. For more information, see the [full text of the law](#) at the website of the Parliament of Canada.
- **Japan:** The Act on Regulation of Transmission of Specific Electronic Mail may apply to certain types of SMS messages. For more information, see the website of the Japanese Ministry of Internal Affairs and Communications.

As a sender, these laws may apply to you even if you don't reside in one of these countries. Some of the laws in this list were originally created to address unsolicited email or telephone calls, but have been interpreted or expanded to apply to SMS messages as well. Other countries and regions may have their own laws related to the transmission of SMS messages. Consult an attorney in each country or region where your customers are located to obtain legal advice.

## Obtain permission

Never send messages to customers who haven't explicitly asked to receive them.

If customers can sign up to receive your messages by using an online form, add a CAPTCHA to the form to prevent automated scripts from subscribing people without their knowledge.

When you receive an SMS opt-in request, send the customer a message that asks them to confirm that they want to receive messages from you. Don't send that customer any additional messages until they confirm their subscription. A subscription confirmation message might resemble the following example:

```
Text YES to join Example Corp. alerts. 2 msgs/month. Msg & data rates may apply.  
Reply HELP for help, STOP to cancel.
```

Maintain records that include the date, time, and source of each opt-in request and confirmation. This might be useful if a carrier or regulatory agency requests it, and can also help you perform routine audits of your customer list.

Finally, note that transactional SMS messages, such as order confirmations or one-time passwords, typically don't require explicit consent as long as you tell your customers that you're going to send them these messages. However, you should never send marketing messages to customers who only provided you with permission to send them transactional messages.

## Audit your customer lists

If you send recurring SMS campaigns, audit your customer lists on a regular basis. Auditing your customer lists ensures that the only customers who receive your messages are those who are interested in receiving them.

When you audit your list, send each opted-in customer a message that reminds them that they're subscribed, and provides them with information about unsubscribing. A reminder message might resemble the following example:

```
You're subscribed to Example Corp. alerts. Msg & data rates may apply.  
Reply HELP for help, STOP to unsubscribe.
```

## Keep records

Keep records that show when each customer requested to receive SMS messages from you, and which messages you sent to each customer. Many countries and regions around the world require SMS senders to maintain these records in a way that can be easily retrieved. Mobile carriers might also request this information from you at any time. The exact information that you have to provide varies by country or region. For more information about record-keeping requirements, review the regulations about commercial SMS messaging in each country or region where your customers are located.

Occasionally, a carrier or regulatory agency asks us to provide proof that a customer opted to receive messages from you. In these situations, AWS Support contacts you with a list of the information that the carrier or agency requires. If you can't provide the necessary information, we may pause your ability to send additional SMS messages.

## Respond appropriately

When a recipient replies to your messages, make sure that you respond with useful information. For example, when a customer responds to one of your messages with the keyword "HELP", send them information about the program that they're subscribed to, the number of messages you'll send each month, and the ways that they can contact you for more information. A HELP response might resemble the following example:

```
HELP: Example Corp. alerts: email help@example.com or call XXX-555-0199. 2 msgs/month.  
Msg & data rates may apply. Reply STOP to cancel.
```

When a customer replies with the keyword "STOP", let them know that they won't receive any further messages. A STOP response might resemble the following example:

```
STOP: You're unsubscribed from Example Corp. alerts. No more messages will be sent.  
Reply HELP, email help@example.com, or call XXX-555-0199 for more info.
```

## Adjust your sending based on engagement

Your customers' priorities can change over time. If customers no longer find your messages to be useful, they might opt out of your messages entirely, or even report your messages as unsolicited. For these reasons, it's important that you adjust your sending practices based on customer engagement.

For customers who rarely engage with your messages, you should adjust the frequency of your messages. For example, if you send weekly messages to engaged customers, you could create a separate monthly digest for customers who are less engaged.

Finally, remove customers who are completely unengaged from your customer lists. This step prevents customers from becoming frustrated with your messages. It also saves you money and helps protect your reputation as a sender.

## Send at appropriate times

Only send messages during normal daytime business hours. If you send messages at dinner time or in the middle of the night, there's a good chance that your customers will unsubscribe from your lists in order to avoid being disturbed. Furthermore, it doesn't make sense to send SMS messages when your customers can't respond to them immediately.

## Avoid cross-channel fatigue

In your campaigns, if you use multiple communication channels (such as email, SMS, and push messages), don't send the same message in every channel. When you send the same message at the

same time in more than one channel, your customers will probably perceive your sending behavior to be annoying rather than helpful.

## Maintain independent lists

When customers opt in to a topic, make sure that they only receive messages about that topic. Don't send your customers messages from topics that they haven't opted into.

## Use dedicated short codes

If you use short codes, maintain a separate short code for each brand and each type of message. For example, if your company has two brands, use a separate short code for each one. Similarly, if you send both transactional and promotional messages, use a separate short code for each type of message. To learn more about requesting short codes, see [Requesting dedicated short codes for SMS messaging with Amazon SNS \(p. 182\)](#).

## Special requirements for sending SMS messages to US destinations

Effective June 1, 2021, US telecom providers no longer support using person-to-person (P2P) long codes for application-to-person (A2P) communications to US destinations. Instead, you need to use one of the following types of [origination numbers \(p. 170\)](#) for messaging US destinations:

- [Short codes \(p. 179\)](#)
- [10-digit long codes \(10DLC\) \(p. 170\)](#)
- [Toll-free numbers \(p. 178\)](#)

## Special requirements for sending SMS messages to recipients in India

By default, when you send messages to recipients in India, Amazon SNS uses International Long Distance Operator (ILDO) connections to transmit those messages. When recipients see a message that's sent over an ILDO connection, it appears to be sent from a random numeric ID.

### Note

The price for sending messages using local routes is shown on the [Amazon SNS Worldwide SMS Pricing](#) page. The price for sending messages using ILDO connections is higher than the price for sending messages through local routes. Currently, the price for sending ILDO messages is USD \$0.02171 per message.

If you prefer to use an alphabetic sender ID for your SMS messages, you have to send those messages over local routes rather than ILDO routes. To send messages using local routes, you must first register your use case and message templates with the Telecom Regulatory Authority of India (TRAI) through Distributed Ledger Technology (DLT) portals. These registration requirements are designed to reduce the number of unsolicited messages that Indian consumers receive and to protect consumers from potentially harmful messages. This registration process is managed by Vodafone India through its Vilpower service.

### Note

You can't use both numeric sender IDs and alphanumeric sender IDs in the same account. If you use both ID types, you must maintain separate accounts for each. For additional content guidelines, see the Vilpower website at <https://www.vilpower.in>.

### Topics

- [Sending SMS messages to India: task overview \(p. 232\)](#)
- [Step 1: Registering with the TRAI \(p. 232\)](#)
- [Step 2: Requesting a sender ID \(p. 232\)](#)
- [Step 3: Sending SMS messages \(p. 233\)](#)
- [Troubleshooting SMS messages sent to recipients in India \(p. 234\)](#)

## Sending SMS messages to India: task overview

To send SMS messages to India, complete the following tasks:

1. [Register with the TRAI \(p. 232\)](#).
2. [Request a sender ID \(p. 232\)](#).
3. [Send SMS messages \(p. 233\)](#).

## Step 1: Registering with the TRAI

Before you can send SMS messages to recipients in India, you must register your organization with the TRAI. Be prepared to provide the following information during the registration process:

- Your organization's Permanent Account Number (PAN).
- Your organization's Tax Deduction Account Number (TAN).
- Your organization's Goods and Services Tax Identification Number (GSTIN).
- Your organization's Corporate Identity Number (CIN).
- A letter of authorization that gives you the authority to register your organization with Vilpower. The Vilpower website includes a template that you can download and modify to fit your needs.

Vilpower charges a fee for completing the registration process. Currently, this fee is ₹5900.

### To register your organization with the TRAI

1. In a web browser, go to the Vilpower website at <https://www.vilpower.in>.
2. Choose **Signup** to create another account. During the registration process, do the following:
  - For the type of entity to register as, choose **As Enterprise**.
  - For Telemarketer Name, use **Infobip Private Limited - ALL**. When prompted, start typing **Infobip** and then choose **Infobip Private Limited – ALL** from the dropdown list.
  - For **Enter Telemarketer ID**, enter **110200001152**.
  - When prompted to provide your Header IDs, enter the sender IDs that you want to register.
  - When prompted to provide your Content Templates, enter the message content that you plan to send to your recipients. Include a template for every message that you plan to send.

#### Note

The Vilpower website is not maintained by Amazon Web Services. Steps on their website are subject to change by Vilpower.

## Step 2: Requesting a sender ID

To request a sender ID in India, you need to file an AWS Support request. Complete the steps at [Requesting sender IDs \(p. 185\)](#). In your request, provide the following required information:

- The AWS Region that the sender plans to send SMS messages from.
- The company name used during the DLT registration process.
- The Principal Entity ID (PEID) that you received after successful DLT entity registration.
- Estimated monthly volumes.
- An explanation of your use case.
- A description of the end user opt-in flow.
- Confirmation that end user opt-ins are collected and registered.

## Step 3: Sending SMS messages

After [registering your organization with TRAI \(p. 232\)](#), you can send SMS messages to recipients in India.

1. Sign in to the [Amazon SNS console](#).
2. In the console menu, set the region selector to a [region that supports SMS messaging \(p. 219\)](#).
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Mobile Text messaging (SMS)** page, choose **Publish text message**. The **Publish SMS message** window opens.
5. For **Message type**, choose one of the following:
  - **Promotional** – Noncritical messages, such as marketing messages.  
When using numeric sender IDs, choose this option.
  - **Transactional** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication.  
When using alphabetic or alphanumeric sender IDs, choose this option.

This message-level setting overrides your default message type, which you set on the [Text messaging preferences](#) page.

For pricing information for promotional and transactional messages, see [Global SMS Pricing](#).

6. For **Number**, type the phone number to which you want to send the message.
7. For **Message**, type the message to send.

When adding content to SMS messages, make sure that it exactly matches the content in the DLT registered template. Carriers block SMS messages if their message content includes additional character returns, spaces, punctuation, or mismatched sentence case. Variables in a template can have 30 or fewer characters.

8. In the **Origination identities** section, for the **Sender ID**, type a custom ID that contains 3-11 characters.

Sender IDs can be numeric for promotional messages, or alphabetic or alphanumeric for transactional messages. The sender ID is displayed as the message sender on the receiving device.

9. Expand the **Country-specific attributes** section and specify the following required attributes for sending SMS messages to recipients in India:

- **Entity ID** – The entity ID or principal entity (PE) ID that you received from the regulatory body for sending SMS messages to recipients in India.

This is a custom, TRAI-provided string of 1–50 characters that uniquely identifies the entity that you registered with the TRAI.

- **Template ID** – The template ID that you received from the regulatory body for sending SMS messages to recipients in India.

This is a custom, TRAI-provided string of 1–50 characters that uniquely identifies the template that you registered with the TRAI. The template ID must be associated with the sender ID that you specified in the previous step, and with the message content.

#### 10. Choose **Publish message**.

For information on sending SMS messages to recipients in other countries, see [Publishing to a mobile phone \(p. 196\)](#).

## Troubleshooting SMS messages sent to recipients in India

The following are some reasons carriers may block SMS messages:

- **No template was found that matched the content sent.**

Content sent: **<#> 12345 is your OTP to verify mobile number. Your OTP is valid for 15 minutes -- ABC Pvt. Ltd.**

Matched template: None

Issue: There are no DLT templates that include <#> or {#var#} at the beginning of the DLT registered template.

- **The value of a variable exceeds 30 characters.**

Content sent: **12345 is your OTP code for ABC (ABC Company - India Private Limited) - (ABC 123456789). Share with your agent only. - ABC Pvt. Ltd.**

Matched template: **{#var#} is your OTP code for {#var#} ({#var#}) - ({#var#}) {#var#}. Share with your agent only. - ABC Pvt. Ltd.**

Issue: The value of “ABC Company - India Private Limited” in the content sent exceeds a single {#var#} character limit of 30.

- **The message sentence case does not match the sentence case in the template.**

Content sent: **12345 is your OTP code for ABC (ABC Company - India Private Limited) - (ABC 123456789). Share with your agent only. - ABC Pvt. Ltd.**

Matched template: **{#var#} is your OTP code for {#var#} ({#var#}) - ({#var#}) {#var#}. Share with your agent only. - ABC PVT. LTD.**

Issue: The company name appended to the DLT matched template is capitalized while the content sent has changed parts of the name to lowercase — “ABC Pvt. Ltd.” vs. “ABC PVT. LTD.”

## Mobile push notifications

With [Amazon SNS](#), you have the ability to send push notification messages directly to apps on mobile devices. Push notification messages sent to a mobile endpoint can appear in the mobile app as message alerts, badge updates, or even sound alerts.

### Topics

- [How user notifications work \(p. 235\)](#)
- [User notification process overview \(p. 235\)](#)

- [Setting up a mobile app \(p. 236\)](#)
- [Sending mobile push notifications \(p. 244\)](#)
- [Mobile app attributes \(p. 247\)](#)
- [Mobile app events \(p. 249\)](#)
- [Mobile push API actions \(p. 251\)](#)
- [Mobile push API errors \(p. 252\)](#)
- [Using the Amazon SNS time to live \(TTL\) message attribute for mobile push notifications \(p. 258\)](#)
- [Supported Regions for mobile applications \(p. 260\)](#)

## How user notifications work

You send push notification messages to both mobile devices and desktops using one of the following supported push notification services:

- [Amazon Device Messaging \(ADM\)](#)
- [Apple Push Notification Service \(APNs\) for both iOS and Mac OS X](#)
- [Baidu Cloud Push \(Baidu\)](#)
- [Firebase Cloud Messaging \(FCM\)](#)
- [Microsoft Push Notification Service for Windows Phone \(MPNS\)](#)
- [Windows Push Notification Services \(WNS\)](#)

Push notification services, such as APNs and FCM, maintain a connection with each app and associated mobile device registered to use their service. When an app and mobile device register, the push notification service returns a device token. Amazon SNS uses the device token to create a mobile endpoint, to which it can send direct push notification messages. In order for Amazon SNS to communicate with the different push notification services, you submit your push notification service credentials to Amazon SNS to be used on your behalf. For more information, see [User notification process overview \(p. 235\)](#).

In addition to sending direct push notification messages, you can also use Amazon SNS to send messages to mobile endpoints subscribed to a topic. The concept is the same as subscribing other endpoint types, such as Amazon SQS, HTTP/S, email, and SMS, to a topic, as described in [What is Amazon SNS? \(p. 1\)](#). The difference is that Amazon SNS communicates using the push notification services in order for the subscribed mobile endpoints to receive push notification messages sent to the topic.

## User notification process overview

1. [Obtain the credentials and device token \(p. 236\)](#) for the mobile platforms that you want to support.
2. Use the credentials to create a platform application object (`PlatformApplicationArn`) using Amazon SNS. For more information, see [Creating a platform endpoint \(p. 237\)](#).
3. Use the returned credentials to request a device token for your mobile app and device from the mobile platforms. The token you receive represents your mobile app and device.
4. Use the device token and the `PlatformApplicationArn` to create a platform endpoint object (`EndpointArn`) using Amazon SNS. For more information, see [Creating a platform endpoint \(p. 237\)](#).
5. Use the `EndpointArn` to [publish a message to an app on a mobile device \(p. 236\)](#). For more information, see [Publishing to a mobile device \(p. 244\)](#) and the [Publish API](#) in the Amazon Simple Notification Service API Reference.

## Setting up a mobile app

This section describes how to use the AWS Management Console with the information described in [Prerequisites for Amazon SNS user notifications \(p. 236\)](#) to set up mobile applications.

### Topics

- [Prerequisites for Amazon SNS user notifications \(p. 236\)](#)
- [Creating a platform application \(p. 236\)](#)
- [Creating a platform endpoint \(p. 237\)](#)
- [Adding device tokens or registration IDs \(p. 241\)](#)

## Prerequisites for Amazon SNS user notifications

To begin using Amazon SNS mobile push notifications, you need the following:

- A set of credentials for connecting to one of the supported push notification services: ADM, APNs, Baidu, FCM, MPNS, or WNS.
- A device token or registration ID for the mobile app and device.
- Amazon SNS configured to send push notification messages to the mobile endpoints.
- A mobile app that is registered and configured to use one of the supported push notification services.

Registering your application with a push notification service requires several steps. Amazon SNS needs some of the information you provide to the push notification service in order to send direct push notification messages to the mobile endpoint. Generally speaking, you need the required credentials for connecting to the push notification service, a device token or registration ID (representing your mobile device and mobile app) received from the push notification service, and the mobile app registered with the push notification service.

The exact form the credentials take differs between mobile platforms, but in every case, these credentials must be submitted while making a connection to the platform. One set of credentials is issued for each mobile app, and it must be used to send a message to any instance of that app.

The specific names will vary depending on which push notification service is being used. For example, when using APNs as the push notification service, you need a *device token*. Alternatively, when using FCM, the device token equivalent is called a *registration ID*. The *device token* or *registration ID* is a string that is sent to the application by the operating system of the mobile device. It uniquely identifies an instance of a mobile app running on a particular mobile device and can be thought of as unique identifiers of this app-device pair.

Amazon SNS stores the credentials (plus a few other settings) as a platform application resource. The device tokens (again with some extra settings) are represented as objects called platform endpoints. Each platform endpoint belongs to one specific platform application, and every platform endpoint can be communicated with using the credentials that are stored in its corresponding platform application.

The following sections include the prerequisites for each of the supported push notification services. Once you've obtained the prerequisite information, you can send a push notification message using the AWS Management Console or the Amazon SNS mobile push APIs. For more information, see [User notification process overview \(p. 235\)](#).

## Creating a platform application

For Amazon SNS to send notification messages to mobile endpoints, whether it is direct or with subscriptions to a topic, you first need to create a platform application. After the app is registered with AWS, the next step is to create an endpoint for the app and mobile device. The endpoint is then used by Amazon SNS for sending notification messages to the app and device.

## To create a platform application

1. Sign in to the [Amazon SNS console](#).
2. From the navigation pane on the left, choose **Mobile** and then choose **Push notifications**.
3. From the **Platform applications** section, choose **Create platform application**.

For the list of Regions where you can create mobile applications, see [Supported Regions for mobile applications \(p. 260\)](#).

4. In the **Application name** box, enter a name to represent your app.  
App names must be made up of only uppercase and lowercase ASCII letters, numbers, underscores, hyphens, and periods, and must be between 1 and 256 characters long.
5. In the **Push notification platform** box, choose the platform that the app is registered with and then enter the appropriate credentials.

### Note

If you are using one of the APNs platforms, you can select **Choose file** to upload the .p12 file (exported from Keychain Access) to Amazon SNS.

6. Choose **Create platform application**.

This registers the app with Amazon SNS, which creates a platform application object for the selected platform and then returns a corresponding PlatformApplicationArn.

## Creating a platform endpoint

When an app and mobile device register with a push notification service, the push notification service returns a device token. Amazon SNS uses the device token to create a mobile endpoint, to which it can send direct push notification messages. For more information, see [Prerequisites for Amazon SNS user notifications \(p. 236\)](#) and [User notification process overview \(p. 235\)](#).

This section describes the recommended approach for creating a platform endpoint.

### Topics

- [Create a platform endpoint \(p. 237\)](#)
- [Pseudo code \(p. 238\)](#)
- [AWS SDK example \(p. 238\)](#)
- [Troubleshooting \(p. 241\)](#)

## Create a platform endpoint

To push notifications to an app with Amazon SNS, that app's device token must first be registered with Amazon SNS by calling the create platform endpoint action. This action takes the Amazon Resource Name (ARN) of the platform application and the device token as parameters and returns the ARN of the created platform endpoint.

The create platform endpoint action does the following:

- If the platform endpoint already exists, then do not create it again. Return to the caller the ARN of the existing platform endpoint.
- If the platform endpoint with the same device token but different settings already exists, then do not create it again. Throw an exception to the caller.
- If the platform endpoint does not exist, then create it. Return to the caller the ARN of the newly-created platform endpoint.

You should not call the create platform endpoint action immediately every time an app starts, because this approach does not always provide a working endpoint. This can happen, for example, when an app is uninstalled and reinstalled on the same device and the endpoint for it already exists but is disabled. A successful registration process should accomplish the following:

1. Ensure a platform endpoint exists for this app-device combination.
2. Ensure the device token in the platform endpoint is the latest valid device token.
3. Ensure the platform endpoint is enabled and ready to use.

## Pseudo code

The following pseudo code describes a recommended practice for creating a working, current, enabled platform endpoint in a wide variety of starting conditions. This approach works whether this is a first time the app is being registered or not, whether the platform endpoint for this app already exists, and whether the platform endpoint is enabled, has the correct device token, and so on. It is safe to call it multiple times in a row, as it will not create duplicate platform endpoints or change an existing platform endpoint if it is already up to date and enabled.

```
retrieve the latest device token from the mobile operating system
if (the platform endpoint ARN is not stored)
    # this is a first-time registration
    call create platform endpoint
    store the returned platform endpoint ARN
endif

call get endpoint attributes on the platform endpoint ARN

if (while getting the attributes a not-found exception is thrown)
    # the platform endpoint was deleted
    call create platform endpoint with the latest device token
    store the returned platform endpoint ARN
else
    if (the device token in the endpoint does not match the latest one) or
        (get endpoint attributes shows the endpoint as disabled)
        call set endpoint attributes to set the latest device token and then enable the
        platform endpoint
        endif
    endif
endif
```

This approach can be used any time the app wants to register or re-register itself. It can also be used when notifying Amazon SNS of a device token change. In this case, you can just call the action with the latest device token value. Some points to note about this approach are:

- There are two cases where it may call the create platform endpoint action. It may be called at the very beginning, where the app does not know its own platform endpoint ARN, as happens during a first-time registration. It is also called if the initial get endpoint attributes action call fails with a not-found exception, as would happen if the application knows its endpoint ARN but it was deleted.
- The get endpoint attributes action is called to verify the platform endpoint's state even if the platform endpoint was just created. This happens when the platform endpoint already exists but is disabled. In this case, the create platform endpoint action succeeds but does not enable the platform endpoint, so you must double-check the state of the platform endpoint before returning success.

## AWS SDK example

The following code shows how to implement the previous pseudo code using the Amazon SNS clients that are provided by the AWS SDKs.

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

Java

**SDK for Java 1.x**

```
class RegistrationExample {

    AmazonSNSClient client = new AmazonSNSClient(); //provide credentials here
    String arnStorage = null;

    public void registerWithSNS() {

        String endpointArn = retrieveEndpointArn();
        String token = "Retrieved from the mobile operating system";

        boolean updateNeeded = false;
        boolean createNeeded = (null == endpointArn);

        if (createNeeded) {
            // No platform endpoint ARN is stored; need to call createEndpoint.
            endpointArn = createEndpoint();
            createNeeded = false;
        }

        System.out.println("Retrieving platform endpoint data...");
        // Look up the platform endpoint and make sure the data in it is current, even
        if
        // it was just created.
        try {
            GetEndpointAttributesRequest geaReq =
                new GetEndpointAttributesRequest()
                .withEndpointArn(endpointArn);
            GetEndpointAttributesResult geaRes =
                client.getEndpointAttributes(geaReq);

            updateNeeded = !geaRes.getAttributes().get("Token").equals(token)
                || !geaRes.getAttributes().get("Enabled").equalsIgnoreCase("true");

        } catch (NotFoundException nfe) {
            // We had a stored ARN, but the platform endpoint associated with it
            // disappeared. Recreate it.
            createNeeded = true;
        }

        if (createNeeded) {
            createEndpoint(token);
        }

        System.out.println("updateNeeded = " + updateNeeded);

        if (updateNeeded) {
            // The platform endpoint is out of sync with the current data;
            // update the token and enable it.
            System.out.println("Updating platform endpoint " + endpointArn);
            Map attrs = new HashMap();
            attrs.put("Token", token);
            attrs.put("Enabled", "true");
            SetEndpointAttributesRequest saeReq =
                new SetEndpointAttributesRequest()
                .withEndpointArn(endpointArn)
                .withAttributes(attrs);
            client.setEndpointAttributes(saeReq);
        }
    }
}
```

```
        }

    /**
     * @return never null
     */
    private String createEndpoint(String token) {

        String endpointArn = null;
        try {
            System.out.println("Creating platform endpoint with token " + token);
            CreatePlatformEndpointRequest cpeReq =
                new CreatePlatformEndpointRequest()
                    .withPlatformApplicationArn(applicationArn)
                    .withToken(token);
            CreatePlatformEndpointResult cpeRes = client
                .createPlatformEndpoint(cpeReq);
            endpointArn = cpeRes.getEndpointArn();
        } catch (InvalidParameterException ipe) {
            String message = ipe.getErrorMessage();
            System.out.println("Exception message: " + message);
            Pattern p = Pattern
                .compile(".*Endpoint (arn:aws:sns[^ ]+) already exists " +
                    "with the same [Tt]oken.*");
            Matcher m = p.matcher(message);
            if (m.matches()) {
                // The platform endpoint already exists for this token, but with additional
                // custom data that createEndpoint doesn't want to overwrite. Use the
                // existing platform endpoint.
                endpointArn = m.group(1);
            } else {
                // Rethrow the exception, because the input is actually bad.
                throw ipe;
            }
        }
        storeEndpointArn(endpointArn);
        return endpointArn;
    }

    /**
     * @return the ARN the app was registered under previously, or null if no
     *         platform endpoint ARN is stored.
     */
    private String retrieveEndpointArn() {
        // Retrieve the platform endpoint ARN from permanent storage,
        // or return null if null is stored.
        return arnStorage;
    }

    /**
     * Stores the platform endpoint ARN in permanent storage for lookup next time.
     */
    private void storeEndpointArn(String endpointArn) {
        // Write the platform endpoint ARN to permanent storage.
        arnStorage = endpointArn;
    }
}
```

- Find instructions and more code on [GitHub](#).

For more information, see [Mobile push API actions \(p. 251\)](#).

## Troubleshooting

### Repeatedly calling create platform endpoint with an outdated device token

Especially for FCM endpoints, you may think it is best to store the first device token the application is issued and then call the create platform endpoint with that device token every time on application startup. This may seem correct since it frees the app from having to manage the state of the device token and Amazon SNS will automatically update the device token to its latest value. However, this solution has a number of serious issues:

- Amazon SNS relies on feedback from FCM to update expired device tokens to new device tokens. FCM retains information about old device tokens for some time, but not indefinitely. Once FCM forgets about the connection between the old device token and the new device token, Amazon SNS will no longer be able to update the device token stored in the platform endpoint to its correct value; it will just disable the platform endpoint instead.
- The platform application will contain multiple platform endpoints corresponding to the same device token.
- Amazon SNS imposes a quota on the number of platform endpoints that can be created starting with the same device token. Eventually, the creation of new endpoints will fail with an invalid parameter exception and the following error message: "This endpoint is already registered with a different token."

### Re-enabling a platform endpoint associated with an invalid device token

When a mobile platform (such as APNs or FCM) informs Amazon SNS that the device token used in the publish request was invalid, Amazon SNS disables the platform endpoint associated with that device token. Amazon SNS will then reject subsequent publishes to that device token. While you may think it is best to simply re-enable the platform endpoint and keep publishing, in most situations doing this will not work: the messages that are published do not get delivered and the platform endpoint becomes disabled again soon afterward.

This is because the device token associated with the platform endpoint is genuinely invalid. Deliveries to it cannot succeed because it no longer corresponds to any installed app. The next time it is published to, the mobile platform will again inform Amazon SNS that the device token is invalid, and Amazon SNS will again disable the platform endpoint.

To re-enable a disabled platform endpoint, it needs to be associated with a valid device token (with a set endpoint attributes action call) and then enabled. Only then will deliveries to that platform endpoint become successful. The only time re-enabling a platform endpoint without updating its device token will work is when a device token associated with that endpoint used to be invalid but then became valid again. This can happen, for example, when an app was uninstalled and then re-installed on the same mobile device and receives the same device token. The approach presented above does this, making sure to only re-enable a platform endpoint after verifying that the device token associated with it is the most current one available.

## Adding device tokens or registration IDs

When you first register an app and mobile device with a notification service, such as Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM), device tokens or registration IDs are returned from the notification service. When you add the device tokens or registration IDs to Amazon SNS, they are used with the `PlatformApplicationArn` API to create an endpoint for the app and device. When Amazon SNS creates the endpoint, an `EndpointArn` is returned. The `EndpointArn` is how Amazon SNS knows which app and mobile device to send the notification message to.

You can add device tokens and registration IDs to Amazon SNS using the following methods:

- Manually add a single token to AWS using the AWS Management Console
- Migrate existing tokens from a CSV file to AWS using the AWS Management Console

- Upload several tokens using the `CreatePlatformEndpoint` API
- Register tokens from devices that will install your apps in the future

### To manually add a device token or registration ID

1. Sign in to the [Amazon SNS console](#).
2. Choose **Apps**, choose your app, and then choose **Add Endpoints**.
3. In the **Endpoint Token** box, enter either the token ID or registration ID, depending on which notification service. For example, with ADM and FCM you enter the registration ID.
4. (Optional) In the **User Data** box, enter arbitrary information to associate with the endpoint. Amazon SNS does not use this data. The data must be in UTF-8 format and less than 2KB.
5. Finally, choose **Add Endpoints**.

Now with the endpoint created, you can either send messages directly to a mobile device or send messages to mobile devices that are subscribed to a topic.

### To migrate existing tokens from a CSV file to AWS

You can migrate existing tokens contained in a CSV file. The CSV file cannot be larger than 2MB. When migrating several tokens, it is recommended to use the `CreatePlatformEndpoint` API. Each of the tokens in the CSV file must be followed by a newline. For example, your CSV file should look similar to the following:

```
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMT1mMWxwRkxMaDNST2luZz01,"User data
with spaces requires quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--
KMT1mMWxwRkxMaDNST2luZz04,"Data,with,commas,requires,quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMT1mMWxwRkxMaDNST2luZz02,"Quoted data
requires ""escaped"" quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMT1mMWxwRkxMaDNST2luZz03, {"key":
"json is allowed", "value":"endpoint", "number": 1}
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--
KMT1mMWxwRkxMaDNST2luZz05,SimpleDataNoQuotes
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMT1mMWxwRkxMaDNST2luZz06,"The following
line has no user data"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMT1mMWxwRkxMaDNST2luZz07
APBTKzPG1CyT6E6oOfpdwLpcRNxQp5vCPFiFeru9oZylc22HvZSwQTDgmmw9WdNlXMerUPXmpX0w1,"Different
token style"
```

1. Sign in to the [Amazon SNS console](#).
2. Choose **Apps**, choose your app, and then choose **Add Endpoints**.
3. Choose **Migrate existing tokens over to AWS**, choose **Choose File**, choose your CSV file, and then choose **Add Endpoints**.

### To upload several tokens using the `CreatePlatformEndpoint` API

The following steps show how to use the sample Java app (`bulkupload` package) provided by AWS to upload several tokens (device tokens or registration IDs) to Amazon SNS. You can use this sample app to help you get started with uploading your existing tokens.

#### Note

The following steps use the Eclipse Java IDE. The steps assume you have installed the AWS SDK for Java and you have the AWS security credentials for your AWS account. For more information, see [AWS SDK for Java](#). For more information about credentials, see [How Do I Get Security Credentials?](#) in the [AWS General Reference](#).

1. Download and unzip the [snsmobilepush.zip](#) file.
2. Create a new Java Project in Eclipse.
3. Import the SNSSamples folder to the top-level directory of the newly created Java Project. In Eclipse, right-click choose the name of the Java Project and then choose **Import**, expand **General**, choose **File System**, choose **Next**, browse to the SNSSamples folder, choose **OK**, and then choose **Finish**.
4. Download a copy of the [OpenCSV library](#) and add it to the Build Path of the bulkupload package.
5. Open the BulkUpload.properties file contained in the bulkupload package.
6. Add the following to BulkUpload.properties:
  - The ApplicationArn to which you want to add endpoints.
  - The absolute path for the location of your CSV file containing the tokens.
  - The names for CSV files (such as goodTokens.csv and badTokens.csv) to be created for logging the tokens that Amazon SNS parses correctly and those that fail.
  - (Optional) The characters to specify the delimiter and quote in the CSV file containing the tokens.
  - (Optional) The number of threads to use to concurrently create endpoints. The default is 1 thread.

Your completed BulkUpload.properties should look similar to the following:

```
applicationarn:arn:aws:sns:us-west-2:111122223333:app/FCM/fcmpushapp
csvfilename:C:\\mytokendirectory\\mytokens.csv
goodfilename:C:\\mylogfiles\\goodtokens.csv
badfilename:C:\\mylogfiles\\badtokens.csv
delimiterchar:-
quotearch:-
numofthreads:5
```

7. Run the BatchCreatePlatformEndpointSample.java application to upload the tokens to Amazon SNS.

In this example, the endpoints that were created for the tokens that were uploaded successfully to Amazon SNS would be logged to goodTokens.csv, while the malformed tokens would be logged to badTokens.csv. In addition, you should see STD OUT logs written to the console of Eclipse, containing content similar to the following:

```
<1>[SUCCESS] The endpoint was created with Arn arn:aws:sns:us-west-2:111122223333:app/
FCM/fcmpushapp/165j2214-051z-3176-b586-138o3d420071
<2>[ERROR: MALFORMED CSV FILE] Null token found in /mytokendirectory/mytokens.csv
```

## To register tokens from devices that will install your apps in the future

You can use one of the following two options:

- **Use the Amazon Cognito service:** Your mobile app will need credentials to create endpoints associated with your Amazon SNS platform application. We recommend that you use temporary credentials that expire after a period of time. For most scenarios, we recommend that you use Amazon Cognito to create temporary security credentials. For more information, see the [Amazon Cognito Developer Guide](#). If you would like to be notified when an app registers with Amazon SNS, you can register to receive an Amazon SNS event that will provide the new endpoint ARN. You can also use the ListEndpointByPlatformApplication API to obtain the full list of endpoints registered with Amazon SNS.
- **Use a proxy server:** If your application infrastructure is already set up for your mobile apps to call in and register on each installation, you can continue to use this setup. Your server will act as a proxy and pass the device token to Amazon SNS mobile push notifications, along with any user data you would like to store. For this purpose, the proxy server will connect to Amazon SNS using your AWS

credentials and use the `CreatePlatformEndpoint` API call to upload the token information. The newly created endpoint Amazon Resource Name (ARN) will be returned, which your server can store for making subsequent publish calls to Amazon SNS.

## Sending mobile push notifications

This section describes how to send mobile push notifications.

### Topics

- [Publishing to a topic \(p. 244\)](#)
- [Publishing to a mobile device \(p. 244\)](#)
- [Publishing with platform-specific payload \(p. 244\)](#)

## Publishing to a topic

You can also use Amazon SNS to send messages to mobile endpoints subscribed to a topic. The concept is the same as subscribing other endpoint types, such as Amazon SQS, HTTP/S, email, and SMS, to a topic, as described in [What is Amazon SNS? \(p. 1\)](#). The difference is that Amazon SNS communicates through notification services like Apple Push Notification Service (APNS) and Google Firebase Cloud Messaging (FCM). Through the notifications service, the subscribed mobile endpoints receive notifications sent to the topic.

## Publishing to a mobile device

You can send Amazon SNS push notification messages directly to an endpoint which represents an application on a mobile device.

### To send a direct message

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Mobile, Push notifications**.
3. On the **Mobile push notifications** page, in the the **Platform applications** section, choose the name of the application, for example **MyApp**.
4. On the **MyApp** page, in the **Endpoints** section, choose an endpoint and then choose **Publish message**.
5. On the **Publish message to endpoint** page, enter the message that will appear in the application on the mobile device and then choose **Publish message**.

Amazon SNS sends the notification message to the platform notification service which, in turn, sends the message to the application.

## Publishing with platform-specific payload

You can use the AWS Management Console or Amazon SNS APIs to send custom messages with platform-specific payloads to mobile devices. For information about using the Amazon SNS APIs, see [Mobile push API actions \(p. 251\)](#) and the `SNSMobilePush.java` file in [snsmobilepush.zip](#).

### Sending JSON-formatted messages

When you send platform-specific payloads, the data must be formatted as JSON key-value pair strings, with the quotation marks escaped.

The following examples show a custom message for the FCM platform.

```
{  
    "GCM": "{ \"notification\": { \"body\": \"Sample message for Android endpoints\", \"title\" :\"TitleTest\" } }"  
}
```

## Sending platform-specific messages

In addition to sending custom data as key-value pairs, you can send platform-specific key-value pairs.

The following example shows the inclusion of the FCM parameters `time_to_live` and `collapse_key` after the custom data key-value pairs in the FCM data parameter.

```
{  
    "GCM": {  
        "notification":  
            { \"body\": \"Sample message for Android endpoints\", \"title\" :\"TitleTest\" },  
        "data":  
            {"time_to_live": 3600, "collapse_key": \"deals\"}  
    }  
}
```

For a list of the key-value pairs supported by each of the push notification services supported in Amazon SNS, see the following:

- [Payload Key Reference](#) in the APNs documentation
- [Firebase Cloud Messaging HTTP Protocol](#) in the FCM documentation
- [Send a Message](#) in the ADM documentation

## Sending messages to an application on multiple platforms

To send a message to an application installed on devices for multiple platforms, such as FCM and APNs, you must first subscribe the mobile endpoints to a topic in Amazon SNS and then publish the message to the topic.

The following example shows a message to send to subscribed mobile endpoints on APNs, FCM, and ADM:

```
{  
    "default": "This is the default message which must be present when publishing a message  
    to a topic. The default message will only be used if a message is not present for  
    one of the notification platforms.",  
    "APNS": "{\"aps\":{\"alert\": \"Check out these awesome deals!\",\"url\":\"www.amazon.com  
\\\" }}",  
    "GCM": "{\"data\":{\"message\": \"Check out these awesome deals!\",\"url\":  
\"www.amazon.com\"}}",  
    "ADM": "{\"data\":{\"message\": \"Check out these awesome deals!\",\"url\":  
\"www.amazon.com\"}}"  
}
```

## Sending messages to APNs as alert or background notifications

Amazon SNS can send messages to APNs as alert or background notifications (for more information, see [Pushing Background Updates to Your App](#) in the APNs documentation).

- An alert APNs notification informs the user by displaying an alert message, playing a sound, or adding a badge to your application's icon.

- A background APNs notification wakes up or instructs your application to act upon the content of the notification, without informing the user.

### Specifying custom APNs header values

We recommend specifying custom values for the [AWS.SNS.MOBILE.APNS.PUSH\\_TYPE reserved message attribute \(p. 69\)](#) using the Amazon SNS Publish API action, AWS SDKs, or the AWS CLI. The following CLI example sets content-available to 1 and apns-push-type to background for the specified topic.

```
aws sns publish \
--endpoint-url https://sns.us-east-1.amazonaws.com \
--target-arn arn:aws:sns:us-east-1:123456789012:endpoint/APNS_PLATFORM/MYAPP/1234a567-
bc89-012d-3e45-6fg7h890123i \
--message '{"APNS_PLATFORM":{"aps":{"content-available":1}}}' \
--message-attributes '{ \
    "AWS.SNS.MOBILE.APNS.TOPIC": \
    {"DataType":"String","StringValue":"com.amazon.mobile.messaging.myapp"}, \
    "AWS.SNS.MOBILE.APNS.PRIORITY": {"DataType":"String","StringValue":"10"}}, \
    "AWS.SNS.MOBILE.APNS.PUSH_TYPE": {"DataType":"String","StringValue":"background"} } \
--message-structure json
```

### Inferring the APNs push type header from the payload

If you don't set the apns-push-type APNs header, Amazon SNS sets header to alert or background depending on the content-available key in the aps dictionary of your JSON-formatted APNs payload configuration.

#### Note

Amazon SNS is able to infer only alert or background headers, although the apns-push-type header can be set to other values.

- apns-push-type is set to alert
  - If the aps dictionary contains content-available set to 1 and *one or more keys* that trigger user interactions.
  - If the aps dictionary contains content-available set to 0 or if the content-available key is absent.
  - If the value of the content-available key isn't an integer or a Boolean.
- apns-push-type is set to background
  - If the aps dictionary *only* contains content-available set to 1 and *no other keys* that trigger user interactions.

#### Important

If Amazon SNS sends a raw configuration object for APNs as a background-only notification, you must include content-available set to 1 in the aps dictionary. Although you can include custom keys, the aps dictionary must not contain any keys that trigger user interactions (for example, alerts, badges, or sounds).

The following is an example raw configuration object.

```
{ \
    "APNS": {"aps":{"content-available":1,"Foo1":"\\"Bar\\", "Foo2":123}} \
}
```

In this example, Amazon SNS sets the apns-push-type APNs header for the message to background. When Amazon SNS detects that the apn dictionary contains the content-available key set to 1—and doesn't contain any other keys that can trigger user interactions—it sets the header to background.

## Mobile app attributes

Amazon Simple Notification Service (Amazon SNS) provides support to log the delivery status of push notification messages. After you configure application attributes, log entries will be sent to CloudWatch Logs for messages sent from Amazon SNS to mobile endpoints. Logging message delivery status helps provide better operational insight, such as the following:

- Know whether a push notification message was delivered from Amazon SNS to the push notification service.
- Identify the response sent from the push notification service to Amazon SNS.
- Determine the message dwell time (the time between the publish timestamp and just before handing off to a push notification service).

To configure application attributes for message delivery status, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

### Topics

- [Configuring message delivery status attributes using the AWS Management Console \(p. 247\)](#)
- [Amazon SNS message delivery status CloudWatch log examples \(p. 247\)](#)
- [Configuring message delivery status attributes with the AWS SDKs \(p. 248\)](#)
- [Platform response codes \(p. 249\)](#)

## Configuring message delivery status attributes using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, point to **Mobile**, and then choose **Push notifications**.
3. From the **Platform applications** section, choose the application that contains the endpoints for which you want receive CloudWatch Logs.
4. Choose **Application Actions** and then choose **Delivery Status**.
5. On the **Delivery Status** dialog box, choose **Create IAM Roles**.  
You will then be redirected to the IAM console.
6. Choose **Allow** to give Amazon SNS write access to use CloudWatch Logs on your behalf.
7. Now, back on the **Delivery Status** dialog box, enter a number in the **Percentage of Success to Sample (0-100)** field for the percentage of successful messages sent for which you want to receive CloudWatch Logs.

### Note

After you configure application attributes for message delivery status, all failed message deliveries generate CloudWatch Logs.

8. Finally, choose **Save Configuration**. You will now be able to view and parse the CloudWatch Logs containing the message delivery status. For more information about using CloudWatch, see the [CloudWatch Documentation](#).

## Amazon SNS message delivery status CloudWatch log examples

After you configure message delivery status attributes for an application endpoint, CloudWatch Logs will be generated. Example logs, in JSON format, are shown as follows:

**SUCCESS**

```
{
    "status": "SUCCESS",
    "notification": {
        "timestamp": "2015-01-26 23:07:39.54",
        "messageId": "9655abe4-6ed6-5734-89f7-e6a6a42de02a"
    },
    "delivery": {
        "statusCode": 200,
        "dwellTimeMs": 65,
        "token": "Examplei7fFachkJ1xjlqT64RaBkcGHOchmf1VQAr9k-IBJtKjp7fedYPzEwT_Pq3Tu0lroqro1cwWJJUvgkcPPYcaXcpPwmG3Bqn-wiqIEzp5zZ7y_jsMOPKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw",
        "attempts": 1,
        "providerResponse": "{\"multicast_id\":5138139752481671853,\"success\":1,\"failure\":0,\"canonical_ids\":0,\"results\":[{\"message_id\":\"0:1422313659698010%d6ba8edff9fd7ecd\"}]}",
        "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/FCM/FCMPushApp/c23e42de-3699-84dd-65f84474629d"
    }
}
```

## FAILURE

```
{
    "status": "FAILURE",
    "notification": {
        "timestamp": "2015-01-26 23:29:35.678",
        "messageId": "c3ad79b0-8996-550a-8bfa-24f05989898f"
    },
    "delivery": {
        "statusCode": 8,
        "dwellTimeMs": 1451,
        "token": "example29z6j5c4df46f80189c4c83fjcfgf7f6257e98542d2jt3395kj73",
        "attempts": 1,
        "providerResponse": "NotificationErrorResponse(command=8, status=InvalidToken, id=1, cause=null)",
        "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/APNS_SANDBOX/APNSPushApp/986cb8a1-4f6b-34b1-9a1b-d9e9cb553944"
    }
}
```

For a list of push notification service response codes, see [Platform response codes \(p. 249\)](#).

## Configuring message delivery status attributes with the AWS SDKs

The [AWS SDKs](#) provide APIs in several languages for using message delivery status attributes with Amazon SNS.

The following Java example shows how to use the `SetPlatformApplicationAttributes` API to configure application attributes for message delivery status of push notification messages. You can use the following attributes for message delivery status: `SuccessFeedbackRoleArn`, `FailureFeedbackRoleArn`, and `SuccessFeedbackSampleRate`. The `SuccessFeedbackRoleArn` and `FailureFeedbackRoleArn` attributes are used to give Amazon SNS write access to use CloudWatch Logs on your behalf. The `SuccessFeedbackSampleRate` attribute is for specifying the sample rate percentage (0-100) of successfully delivered messages. After you configure the `FailureFeedbackRoleArn` attribute, then all failed message deliveries generate CloudWatch Logs.

```
SetPlatformApplicationAttributesRequest setPlatformApplicationAttributesRequest = new SetPlatformApplicationAttributesRequest();
```

```
Map<String, String> attributes = new HashMap<>();
attributes.put("SuccessFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("FailureFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("SuccessFeedbackSampleRate", "5");
setPlatformApplicationAttributesRequest.withAttributes(attributes);
setPlatformApplicationAttributesRequest.setPlatformApplicationArn("arn:aws:sns:us-
west-2:111122223333:app/FCM/FCMPushApp");
sns.setPlatformApplicationAttributes(setPlatformApplicationAttributesRequest);
```

For more information about the SDK for Java, see [Getting Started with the AWS SDK for Java](#).

## Platform response codes

The following is a list of links for the push notification service response codes:

Push notification service	Response codes
Amazon Device Messaging (ADM)	See <a href="#">Response Format</a> in the ADM documentation.
Apple Push Notification Service (APNs)	See <a href="#">HTTP/2 Response from APNs</a> in <a href="#">Communicating with APNs</a> in the <i>Local and Remote Notification Programming Guide</i> .
Firebase Cloud Messaging (FCM)	See <a href="#">Downstream Message Error Response Codes</a> in the Firebase Cloud Messaging documentation.
Microsoft Push Notification Service for Windows Phone (MPNS)	See <a href="#">Push Notification Service Response Codes for Windows Phone 8</a> in the Windows 8 Development documentation.
Windows Push Notification Services (WNS)	See "Response codes" in <a href="#">Push Notification Service Request and Response Headers (Windows Runtime Apps)</a> in the Windows 8 Development documentation.

## Mobile app events

Amazon SNS provides support to trigger notifications when certain application events occur. You can then take some programmatic action on that event. Your application must include support for a push notification service such as Apple Push Notification Service (APNs), Firebase Cloud Messaging (FCM), and Windows Push Notification Services (WNS). You set application event notifications using the Amazon SNS console, AWS CLI, or the AWS SDKs.

### Topics

- [Available application events \(p. 249\)](#)
- [Sending mobile push notifications \(p. 250\)](#)

## Available application events

Application event notifications track when individual platform endpoints are created, deleted, and updated, as well as delivery failures. The following are the attribute names for the application events.

Attribute name	Notification trigger
EventEndpointCreated	A new platform endpoint is added to your application.

Attribute name	Notification trigger
EventEndpointDeleted	Any platform endpoint associated with your application is deleted.
EventEndpointUpdated	Any of the attributes of the platform endpoints associated with your application are changed.
EventDeliveryFailure	A delivery to any of the platform endpoints associated with your application encounters a permanent failure.  <b>Note</b> To track delivery failures on the platform application side, subscribe to message delivery status events for the application. For more information, see <a href="#">Using Amazon SNS Application Attributes for Message Delivery Status</a> .

You can associate any attribute with an application which can then receive these event notifications.

## Sending mobile push notifications

To send application event notifications, you specify a topic to receive the notifications for each type of event. As Amazon SNS sends the notifications, the topic can route them to endpoints that will take programmatic action.

### Important

High-volume applications will create a large number of application event notifications (for example, tens of thousands), which will overwhelm endpoints meant for human use, such as email addresses, phone numbers, and mobile applications. Consider the following guidelines when you send application event notifications to a topic:

- Each topic that receives notifications should contain only subscriptions for programmatic endpoints, such as HTTP or HTTPS endpoints, Amazon SQS queues, or AWS Lambda functions.
- To reduce the amount of processing that is triggered by the notifications, limit each topic's subscriptions to a small number (for example, five or fewer).

You can send application event notifications using the Amazon SNS console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

### AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Mobile, Push notifications**.
3. On the **Mobile push notifications** page, in the the **Platform applications** section, choose an application and then choose **Edit**.
4. Expand the **Event notifications** section.
5. Choose **Actions, Configure events**.
6. Enter the ARNs for topics to be used for the following events:
  - Endpoint Created
  - Endpoint Deleted
  - Endpoint Updated
  - Delivery Failure
7. Choose **Save changes**.

## AWS CLI

Run the [set-platform-application-attributes](#) command.

The following example sets the same Amazon SNS topic for all four application events:

```
aws sns set-platform-application-attributes
--platform-application-arn arn:aws:sns:us-east-1:12345EXAMPLE:app/FCM/
MyFCMPlatformApplication
--attributes EventEndpointCreated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointDeleted="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointUpdated="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventDeliveryFailure="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents"
```

## AWS SDKs

Set application event notifications by submitting a `SetPlatformApplicationAttributes` request with the Amazon SNS API using an AWS SDK.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Mobile push API actions

To use the Amazon SNS mobile push APIs, you must first meet the prerequisites for the push notification service, such as Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM). For more information about the prerequisites, see [Prerequisites for Amazon SNS user notifications \(p. 236\)](#).

To send a push notification message to a mobile app and device using the APIs, you must first use the `CreatePlatformApplication` action, which returns a `PlatformApplicationArn` attribute. The `PlatformApplicationArn` attribute is then used by `CreatePlatformEndpoint`, which returns an `EndpointArn` attribute. You can then use the `EndpointArn` attribute with the `Publish` action to send a notification message to a mobile app and device, or you could use the `EndpointArn` attribute with the `Subscribe` action for subscription to a topic. For more information, see [User notification process overview \(p. 235\)](#).

The Amazon SNS mobile push APIs are as follows:

### [CreatePlatformApplication](#)

Creates a platform application object for one of the supported push notification services, such as APNs and FCM, to which devices and mobile apps may register. Returns a `PlatformApplicationArn` attribute, which is used by the `CreatePlatformEndpoint` action.

### [CreatePlatformEndpoint](#)

Creates an endpoint for a device and mobile app on one of the supported push notification services. `CreatePlatformEndpoint` uses the `PlatformApplicationArn` attribute returned from the `CreatePlatformApplication` action. The `EndpointArn` attribute, which is returned when using `CreatePlatformEndpoint`, is then used with the `Publish` action to send a notification message to a mobile app and device.

### [CreateTopic](#)

Creates a topic to which messages can be published.

### [DeleteEndpoint](#)

Deletes the endpoint for a device and mobile app on one of the supported push notification services.

[DeletePlatformApplication](#)

Deletes a platform application object.

[DeleteTopic](#)

Deletes a topic and all its subscriptions.

[GetEndpointAttributes](#)

Retrieves the endpoint attributes for a device and mobile app.

[GetPlatformApplicationAttributes](#)

Retrieves the attributes of the platform application object.

[ListEndpointsByPlatformApplication](#)

Lists the endpoints and endpoint attributes for devices and mobile apps in a supported push notification service.

[ListPlatformApplications](#)

Lists the platform application objects for the supported push notification services.

[Publish](#)

Sends a notification message to all of a topic's subscribed endpoints.

[SetEndpointAttributes](#)

Sets the attributes for an endpoint for a device and mobile app.

[SetPlatformApplicationAttributes](#)

Sets the attributes of the platform application object.

[Subscribe](#)

Prepares to subscribe an endpoint by sending the endpoint a confirmation message. To actually create a subscription, the endpoint owner must call the ConfirmSubscription action with the token from the confirmation message.

[Unsubscribe](#)

Deletes a subscription.

## Mobile push API errors

Errors that are returned by the Amazon SNS APIs for mobile push are listed in the following table. For more information about the Amazon SNS APIs for mobile push, see [Mobile push API actions \(p. 251\)](#).

Error	Description	HTTPS status code	API Action
Application Name is null string	The required application name is set to null.	400	<a href="#">CreatePlatformApplication</a>
Platform Name is null string	The required platform name is set to null.	400	<a href="#">CreatePlatformApplication</a>
Platform Name is invalid	An invalid or out-of-range value was supplied for the platform name.	400	<a href="#">CreatePlatformApplication</a>

Error	Description	HTTPS status code	API Action
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is not provided	The APNs principal, which is the SSL certificate, was not provided.	400	CreatePlatformApplication
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	CreatePlatformApplication
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication

Error	Description	HTTPS status code	API Action
FCM — serverAPIKey is empty	The FCM credential, which is the API key, is empty.	400	CreatePlatformApplication
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	CreatePlatformApplication
FCM — serverAPIKey is invalid	The FCM credential, which is the API key, is invalid.	400	CreatePlatformApplication
ADM — clientsecret is not provided	The required client secret is not provided.	400	CreatePlatformApplication
ADM — clientsecret is a null string	The required string for the client secret is null.	400	CreatePlatformApplication
ADM — client_secret is empty string	The required string for the client secret is empty.	400	CreatePlatformApplication
ADM — client_secret is not valid	The required string for the client secret is not valid.	400	CreatePlatformApplication
ADM — client_id is empty string	The required string for the client ID is empty.	400	CreatePlatformApplication
ADM — clientId is not provided	The required string for the client ID is not provided.	400	CreatePlatformApplication
ADM — clientid is a null string	The required string for the client ID is null.	400	CreatePlatformApplication
ADM — client_id is not valid	The required string for the client ID is not valid.	400	CreatePlatformApplication
EventEndpointCreated has invalid ARN format	EventEndpointCreated has invalid ARN format.	400	CreatePlatformApplication
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	CreatePlatformApplication
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	CreatePlatformApplication
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	CreatePlatformApplication

Error	Description	HTTPS status code	API Action
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	CreatePlatformApplication
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	CreatePlatformApplication
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure is not an existing topic.	400	CreatePlatformApplication
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	CreatePlatformApplication
Platform ARN is invalid	Platform ARN is invalid.	400	SetPlatformAttributes
Platform ARN is valid but does not belong to the user	Platform ARN is valid but does not belong to the user.	400	SetPlatformAttributes
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	SetPlatformAttributes
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes
APNs — Principal is not provided	The APNs principal, which is the SSL certificate, was not provided.	400	SetPlatformAttributes

Error	Description	HTTPS status code	API Action
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	<code>SetPlatformAttributes</code>
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	<code>SetPlatformAttributes</code>
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see <a href="#">CreatePlatformApplication</a> in the Amazon Simple Notification Service API Reference.	400	<code>SetPlatformAttributes</code>
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	<code>SetPlatformAttributes</code>
ADM — clientId is not provided	The required string for the client ID is not provided.	400	<code>SetPlatformAttributes</code>
ADM — clientid is a null string	The required string for the client ID is null.	400	<code>SetPlatformAttributes</code>
ADM — clientsecret is not provided	The required client secret is not provided.	400	<code>SetPlatformAttributes</code>
ADM — clientsecret is a null string	The required string for the client secret is null.	400	<code>SetPlatformAttributes</code>
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	<code>SetPlatformAttributes</code>
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	<code>SetPlatformAttributes</code>
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	<code>SetPlatformAttributes</code>
EventDeliveryAttemptFailed has invalid ARN format	EventDeliveryAttemptFailed has invalid ARN format.	400	<code>SetPlatformAttributes</code>
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	<code>SetPlatformAttributes</code>
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	<code>SetPlatformAttributes</code>

Error	Description	HTTPS status code	API Action
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	SetPlatformAttributes
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	SetPlatformAttributes
EventDeliveryAttemptFailed is not an existing Topic	EventDeliveryAttemptFailed is not an existing topic.	400	SetPlatformAttributes
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	SetPlatformAttributes
Platform ARN is invalid	The platform ARN is invalid.	400	GetPlatformApplicationAttributes
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	GetPlatformApplicationAttributes
Token specified is invalid	The specified token is invalid.	400	ListPlatformApplications
Platform ARN is invalid	The platform ARN is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	ListEndpointsByPlatformApplication
Token specified is invalid	The specified token is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	DeletePlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	DeletePlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	CreatePlatformEndpoint
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	CreatePlatformEndpoint
Token is not specified	The token is not specified.	400	CreatePlatformEndpoint
Token is not of correct length	The token is not the correct length.	400	CreatePlatformEndpoint
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	CreatePlatformEndpoint

Error	Description	HTTPS status code	API Action
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	DeleteEndpoint
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	DeleteEndpoint
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	SetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	SetEndpointAttributes
Token is not specified	The token is not specified.	400	SetEndpointAttributes
Token is not of correct length	The token is not the correct length.	400	SetEndpointAttributes
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	SetEndpointAttributes
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	GetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	GetEndpointAttributes
Target ARN is invalid	The target ARN is invalid.	400	Publish
Target ARN is valid but does not belong to the user	The target ARN is valid, but does not belong to the user.	403	Publish
Message format is invalid	The message format is invalid.	400	Publish
Message size is larger than supported by protocol/end-service	The message size is larger than supported by the protocol/end-service.	400	Publish

## Using the Amazon SNS time to live (TTL) message attribute for mobile push notifications

Amazon Simple Notification Service (Amazon SNS) provides support for setting a *Time To Live (TTL)* message attribute for mobile push notifications messages. This is in addition to the existing capability of setting TTL within the Amazon SNS message body for the mobile push notification services that support this, such as Amazon Device Messaging (ADM) and Firebase Cloud Messaging (FCM).

The TTL message attribute is used to specify expiration metadata about a message. This allows you to specify the amount of time that the push notification service, such as Apple Push Notification Service (APNs) or FCM, has to deliver the message to the endpoint. If for some reason (such as the mobile device has been turned off) the message is not deliverable within the specified TTL, then the message will be dropped and no further attempts to deliver it will be made. To specify TTL within message attributes, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

#### Topics

- [TTL message attributes for push notification services \(p. 259\)](#)
- [Precedence order for determining TTL \(p. 259\)](#)
- [Specifying TTL using the AWS Management Console \(p. 260\)](#)

## TTL message attributes for push notification services

The following is a list of the TTL message attributes for push notification services that you can use to set when using the AWS SDKs or query API:

Push notification service	TTL message attribute
Amazon Device Messaging (ADM)	AWS.SNS.MOBILE.ADM.TTL
Apple Push Notification Service (APNs)	AWS.SNS.MOBILE.APNS.TTL
Apple Push Notification Service Sandbox (APNs_SANDBOX)	AWS.SNS.MOBILE.APNS_SANDBOX.TTL
Baidu Cloud Push (Baidu)	AWS.SNS.MOBILE.BAIDU.TTL
Firebase Cloud Messaging (FCM)	AWS.SNS.MOBILE.FCM.TTL
Windows Push Notification Services (WNS)	AWS.SNS.MOBILE.WNS.TTL

Each of the push notification services handle TTL differently. Amazon SNS provides an abstract view of TTL over all the push notification services, which makes it easier to specify TTL. When you use the AWS Management Console to specify TTL (in seconds), you only have to enter the TTL value once and Amazon SNS will then calculate the TTL for each of the selected push notification services when publishing the message.

TTL is relative to the publish time. Before handing off a push notification message to a specific push notification service, Amazon SNS computes the dwell time (the time between the publish timestamp and just before handing off to a push notification service) for the push notification and passes the remaining TTL to the specific push notification service. If TTL is shorter than the dwell time, Amazon SNS won't attempt to publish.

If you specify a TTL for a push notification message, then the TTL value must be a positive integer, unless the value of 0 has a specific meaning for the push notification service—such as with APNs and FCM. If the TTL value is set to 0 and the push notification service does not have a specific meaning for 0, then Amazon SNS will drop the message. For more information about the TTL parameter set to 0 when using APNs, see *Table A-3 Item identifiers for remote notifications* in the [Binary Provider API](#) documentation.

## Precedence order for determining TTL

The precedence that Amazon SNS uses to determine the TTL for a push notification message is based on the following order, where the lowest number has the highest priority:

1. Message attribute TTL
2. Message body TTL
3. Push notification service default TTL (varies per service)
4. Amazon SNS default TTL (4 weeks)

If you set different TTL values (one in message attributes and another in the message body) for the same message, then Amazon SNS will modify the TTL in the message body to match the TTL specified in the message attribute.

## Specifying TTL using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Mobile, Push notifications**.
3. On the **Mobile push notifications** page, in the **Platform applications** section, choose an application.
4. On the [\*\*MyApplication\*\*](#) page, in the **Endpoints** section, choose an application endpoint and then choose **Publish message**.
5. In the **Message details** section, enter the TTL (the number of seconds that the push notification service has to deliver the message to the endpoint).
6. Choose **Publish message**.

## Supported Regions for mobile applications

Currently, you can create mobile applications in the following Regions:

- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- South America (São Paulo)

## Email notifications

This page describes how to subscribe an [email address \(p. 260\)](#) to an Amazon SNS topic using the AWS Management Console, AWS SDK for Java, or AWS SDK for .NET.

### Notes

- You can't customize the body of the email message. The email delivery feature is intended to provide internal system alerts, not marketing messages.
- Email delivery throughput is throttled according to [Amazon SNS quotas](#).

## To subscribe an email address to an Amazon SNS topic using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:
  - a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a topic.
  - b. For **Protocol**, choose **Email**.
  - c. For **Endpoint**, enter the email address.
  - d. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see [Amazon SNS subscription filter policies \(p. 71\)](#).
  - e. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see [Amazon SNS dead-letter queues \(DLQs\) \(p. 96\)](#).
  - f. Choose **Create subscription**.

The console creates the subscription and opens the subscription's **Details** page.

You must confirm the subscription before the email address can start to receive messages.

### To confirm a subscription

1. Check your email inbox and choose **Confirm subscription** in the email from Amazon SNS.
2. Amazon SNS opens your web browser and displays a subscription confirmation with your subscription ID.

## To subscribe an email address to an Amazon SNS topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see [The shared config and credentials files](#) in the *AWS SDKs and Tools Reference Guide*.

The following code examples show how to subscribe an email address to an Amazon SNS topic.

C++

### SDK for C++

```
/**  
 * Subscribe an email address endpoint to a topic - demonstrates how to initiate a  
 * subscription to an Amazon SNS topic with delivery  
 * to an email address.  
 *  
 * SNS will send a subscription confirmation email to the email address provided  
 * which you need to confirm to  
 * receive messages.  
 *  
 * <protocol_value> set to "email" provides delivery of message via SMTP (see  
 https://docs.aws.amazon.com/sns/latest/api/API_Subscribe.html for available  
 protocols).
```

```
* <topic_arn_value> can be obtained from run_list_topics executable and includes
the "arn:" prefix.
*/

int main(int argc, char ** argv)
{
    if (argc != 4)
    {
        std::cout << "Usage: subscribe_email <protocol_value=email> <topic_arn_value>" 
                  " <email_address>" << std::endl;
        return 1;
    }

    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        Aws::SNS::SNSClient sns;
        Aws::String protocol = argv[1];
        Aws::String topic_arn = argv[2];
        Aws::String endpoint = argv[3];

        Aws::SNS::Model::SubscribeRequest s_req;
        s_req.SetTopicArn(topic_arn);
        s_req.SetProtocol(protocol);
        s_req.SetEndpoint(endpoint);

        auto s_out = sns.Subscribe(s_req);

        if (s_out.IsSuccess())
        {
            std::cout << "Subscribed successfully " << std::endl;
        }
        else
        {
            std::cout << "Error while subscribing " << s_out.GetError().GetMessage()
                  << std::endl;
        }
    }

    Aws::ShutdownAPI(options);
    return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for C++ API Reference*.

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Go API Reference*.

## Java

### SDK for Java 2.x

```
public static void subEmail(SnsClient snsClient, String topicArn, String email)
{
```

```
try {
    SubscribeRequest request = SubscribeRequest.builder()
        .protocol("email")
        .endpoint(email)
        .returnSubscriptionArn(true)
        .topicArn(topicArn)
        .build();

    SubscribeResponse result = snsClient.subscribe(request);
    System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is " + result.sdkHttpResponse().statusCode());

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SubscribeCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
  Protocol: "email" /* required */,
  TopicArn: "TOPIC_ARN", //TOPIC_ARN
  Endpoint: "EMAIL_ADDRESS", //EMAIL_ADDRESS
};

const run = async () => {
  try {
    const data = await snsClient.send(new SubscribeCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Subscribe](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 * message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:11112223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in [AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
```

```
"""
    self.sns_resource = sns_resource

def subscribe(topic, protocol, endpoint):
    """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
    subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.

    :param topic: The topic to subscribe to.
    :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
    :param endpoint: The endpoint that receives messages, such as a phone
        number
            (in E.164 format) for SMS messages, or an email address
    for
            email messages.
    :return: The newly added subscription.
    """
    try:
        subscription = topic.subscribe(
            Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True)
        logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
topic.arn)
    except ClientError:
        logger.exception(
            "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
topic.arn)
        raise
    else:
        return subscription
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Python (Boto3) API Reference*.

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def subscription_created?(sns_client, topic_arn, protocol, endpoint)

    sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
endpoint)

rescue StandardError => e
    puts "Error while creating the subscription: #{e.message}"
end

# Full example call:
def run_me

protocol = 'email'
endpoint = 'EMAIL_ADDRESS'
topic_arn = 'TOPIC_ARN'
region = 'REGION'

sns_client = Aws::SNS::Client.new(region: region)
```

```
puts "Creating the subscription."  
  
if subscription_created?(sns_client, topic_arn, protocol, endpoint)  
  puts 'The subscription was created.'  
else  
  puts 'The subscription was not created. Stopping program.'  
  exit 1  
end  
  
run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [Subscribe](#) in [AWS SDK for Ruby API Reference](#).

## Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn subscribe_and_publish(  
    client: &Client,  
    topic_arn: &str,  
    email_address: &str,  
) -> Result<(), Error> {  
    println!("Receiving on topic with ARN: `{}`, topic_arn);  
  
    let rsp = client  
        .subscribe()  
        .topic_arn(topic_arn)  
        .protocol("email")  
        .endpoint(email_address)  
        .send()  
        .await?;  
  
    println!("Added a subscription: {:?}", rsp);  
  
    let rsp = client  
        .publish()  
        .topic_arn(topic_arn)  
        .message("hello sns!")  
        .send()  
        .await?;  
  
    println!("Published message: {:?}", rsp);  
    Ok(())  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in [AWS SDK for Rust API reference](#).

# Code examples for Amazon SNS

The following code examples show how to use Amazon SNS with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Code examples

- [Cross-service examples for Amazon SNS \(p. 268\)](#)
  - [Build an application to submit data to a DynamoDB table \(p. 268\)](#)
  - [Build a publish and subscription web application that translates messages \(p. 269\)](#)
  - [Create an Amazon Textract explorer application \(p. 269\)](#)
  - [Detect people and objects in a video with Amazon Rekognition using an AWS SDK \(p. 270\)](#)
  - [Use API Gateway to invoke a Lambda function \(p. 271\)](#)
  - [Use scheduled events to invoke a Lambda function \(p. 272\)](#)
- [Scenario examples for Amazon SNS \(p. 272\)](#)
  - [Create a platform endpoint for Amazon SNS push notifications using an AWS SDK \(p. 273\)](#)
  - [Create and publish to a FIFO Amazon SNS topic using an AWS SDK \(p. 275\)](#)
  - [Publish SMS messages to an Amazon SNS topic using an AWS SDK \(p. 277\)](#)
  - [Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK \(p. 278\)](#)
- [API examples for Amazon SNS \(p. 280\)](#)
  - [Add tags to an Amazon SNS topic using an AWS SDK \(p. 281\)](#)
  - [Check whether a phone number is opted out of Amazon SNS using an AWS SDK \(p. 281\)](#)
  - [Confirm an endpoint owner wants to receive Amazon SNS messages using an AWS SDK \(p. 284\)](#)
  - [Create an Amazon SNS topic using an AWS SDK \(p. 286\)](#)
  - [Delete an Amazon SNS subscription using an AWS SDK \(p. 291\)](#)
  - [Delete an Amazon SNS topic using an AWS SDK \(p. 294\)](#)
  - [Get the properties of an Amazon SNS topic using an AWS SDK \(p. 297\)](#)
  - [Get the settings for sending Amazon SNS SMS messages using an AWS SDK \(p. 301\)](#)
  - [List phone numbers opted out of Amazon SNS using an AWS SDK \(p. 304\)](#)
  - [List the subscribers of an Amazon SNS topic using an AWS SDK \(p. 305\)](#)
  - [List Amazon SNS topics using an AWS SDK \(p. 310\)](#)
  - [Publish an Amazon SNS SMS text message using an AWS SDK \(p. 315\)](#)
  - [Publish to an Amazon SNS topic using an AWS SDK \(p. 318\)](#)
  - [Set a dead-letter queue for an Amazon SNS subscription using an AWS SDK \(p. 324\)](#)
  - [Set an Amazon SNS filter policy using an AWS SDK \(p. 324\)](#)
  - [Set the default settings for sending Amazon SNS SMS messages using an AWS SDK \(p. 326\)](#)
  - [Set Amazon SNS topic attributes using an AWS SDK \(p. 328\)](#)
  - [Subscribe a Lambda function to receive notifications from an Amazon SNS topic using an AWS SDK \(p. 331\)](#)
  - [Subscribe a mobile application to an Amazon SNS topic using an AWS SDK \(p. 334\)](#)
  - [Subscribe an HTTP endpoint to an Amazon SNS topic using an AWS SDK \(p. 336\)](#)

- [Subscribe an email address to an Amazon SNS topic using an AWS SDK \(p. 337\)](#)

## Cross-service examples for Amazon SNS

The following code examples show how to use Amazon SNS with AWS SDKs.

### Examples

- [Build an application to submit data to a DynamoDB table \(p. 268\)](#)
- [Build a publish and subscription web application that translates messages \(p. 269\)](#)
- [Create an Amazon Textract explorer application \(p. 269\)](#)
- [Detect people and objects in a video with Amazon Rekognition using an AWS SDK \(p. 270\)](#)
- [Use API Gateway to invoke a Lambda function \(p. 271\)](#)
- [Use scheduled events to invoke a Lambda function \(p. 272\)](#)

## Build an application to submit data to a DynamoDB table

The following code examples show how to build an application that submits data to an Amazon DynamoDB table and notifies you when a user updates the table.

Java

### SDK for Java 2.x

Shows how to create a dynamic web application that submits data using the Amazon DynamoDB Java API and sends a text message using the Amazon Simple Notification Service Java API.

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 SNS

JavaScript

### SDK for JavaScript V3

This example shows how to build an app that enables users to submit data to an Amazon DynamoDB table, and send a text message to the administrator using Amazon Simple Notification Service (Amazon SNS).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](#).

### Services used in this example

- DynamoDB
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Build a publish and subscription web application that translates messages

The following code example shows how to create a web application that has subscription and publish functionality and translates messages.

Java

### SDK for Java 2.x

Shows how to use the Amazon Simple Notification Service Java API to create a web application that has subscription and publish functionality. In addition, this example application also translates messages.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

For complete source code and instructions on how to set up and run the example that uses the Java Async API, see the full example on [GitHub](#).

### Services used in this example

- Amazon SNS
- Amazon Translate

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

### SDK for JavaScript V3

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

### Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS

- Amazon SQS
- Amazon Textract

Python

#### **SDK for Python (Boto3)**

Shows how to use the AWS SDK for Python (Boto3) with Amazon Textract to detect text, form, and table elements in a document image. The input image and Amazon Textract output are shown in a Tkinter application that lets you explore the detected elements.

- Submit a document image to Amazon Textract and explore the output of detected elements.
- Submit images directly to Amazon Textract or through an Amazon Simple Storage Service (Amazon S3) bucket.
- Use asynchronous APIs to start a job that publishes a notification to an Amazon Simple Notification Service (Amazon SNS) topic when the job completes.
- Poll an Amazon Simple Queue Service (Amazon SQS) queue for a job completion message and display the results.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

#### **Services used in this example**

- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## 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.

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 SNS

- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Use API Gateway to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by Amazon API Gateway that sends anniversary text messages.

Java

### SDK for Java 2.x

Shows how to create an AWS Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create a Lambda function invoked by Amazon API Gateway that scans an Amazon DynamoDB table for work anniversaries and uses Amazon Simple Notification Service (Amazon SNS) to send a text message to your employees that congratulates them at their one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

### Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

### SDK for JavaScript V3

Shows how to create an AWS Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create a Lambda function invoked by Amazon API Gateway that scans an Amazon DynamoDB table for work anniversaries and uses Amazon Simple Notification Service (Amazon SNS) to send a text message to your employees that congratulates them at their one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](#).

### Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Use scheduled events to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by an Amazon EventBridge scheduled event that sends anniversary text messages.

Java

### SDK for Java 2.x

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

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
- EventBridge
- Lambda
- Amazon SNS

JavaScript

### SDK for JavaScript V3

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](#).

### Services used in this example

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Scenario examples for Amazon SNS

The following code examples show how to use Amazon SNS with AWS SDKs.

## Examples

- [Create a platform endpoint for Amazon SNS push notifications using an AWS SDK \(p. 273\)](#)
- [Create and publish to a FIFO Amazon SNS topic using an AWS SDK \(p. 275\)](#)
- [Publish SMS messages to an Amazon SNS topic using an AWS SDK \(p. 277\)](#)
- [Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK \(p. 278\)](#)

# Create a platform endpoint for Amazon SNS push notifications using an AWS SDK

The following code example shows how to create a platform endpoint for Amazon SNS push notifications.

Java

### SDK for Java 1.x

```
class RegistrationExample {

    AmazonSNSClient client = new AmazonSNSClient(); //provide credentials here
    String arnStorage = null;

    public void registerWithSNS() {

        String endpointArn = retrieveEndpointArn();
        String token = "Retrieved from the mobile operating system";

        boolean updateNeeded = false;
        boolean createNeeded = (null == endpointArn);

        if (createNeeded) {
            // No platform endpoint ARN is stored; need to call createEndpoint.
            endpointArn = createEndpoint();
            createNeeded = false;
        }

        System.out.println("Retrieving platform endpoint data...");
        // Look up the platform endpoint and make sure the data in it is current, even
        if
            // it was just created.
            try {
                GetEndpointAttributesRequest geaReq =
                    new GetEndpointAttributesRequest()
                    .withEndpointArn(endpointArn);
                GetEndpointAttributesResult geaRes =
                    client.getEndpointAttributes(geaReq);

                updateNeeded = !geaRes.getAttributes().get("Token").equals(token)
                    || !geaRes.getAttributes().get("Enabled").equalsIgnoreCase("true");

            } catch (NotFoundException nfe) {
                // We had a stored ARN, but the platform endpoint associated with it
                // disappeared. Recreate it.
                createNeeded = true;
            }
        }

        if (createNeeded) {
            createEndpoint(token);
        }
    }
}
```

```

System.out.println("updateNeeded = " + updateNeeded);

if (updateNeeded) {
    // The platform endpoint is out of sync with the current data;
    // update the token and enable it.
    System.out.println("Updating platform endpoint " + endpointArn);
    Map attrs = new HashMap();
    attrs.put("Token", token);
    attrs.put("Enabled", "true");
    SetEndpointAttributesRequest saeReq =
        new SetEndpointAttributesRequest()
            .withEndpointArn(endpointArn)
            .withAttributes(attrs);
    client.setEndpointAttributes(saeReq);
}

/***
 * @return never null
 */
private String createEndpoint(String token) {

    String endpointArn = null;
    try {
        System.out.println("Creating platform endpoint with token " + token);
        CreatePlatformEndpointRequest cpeReq =
            new CreatePlatformEndpointRequest()
                .withPlatformApplicationArn(applicationArn)
                .withToken(token);
        CreatePlatformEndpointResult cpeRes = client
            .createPlatformEndpoint(cpeReq);
        endpointArn = cpeRes.getEndpointArn();
    } catch (InvalidParameterException ipe) {
        String message = ipe.getErrorMessage();
        System.out.println("Exception message: " + message);
        Pattern p = Pattern
            .compile(".*Endpoint (arn:aws:sns[^ ]+) already exists " +
                "with the same [Tt]oken.*");
        Matcher m = p.matcher(message);
        if (m.matches()) {
            // The platform endpoint already exists for this token, but with additional
            // custom data that createEndpoint doesn't want to overwrite. Use the
            // existing platform endpoint.
            endpointArn = m.group(1);
        } else {
            // Rethrow the exception, because the input is actually bad.
            throw ipe;
        }
    }
    storeEndpointArn(endpointArn);
    return endpointArn;
}

/***
 * @return the ARN the app was registered under previously, or null if no
 *         platform endpoint ARN is stored.
 */
private String retrieveEndpointArn() {
    // Retrieve the platform endpoint ARN from permanent storage,
    // or return null if null is stored.
    return arnStorage;
}

/***
 * Stores the platform endpoint ARN in permanent storage for lookup next time.
*/

```

```
* */
private void storeEndpointArn(String endpointArn) {
    // Write the platform endpoint ARN to permanent storage.
    arnStorage = endpointArn;
}
```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Create and publish to a FIFO Amazon SNS topic using an AWS SDK

The following code example shows how to create and publish to a FIFO Amazon SNS topic.

Java

### SDK for Java 1.x

Create a FIFO topic and FIFO queues. Subscribe the queues to the topic.

```
// Create API clients

AWSCredentialsProvider credentials = getCredentials();

AmazonSNS sns = new AmazonSNSClient(credentials);
AmazonSQS sqs = new AmazonSQSClient(credentials);

// Create FIFO topic

Map<String, String> topicAttributes = new HashMap<String, String>();

topicAttributes.put("FifoTopic", "true");
topicAttributes.put("ContentBasedDeduplication", "false");

String topicArn = sns.createTopic(
    new CreateTopicRequest()
        .withName("PriceUpdatesTopic.fifo")
        .withAttributes(topicAttributes)
).getTopicArn();

// Create FIFO queues

Map<String, String> queueAttributes = new HashMap<String, String>();

queueAttributes.put("FifoQueue", "true");

// Disable content-based deduplication because messages published with the same
// body
// might carry different attributes that must be processed independently.
// The price management system uses the message attributes to define whether a
// given
// price update applies to the wholesale application or to the retail application.
queueAttributes.put("ContentBasedDeduplication", "false");

String wholesaleQueueUrl = sqs.createQueue(
    new CreateQueueRequest()
        .withName("WholesaleQueue.fifo")
```

```

        .withAttributes(queueAttributes)
    ).getQueueUrl();

String retailQueueUrl = sqs.createQueue(
    new CreateQueueRequest()
        .withName("RetailQueue fifo")
        .withAttributes(queueAttributes)
).getQueueUrl();

// Subscribe FIFO queues to FIFO topic, setting required permissions

String wholesaleSubscriptionArn =
    Topics.subscribeQueue(sns, sqs, topicArn, wholesaleQueueUrl);

String retailSubscriptionArn =
    Topics.subscribeQueue(sns, sqs, topicArn, retailQueueUrl);

```

Set a filter policy on each subscription so that each subscriber application receives only the price updates that it needs.

```

// Set the Amazon SNS subscription filter policies

SNSMessageFilterPolicy wholesalePolicy = new SNSMessageFilterPolicy();
wholesalePolicy.addAttribute("business", "wholesale");
wholesalePolicy.apply(sns, wholesaleSubscriptionArn);

SNSMessageFilterPolicy retailPolicy = new SNSMessageFilterPolicy();
retailPolicy.addAttribute("business", "retail");
retailPolicy.apply(sns, retailSubscriptionArn);

```

Compose and publish a message that updates the wholesale price.

```

// Publish message to FIFO topic

String subject = "Price Update";
String payload = "{\"product\": 214, \"price\": 79.99}";
String groupId = "PID-214";
String dedupId = UUID.randomUUID().toString();
String attributeName = "business";
String attributeValue = "wholesale";

Map<String, MessageAttributeValue> attributes = new HashMap<>();

attributes.put(
    attributeName,
    new MessageAttributeValue()
        .withDataType("String")
        .withStringValue(attributeValue));

sns.publish(
    new PublishRequest()
        .withTopicArn(topicArn)
        .withSubject(subject)
        .withMessage(payload)
        .withMessageGroupId(groupId)
        .withMessageDuplicationId(dedupId)
        .withMessageAttributes(attributes));

```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Publish SMS messages to an Amazon SNS topic using an AWS SDK

The following code example shows how to:

- Create an Amazon SNS topic.
- Subscribe phone numbers to the topic.
- Publish SMS messages to the topic so that all subscribed phone numbers receive the message at once.

Java

### SDK for Java 1.x

Create a topic and return its ARN.

```
public static String createSNSTopic(AmazonSNSClient snsClient) {
    CreateTopicRequest createTopic = new CreateTopicRequest("mySNSTopic");
    CreateTopicResult result = snsClient.createTopic(createTopic);
    System.out.println("Create topic request: " +
        snsClient.getCachedResponseMetadata(createTopic));
    System.out.println("Create topic result: " + result);
    return result.getTopicArn();
}
```

Subscribe an endpoint to a topic.

```
public static void subscribeToTopic(AmazonSNSClient snsClient, String topicArn,
    String protocol, String endpoint) {
    SubscribeRequest subscribe = new SubscribeRequest(topicArn, protocol,
        endpoint);
    SubscribeResult subscribeResult = snsClient.subscribe(subscribe);
    System.out.println("Subscribe request: " +
        snsClient.getCachedResponseMetadata(subscribe));
    System.out.println("Subscribe result: " + subscribeResult);
}
```

Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```
public static void addMessageAttributes(Map<String, MessageAttributeValue>
    smsAttributes) {
    smsAttributes.put("AWS.SNS.SMS.SenderID", new MessageAttributeValue()
        .withStringValue("mySenderId") //The sender ID shown on the device.
        .withDataType("String"));
    smsAttributes.put("AWS.SNS.SMS.MaxPrice", new MessageAttributeValue()
        .withStringValue("0.50") //Sets the max price to 0.50 USD.
        .withDataType("Number"));
    smsAttributes.put("AWS.SNS.SMS.SMSType", new MessageAttributeValue()
        .withStringValue("Promotional") //Sets the type to promotional.
        .withDataType("String"));
}
```

Publish a message to a topic. The message is sent to every subscriber.

```
public static void sendSMSMessageToTopic(AmazonSNSClient snsClient, String topicArn,
                                         String message, Map<String, MessageAttributeValue> smsAttributes) {
    PublishResult result = snsClient.publish(new PublishRequest()
        .withTopicArn(topicArn)
        .withMessage(message)
        .withMessageAttributes(smsAttributes));
    System.out.println(result);
}
```

Call the previous functions to create a topic, subscribe a phone number, set message attributes, and publish a message to the topic.

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();

    String topicArn = createSNSTopic(snsClient);
    String phoneNumber = "+1XXX5550100";
    // Specify a protocol of "sms" when subscribing a phone number.
    subscribeToTopic(snsClient, topicArn, "sms", phoneNumber);

    String message = "My SMS message";
    Map<String, MessageAttributeValue> smsAttributes =
        new HashMap<String, MessageAttributeValue>();
    addMessageAttributes(smsAttributes)
    sendSMSMessageToTopic(snsClient, topicArn, message, smsAttributes);
}
```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK

The following code example shows how to publish a large message to Amazon SNS using Amazon S3 to store the message payload.

Java

### SDK for Java 1.x

To publish a large message, use the Amazon SNS Extended Client Library for Java. The message that you send references an Amazon S3 object containing the actual message content.

```
import com.amazonaws.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.sqs.javamessaging.ExtendedClientConfiguration;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
```

```

import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;
import com.amazonaws.services.sns.util.Topics;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import software.amazon sns AmazonSNSExtendedClient;
import software.amazon sns SNSExtendedClientConfiguration;

public class Example {

    public static void main(String[] args) {
        final String BUCKET_NAME = "extended-client-bucket";
        final String TOPIC_NAME = "extended-client-topic";
        final String QUEUE_NAME = "extended-client-queue";
        final Regions region = Regions.DEFAULT_REGION;

        //Message threshold controls the maximum message size that will be allowed
        to be published
        //through SNS using the extended client. Payload of messages exceeding this
        value will be stored in
        //S3. The default value of this parameter is 256 KB which is the maximum
        message size in SNS (and SQS).
        final int EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD = 32;

        //Initialize SNS, SQS and S3 clients
        final AmazonSNS snsClient =
        AmazonSNSClientBuilder.standard().withRegion(region).build();
        final AmazonSQS sqsClient =
        AmazonSQSClientBuilder.standard().withRegion(region).build();
        final AmazonS3 s3Client =
        AmazonS3ClientBuilder.standard().withRegion(region).build();

        //Create bucket, topic, queue and subscription
        s3Client.createBucket(BUCKET_NAME);
        final String topicArn = snsClient.createTopic(
            new CreateTopicRequest().withName(TOPIC_NAME)
        ).getTopicArn();
        final String queueUrl = sqsClient.createQueue(
            new CreateQueueRequest().withQueueName(QUEUE_NAME)
        ).getQueueUrl();
        final String subscriptionArn = Topics.subscribeQueue(
            snsClient, sqsClient, topicArn, queueUrl
        );

        //To read message content stored in S3 transparently through SQS extended
        client,
        //set the RawMessageDelivery subscription attribute to TRUE
        final SetSubscriptionAttributesRequest subscriptionAttributesRequest = new
        SetSubscriptionAttributesRequest();
        subscriptionAttributesRequest.setSubscriptionArn(subscriptionArn);
        subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
        subscriptionAttributesRequest.setAttributeValue("TRUE");
        snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);

        //Initialize SNS extended client
        //PayloadSizeThreshold triggers message content storage in S3 when the
        threshold is exceeded
        //To store all messages content in S3, use AlwaysThroughS3 flag
        final SNSExtendedClientConfiguration snsExtendedClientConfiguration = new
        SNSExtendedClientConfiguration()
            .withPayloadSupportEnabled(s3Client, BUCKET_NAME)
            .withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
        final AmazonSNSExtendedClient snsExtendedClient = new
        AmazonSNSExtendedClient(snsClient, snsExtendedClientConfiguration);
    }
}

```

```
//Publish message via SNS with storage in S3
final String message = "This message is stored in S3 as it exceeds the
threshold of 32 bytes set above.";
snsExtendedClient.publish(topicArn, message);

//Initialize SQS extended client
final ExtendedClientConfiguration sqsExtendedClientConfiguration = new
ExtendedClientConfiguration()
    .withPayloadSupportEnabled(s3Client, BUCKET_NAME);
final AmazonSQSExtendedClient sqsExtendedClient =
    new AmazonSQSExtendedClient(sqsClient,
sqsExtendedClientConfiguration);

//Read the message from the queue
final ReceiveMessageResult result =
sqsExtendedClient.receiveMessage(queueUrl);
System.out.println("Received message is " +
result.getMessages().get(0).getBody());
}
```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## API examples for Amazon SNS

The following code examples show how to use Amazon SNS with AWS SDKs.

### Examples

- [Add tags to an Amazon SNS topic using an AWS SDK \(p. 281\)](#)
- [Check whether a phone number is opted out of Amazon SNS using an AWS SDK \(p. 281\)](#)
- [Confirm an endpoint owner wants to receive Amazon SNS messages using an AWS SDK \(p. 284\)](#)
- [Create an Amazon SNS topic using an AWS SDK \(p. 286\)](#)
- [Delete an Amazon SNS subscription using an AWS SDK \(p. 291\)](#)
- [Delete an Amazon SNS topic using an AWS SDK \(p. 294\)](#)
- [Get the properties of an Amazon SNS topic using an AWS SDK \(p. 297\)](#)
- [Get the settings for sending Amazon SNS SMS messages using an AWS SDK \(p. 301\)](#)
- [List phone numbers opted out of Amazon SNS using an AWS SDK \(p. 304\)](#)
- [List the subscribers of an Amazon SNS topic using an AWS SDK \(p. 305\)](#)
- [List Amazon SNS topics using an AWS SDK \(p. 310\)](#)
- [Publish an Amazon SNS SMS text message using an AWS SDK \(p. 315\)](#)
- [Publish to an Amazon SNS topic using an AWS SDK \(p. 318\)](#)
- [Set a dead-letter queue for an Amazon SNS subscription using an AWS SDK \(p. 324\)](#)
- [Set an Amazon SNS filter policy using an AWS SDK \(p. 324\)](#)
- [Set the default settings for sending Amazon SNS SMS messages using an AWS SDK \(p. 326\)](#)
- [Set Amazon SNS topic attributes using an AWS SDK \(p. 328\)](#)
- [Subscribe a Lambda function to receive notifications from an Amazon SNS topic using an AWS SDK \(p. 331\)](#)

- [Subscribe a mobile application to an Amazon SNS topic using an AWS SDK \(p. 334\)](#)
- [Subscribe an HTTP endpoint to an Amazon SNS topic using an AWS SDK \(p. 336\)](#)
- [Subscribe an email address to an Amazon SNS topic using an AWS SDK \(p. 337\)](#)

## Add tags to an Amazon SNS topic using an AWS SDK

The following code example shows how to add tags to an Amazon SNS topic.

Java

### SDK for Java 2.x

```
public static void addTopicTags(SnsClient snsClient, String topicArn) {  
  
    try {  
        Tag tag = Tag.builder()  
            .key("Team")  
            .value("Development")  
            .build();  
  
        Tag tag2 = Tag.builder()  
            .key("Environment")  
            .value("Gamma")  
            .build();  
  
        List<Tag> tagList = new ArrayList<>();  
        tagList.add(tag);  
        tagList.add(tag2);  
  
        TagResourceRequest tagResourceRequest = TagResourceRequest.builder()  
            .resourceArn(topicArn)  
            .tags(tagList)  
            .build();  
  
        snsClient.tagResource(tagResourceRequest);  
        System.out.println("Tags have been added to "+topicArn);  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [TagResource in AWS SDK for Java 2.x API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Check whether a phone number is opted out of Amazon SNS using an AWS SDK

The following code examples show how to check whether a phone number is opted out of receiving Amazon SNS messages.

.NET

AWS SDK for .NET

```
/// <summary>
/// Checks to see if the supplied phone number has been opted out.
/// </summary>
/// <param name="client">The initialized Amazon SNS Client object used
/// to check if the phone number has been opted out.</param>
/// <param name="phoneNumber">A string representing the phone number
/// to check.</param>
public static async Task<br/>
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)<br/>
{
    var request = new CheckIfPhoneNumberIsOptedOutRequest<br/>
    {
        PhoneNumber = phoneNumber,
    };<br/>
    try<br/>
    {
        var response = await<br/>
client.CheckIfPhoneNumberIsOptedOutAsync(request);<br/>
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            string optOutStatus = response.IsOptedOut ? "opted out" : "not
opted out.";
            Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
        }
    }
    catch (AuthorizationErrorException ex)
    {
        Console.WriteLine($"{ex.Message}");
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

```
public static void checkPhone(SnsClient snsClient, String phoneNumber) {<br/>
    try {<br/>
        CheckIfPhoneNumberIsOptedOutRequest request =<br/>
CheckIfPhoneNumberIsOptedOutRequest.builder()<br/>
            .phoneNumber(phoneNumber)<br/>
            .build();<br/>
<br/>
        CheckIfPhoneNumberIsOptedOutResponse result =<br/>
snsClient.checkIfPhoneNumberIsOptedOut(request);<br/>
<br/>
        System.out.println(result.isOptedOut() + "Phone Number " + phoneNumber<br/>
+ " has Opted Out of receiving sns messages." +<br/>
```

```
    "\n\nStatus was " + result.sdkHttpResponse().statusCode());  
}  
} catch (SnsException e) {  
    System.err.println(e.awsErrorDetails().errorMessage());  
    System.exit(1);  
}  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create SNS service object.  
const snsClient = new SNSClient({ region: REGION });  
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";  
import {snsClient } from "./libs/snsClient.js";  
  
// Set the parameters  
const params = { phoneNumber: "353861230764" }; //PHONE_NUMBER, in the E.164 phone  
// number structure  
  
const run = async () => {  
    try {  
        const data = await snsClient.send(  
            new CheckIfPhoneNumberIsOptedOutCommand(params)  
        );  
        console.log("Success.", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err.stack);  
    }  
};  
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Indicates whether the phone number owner has opted out of receiving SMS messages
 * from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$phone = '+1XXX5550100';

try {
    $result = $SnSclient->checkIfPhoneNumberIsOptedOut([
        'phoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [CheckIfPhoneNumberIsOptedOut](#) in [AWS SDK for PHP API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Confirm an endpoint owner wants to receive Amazon SNS messages using an AWS SDK

The following code examples show how to confirm the owner of an endpoint wants to receive Amazon SNS messages by validating the token sent to the endpoint by an earlier Subscribe action.

Java

### SDK for Java 2.x

```
public static void confirmSub(SnsClient snsClient, String subscriptionToken,
String topicArn ) {

    try {
        ConfirmSubscriptionRequest request =
ConfirmSubscriptionRequest.builder()
            .token(subscriptionToken)
```

```
        .topicArn(topicArn)
        .build();

    ConfirmSubscriptionResponse result =
snsClient.confirmSubscription(request);
    System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode() + "\n\nSubscription Arn: \n\n" +
result.subscriptionArn());

} catch (SnsException e) {
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ConfirmSubscription](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import {snsClient } from "./libs/snsClient.js";

// Set the parameters
const params = {
  Token: "TOKEN", // Required. Token sent to the endpoint by an earlier Subscribe
  action. */
  TopicArn: "TOPIC_ARN", // Required
  AuthenticateOnUnsubscribe: "true", // 'true' or 'false'
};

const run = async () => {
  try {
    const data = await snsClient.send(new ConfirmSubscriptionCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [ConfirmSubscription](#) in *AWS SDK for JavaScript API Reference*.

PHP

#### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Verifies an endpoint owner's intent to receive messages by validating the token
 * sent to the endpoint by an earlier Subscribe action.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription_token = 'arn:aws:sns:us-east-1:111122223333:MyTopic:123456-
abcd-12ab-1234-12ba3dc1234a';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Token' => $subscription_token,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ConfirmSubscription](#) in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Create an Amazon SNS topic using an AWS SDK

The following code examples show how to create an Amazon SNS topic.

.NET

#### AWS SDK for .NET

```
/// <summary>
```

```
/// Creates a new SNS topic using the supplied topic name.  
/// </summary>  
/// <param name="client">The initialized SNS client object used to  
/// create the new topic.</param>  
/// <param name="topicName">A string representing the topic name.</param>  
/// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>  
public static async Task<string>  
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)  
{  
    var request = new CreateTopicRequest  
    {  
        Name = topicName,  
    };  
  
    var response = await client.CreateTopicAsync(request);  
  
    return response.TopicArn;  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for .NET API Reference](#).

## C++

### SDK for C++

```
Aws::SDKOptions options;  
Aws::InitAPI(options);  
{  
    Aws::String topic_name = argv[1];  
    Aws::SNS::SNSClient sns;  
  
    Aws::SNS::Model::CreateTopicRequest ct_req;  
    ct_req.SetName(topic_name);  
  
    auto ct_out = sns.CreateTopic(ct_req);  
  
    if (ct_out.IsSuccess())  
    {  
        std::cout << "Successfully created topic " << topic_name << std::endl;  
    }  
    else  
    {  
        std::cout << "Error creating topic " << topic_name << ":" <<  
        ct_out.GetError().GetMessage() << std::endl;  
    }  
}  
  
Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for C++ API Reference](#).

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).

- For API details, see [CreateTopic in AWS SDK for Go API Reference](#).

## Java

### SDK for Java 2.x

```
public static String createSNSTopic(SnsClient snsClient, String topicName ) {  
  
    CreateTopicResponse result = null;  
    try {  
        CreateTopicRequest request = CreateTopicRequest.builder()  
            .name(topicName)  
            .build();  
  
        result = snsClient.createTopic(request);  
        return result.topicArn();  
    } catch (SnsException e) {  
  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
    return "";  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Java 2.x API Reference](#).

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create SNS service object.  
const snsClient = new SNSClient({ region: REGION });  
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {CreateTopicCommand} from "@aws-sdk/client-sns";  
import {snsClient} from "./libs/snsClient.js";  
  
// Set the parameters  
const params = { Name: "TOPIC_NAME" }; //TOPIC_NAME  
  
const run = async () => {  
    try {  
        const data = await snsClient.send(new CreateTopicCommand(params));  
        console.log("Success.", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err.stack);  
    }  
};
```

```
    }
};

run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [CreateTopic](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Create a Simple Notification Service topics in your AWS account at the requested
region.
*
* This code expects that you have AWS credentials set up per:
* https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
*/

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topicname = 'myTopic';

try {
    $result = $SnSclient->createTopic([
        'Name' => $topicname,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [CreateTopic](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
```

```
def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

def create_topic(self, name):
    """
    Creates a notification topic.

    :param name: The name of the topic to create.
    :return: The newly created topic.
    """
    try:
        topic = self.sns_resource.create_topic(Name=name)
        logger.info("Created topic %s with ARN %s.", name, topic.arn)
    except ClientError:
        logger.exception("Couldn't create topic %s.", name)
        raise
    else:
        return topic
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def topic_created?(sns_client, topic_name)

  sns_client.create_topic(name: topic_name)
  rescue StandardError => e
    puts "Error while creating the topic named '#{topic_name}': #{e.message}"
  end

# Full example call:
def run_me
  topic_name = 'TOPIC_NAME'
  region = 'REGION'

  sns_client = Aws::SNS::Client.new(region: region)

  puts "Creating the topic '#{topic_name}'..."

  if topic_created?(sns_client, topic_name)
    puts 'The topic was created.'
  else
    puts 'The topic was not created. Stopping program.'
    exit 1
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).

- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [CreateTopic in AWS SDK for Ruby API Reference](#).

Rust

#### SDK for Rust

##### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn.as_deref().unwrap_or_default()
    );

    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [CreateTopic in AWS SDK for Rust API reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Delete an Amazon SNS subscription using an AWS SDK

The following code examples show how to delete an Amazon SNS subscription.

C++

#### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String subscription_arn = argv[1];

    Aws::SNS::Model::UnsubscribeRequest s_req;
    s_req.SetSubscriptionArn(subscription_arn);

    auto s_out = sns.Unsubscribe(s_req);

    if (s_out.IsSuccess())
    {
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else
```

```
    std::cout << "Error while unsubscribing " << s_out.GetError().GetMessage()
    << std::endl;
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe in AWS SDK for C++ API Reference](#).

## Java

### SDK for Java 2.x

```
public static void unSub(SnsClient snsClient, String subscriptionArn) {

    try {
        UnsubscribeRequest request = UnsubscribeRequest.builder()
            .subscriptionArn(subscriptionArn)
            .build();

        UnsubscribeResponse result = snsClient.unsubscribe(request);

        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode()
            + "\n\nSubscription was removed for " + request.subscriptionArn());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe in AWS SDK for Java 2.x API Reference](#).

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {UnsubscribeCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";
```

```
// Set the parameters
const params = { SubscriptionArn: "TOPIC_SUBSCRIPTION_ARN" }; // 
TOPIC_SUBSCRIPTION_ARN

const run = async () => {
  try {
    const data = await snsClient.send(new UnsubscribeCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};

run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Unsubscribe in AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Deletes a subscription to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $SnSclient->unsubscribe([
        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Unsubscribe in AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:  
    """Encapsulates Amazon SNS topic and subscription functions."""  
    def __init__(self, sns_resource):  
        """  
        :param sns_resource: A Boto3 Amazon SNS resource.  
        """  
        self.sns_resource = sns_resource  
  
    def delete_subscription(subscription):  
        """  
        Unsubscribes and deletes a subscription.  
        """  
        try:  
            subscription.delete()  
            logger.info("Deleted subscription %s.", subscription.arn)  
        except ClientError:  
            logger.exception("Couldn't delete subscription %s.", subscription.arn)  
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Unsubscribe in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Delete an Amazon SNS topic using an AWS SDK

The following code examples show how to delete an Amazon SNS topic and all subscriptions to that topic.

### .NET

#### AWS SDK for .NET

```
/// <summary>  
/// This example deletes an existing Amazon Simple Notification Service  
/// (Amazon SNS) topic. The example was created using the AWS SDK for .NET  
/// version 3.7 and .NET Core 5.0.  
/// </summary>  
public class DeleteSNSTopic  
{  
    public static async Task Main()  
    {  
        string topicArn = "arn:aws:sns:us-east-2:704825161248:ExampleSNSTopic";  
        IAmazonSimpleNotificationService client = new  
        AmazonSimpleNotificationServiceClient();  
  
        var response = await client.DeleteTopicAsync(topicArn);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for .NET API Reference*.

C++

**SDK for C++**

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::String topic_arn = argv[1];
    Aws::SNS::SNSClient sns;

    Aws::SNS::Model::DeleteTopicRequest dt_req;
    dt_req.SetTopicArn(topic_arn);

    auto dt_out = sns.DeleteTopic(dt_req);

    if (dt_out.IsSuccess())
    {
        std::cout << "Successfully deleted topic " << topic_arn << std::endl;
    }
    else
    {
        std::cout << "Error deleting topic " << topic_arn << ":" <<
        dt_out.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for C++ API Reference*.

Java

**SDK for Java 2.x**

```
public static void deleteSNSTopic(SnsClient snsClient, String topicArn ) {

    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).

- For API details, see [DeleteTopic](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js

// Import required AWS SDK clients and commands for Node.js
import {DeleteTopicCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { TopicArn: "TOPIC_ARN" }; //TOPIC_ARN

const run = async () => {
  try {
    const data = await snsClient.send(new DeleteTopicCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Deletes a SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */
```

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteTopic](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Get the properties of an Amazon SNS topic using an AWS SDK

The following code examples show how to get the properties of an Amazon SNS topic.

.NET

AWS SDK for .NET

```
/// <summary>
/// This example shows how to retrieve the attributes of an Amazon Simple
/// Notification Service (Amazon SNS) topic. The example was written using
/// the AWS SDK for .NET 3.7 and .NET Core 5.0.
/// </summary>
public class GetTopicAttributes
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-west-2:000000000000:ExampleSNSTopic";
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var attributes = await GetTopicAttributesAsync(client, topicArn);
        DisplayTopicAttributes(attributes);
    }

    /// <summary>
    /// Given the ARN of the Amazon SNS topic, this method retrieves the topic
    /// attributes.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the attributes for the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic for which to retrieve
    /// the attributes.</param>
    /// <returns>A Dictionary of topic attributes.</returns>
    public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
{
    var response = await client.GetTopicAttributesAsync(topicArn);

    return response.Attributes;
}

    /// <summary>
    /// This method displays the attributes for an Amazon SNS topic.
    /// </summary>
    /// <param name="topicAttributes">A Dictionary containing the
    /// attributes for an Amazon SNS topic.</param>
    public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
{
    foreach (KeyValuePair<string, string> entry in topicAttributes)
    {
        Console.WriteLine($"{entry.Key}: {entry.Value}\n");
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetTopicAttributes](#) in *AWS SDK for .NET API Reference*.

C++

**SDK for C++**

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String topic_arn = argv[1];

    Aws::SNS::Model::GetTopicAttributesRequest gta_req;
    gta_req.SetTopicArn(topic_arn);

    auto gta_out = sns.GetTopicAttributes(gta_req);

    if (gta_out.IsSuccess())
    {
        std::cout << "Topic Attributes:" << std::endl;
        for (auto const &attribute : gta_out.GetResult().GetAttributes())
        {
            std::cout << "  * " << attribute.first << " : " << attribute.second <<
        std::endl;
        }
    }
    else
    {
        std::cout << "Error while getting Topic attributes " <<
        gta_out.GetError().GetMessage()
        << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetTopicAttributes](#) in *AWS SDK for C++ API Reference*.

Java

**SDK for Java 2.x**

```
public static void getSNSTopicAttributes(SnsClient snsClient, String topicArn )
{
    try {
        GetTopicAttributesRequest request = GetTopicAttributesRequest.builder()
            .topicArn(topicArn)
            .build();

        GetTopicAttributesResponse result =
        snsClient.getTopicAttributes(request);
        System.out.println("\n\nStatus is " +
result.sdkHttpResponse().statusCode() + "\n\nAttributes: \n\n" +
result.attributes());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetTopicAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {GetTopicAttributesCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { TopicArn: "TOPIC_ARN" }; // TOPIC_ARN

const run = async () => {
  try {
    const data = await snsClient.send(new GetTopicAttributesCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetTopicAttributes](#) in *AWS SDK for JavaScript API Reference*.

### SDK for JavaScript V2

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set region
AWS.config.update({region: 'REGION'});

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({apiVersion: '2010-03-31'}).getTopicAttributes({TopicArn: 'TOPIC_ARN'}).promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise.then(
```

```
function(data) {
    console.log(data);
}).catch(
    function(err) {
        console.error(err, err.stack);
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetTopicAttributes](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->getTopicAttributes([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetTopicAttributes](#) in [AWS SDK for PHP API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Get the settings for sending Amazon SNS SMS messages using an AWS SDK

The following code examples show how to get the settings for sending Amazon SNS SMS messages.

### C++

#### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;

    Aws::SNS::Model::GetSMSAttributesRequest gsmst_req;
```

```
//Set the request to only retrieve the DefaultSMSType setting.  
//Without the following line, GetSMSAttributes would retrieve all settings.  
gsmst_req.AddAttributes("DefaultSMSType");  
  
auto gsmst_out = sns.GetSMSAttributes(gsmst_req);  
  
if (gsmst_out.IsSuccess())  
{  
    for (auto const& att : gsmst_out.GetResult().GetAttributes())  
    {  
        std::cout << att.first << ":" << att.second << std::endl;  
    }  
}  
else  
{  
    std::cout << "Error while getting SMS Type: '" <<  
gsmst_out.GetError().GetMessage()  
        << "'" << std::endl;  
}  
  
Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetSMSAttributes](#) in *AWS SDK for C++ API Reference*.

## Java

### SDK for Java 2.x

```
public static void getSNSAttributes(SnsClient snsClient, String topicArn ) {  
  
    try {  
        GetSubscriptionAttributesRequest request =  
GetSubscriptionAttributesRequest.builder()  
            .subscriptionArn(topicArn)  
            .build();  
  
        // Get the Subscription attributes  
        GetSubscriptionAttributesResponse res =  
snsClient.getSubscriptionAttributes(request);  
        Map<String, String> map = res.attributes();  
  
        // Iterate through the map  
        Iterator iter = map.entrySet().iterator();  
        while (iter.hasNext()) {  
            Map.Entry entry = (Map.Entry) iter.next();  
            System.out.println("[Key] : " + entry.getKey() + " [Value] : " +  
entry.getValue());  
        }  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
  
    System.out.println("\n\nStatus was good");  
}
```

- Find instructions and more code on [GitHub](#).

- For API details, see [GetSMSAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {GetSMSAttributesCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
var params = {
  attributes: [
    "DefaultSMSType",
    "ATTRIBUTE_NAME",
    /* more items */
  ],
};

const run = async () => {
  try {
    const data = await snsClient.send(new GetSMSAttributesCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetSMSAttributes](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Get the type of SMS Message sent by default from the AWS SNS service.
 *
```

```
* This code expects that you have AWS credentials set up per:  
* https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
guide_credentials.html  
*/  
  
$SnSclient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
try {  
    $result = $SnSclient->getSMSAttributes([  
        'attributes' => ['DefaultSMSType'],  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [GetSMSAttributes](#) in [AWS SDK for PHP API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## List phone numbers opted out of Amazon SNS using an AWS SDK

The following code examples show how to list phone numbers that are opted out of receiving Amazon SNS messages.

Java

### SDK for Java 2.x

```
public static void listOpts( SnsClient snsClient ) {  
  
    try {  
  
        ListPhoneNumbersOptedOutRequest request =  
ListPhoneNumbersOptedOutRequest.builder().build();  
        ListPhoneNumbersOptedOutResponse result =  
snsClient.listPhoneNumbersOptedOut(request);  
        System.out.println("Status is " + result.sdkHttpResponse().statusCode()  
+ "\n\nPhone Numbers: \n\n" + result.phoneNumbers());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).

- For API details, see [ListPhoneNumbersOptedOut](#) in *AWS SDK for Java 2.x API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages
 * from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnsClient->listPhoneNumbersOptedOut([
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [ListPhoneNumbersOptedOut](#) in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## List the subscribers of an Amazon SNS topic using an AWS SDK

The following code examples show how to retrieve the list of subscribers of an Amazon SNS topic.

### .NET

#### AWS SDK for .NET

```
/// <summary>
/// This example will retrieve a list of the existing Amazon Simple
```

```
/// Notification Service (Amazon SNS) subscriptions. The example was
/// created using the AWS SDK for .NET 3.7 and .NET Core 5.0.
/// </summary>
public class ListSubscriptions
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var subscriptions = await GetSubscriptionsListAsync(client);

        DisplaySubscriptionList(subscriptions);
    }

    /// <summary>
    /// Gets a list of the existing Amazon SNS subscriptions.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to obtain the list of subscriptions.</param>
    /// <returns>A List containing information about each subscription.</returns>
    public static async Task<List<Subscription>>
GetSubscriptionsListAsync(IAmazonSimpleNotificationService client)
    {
        var response = await client.ListSubscriptionsAsync();

        return response.Subscriptions;
    }

    /// <summary>
    /// Display a list of Amazon SNS subscription information.
    /// </summary>
    /// <param name="subscriptionList">A list containing details for existing
    /// Amazon SNS subscriptions.</param>
    public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
    {
        foreach (var subscription in subscriptionList)
        {
            Console.WriteLine($"Owner: {subscription.Owner}");
            Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
            Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
            Console.WriteLine($"Endpoint: {subscription.Endpoint}");
            Console.WriteLine($"Protocol: {subscription.Protocol}");
            Console.WriteLine();
        }
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in [AWS SDK for .NET API Reference](#).

C++

#### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
```

```
{  
    Aws::SNS::SNSClient sns;  
  
    Aws::SNS::Model::ListSubscriptionsRequest ls_req;  
  
    auto ls_out = sns.ListSubscriptions(ls_req);  
  
    if (ls_out.IsSuccess())  
    {  
        std::cout << "Subscriptions list:" << std::endl;  
        for (auto const& subscription : ls_out.GetResult().GetSubscriptions())  
        {  
            std::cout << " * " << subscription.GetSubscriptionArn() << std::endl;  
        }  
    }  
    else  
    {  
        std::cout << "Error listing subscriptions " << ls_out.GetError().GetMessage()  
<<  
        std::endl;  
    }  
}  
  
Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in *AWS SDK for C++ API Reference*.

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in *AWS SDK for Go API Reference*.

## Java

### SDK for Java 2.x

```
public static void listSNSSubscriptions( SnsClient snsClient) {  
  
    try {  
        ListSubscriptionsRequest request = ListSubscriptionsRequest.builder()  
            .build();  
  
        ListSubscriptionsResponse result =  
snsClient.listSubscriptions(request);  
        System.out.println(result.subscriptions());  
  
    } catch (SnsException e) {  
  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {ListSubscriptionsByTopicCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = { TopicArn: "TOPIC_ARN" }; //TOPIC_ARN

const run = async () => {
  try {
    const data = await snsClient.send(new ListSubscriptionsByTopicCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListSubscriptions](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Returns a list of Amazon SNS subscriptions in the requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
```

```
]);
try {
    $result = $SnSclient->listSubscriptions([
        ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def list_subscriptions(self, topic=None):
        """
        Lists subscriptions for the current account, optionally limited to a
        specific topic.

        :param topic: When specified, only subscriptions to this topic are
        returned.
        :return: An iterator that yields the subscriptions.
        """
        try:
            if topic is None:
                subs_iter = self.sns_resource.subscriptions.all()
            else:
                subs_iter = topic.subscriptions.all()
                logger.info("Got subscriptions.")
        except ClientError:
            logger.exception("Couldn't get subscriptions.")
            raise
        else:
            return subs_iter
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListSubscriptions](#) in *AWS SDK for Python (Boto3) API Reference*.

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'
```

```
def show_subscriptions?(sns_client, topic_arn)
  topic = sns_client.topic(topic_arn)
  topic.subscriptions.each do |s|
    puts s.attributes['Endpoint']
  end

rescue StandardError => e
  puts "Error while sending the message: #{e.message}"
end

def run_me

  topic_arn = 'SNS_TOPIC_ARN'
  region = 'REGION'

  sns_client = Aws::SNS::Resource.new(region: region)

  puts "Listing subscriptions to the topic."

  if show_subscriptions?(sns_client, topic_arn)
  else
    puts 'There was an error. Stopping program.'
    exit 1
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [ListSubscriptions](#) in [AWS SDK for Ruby API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## List Amazon SNS topics using an AWS SDK

The following code examples show how to list Amazon SNS topics.

.NET

### AWS SDK for .NET

```
public static async Task Main()
{
    IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

    await GetTopicListAsync(client);
}

/// <summary>
/// Retrieves the list of Amazon SNS topics in groups of up to 100
/// topics.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object used
```

```
    /// to retrieve the list of topics.</param>
    public static async Task<IList<AmazonSimpleNotificationServiceTopic>> GetTopicListAsync(IAmazonSimpleNotificationService client)
    {
        // If there are more than 100 Amazon SNS topics, the call to
        // ListTopicsAsync will return a value to pass to the
        // method to retrieve the next 100 (or less) topics.
        string nextToken = string.Empty;

        do
        {
            var response = await client.ListTopicsAsync(nextToken);
            DisplayTopicsList(response.Topics);
            nextToken = response.NextToken;
        }
        while (!string.IsNullOrEmpty(nextToken));
    }

    /// <summary>
    /// Displays the list of Amazon SNS Topic ARNs.
    /// </summary>
    /// <param name="topicList">The list of Topic ARNs.</param>
    public static void DisplayTopicsList(List<AmazonSimpleNotificationServiceTopic> topicList)
    {
        foreach (var topic in topicList)
        {
            Console.WriteLine($"{{topic.TopicArn}}");
        }
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in *AWS SDK for .NET API Reference*.

## C++

### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;

    Aws::SNS::Model::ListTopicsRequest lt_req;

    auto lt_out = sns.ListTopics(lt_req);

    if (lt_out.IsSuccess())
    {
        std::cout << "Topics list:" << std::endl;
        for (auto const &topic : lt_out.GetResult().GetTopics())
        {
            std::cout << "    * " << topic.GetTopicArn() << std::endl;
        }
    }
    else
    {
        std::cout << "Error listing topics " << lt_out.GetError().GetMessage() <<
                    std::endl;
    }
}
```

```
Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in *AWS SDK for C++ API Reference*.

Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in *AWS SDK for Go API Reference*.

Java

### SDK for Java 2.x

```
public static void listSNSTopics(SnsClient snsClient) {  
  
    try {  
        ListTopicsRequest request = ListTopicsRequest.builder()  
            .build();  
  
        ListTopicsResponse result = snsClient.listTopics(request);  
        System.out.println("Status was " +  
            result.sdkHttpResponse().statusCode() + "\n\nTopics\n" + result.topics());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create SNS service object.  
const snsClient = new SNSClient({ region: REGION });  
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {ListTopicsCommand} from "@aws-sdk/client-sns";  
import {snsClient} from "./libs/snsClient.js";
```

```
const run = async () => {
  try {
    const data = await snsClient.send(new ListTopicsCommand({}));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};

run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListTopics](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Returns a list of the requester's topics from your AWS SNS account in the region
 * specified.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSClient->listTopics([
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in [AWS SDK for PHP API Reference](#).

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
```

```
"""Encapsulates Amazon SNS topic and subscription functions."""
def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

def list_topics(self):
    """
    Lists topics for the current account.

    :return: An iterator that yields the topics.
    """
    try:
        topics_iter = self.sns_resource.topics.all()
        logger.info("Got topics.")
    except ClientError:
        logger.exception("Couldn't get topics.")
        raise
    else:
        return topics_iter
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics in AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def list_topics?(sns_client)
    sns_client.topics.each do |topic|
        puts topic.arn
    rescue StandardError => e
        puts "Error while listing the topics: #{e.message}"
    end
end

def run_me

    region = 'REGION'
    sns_client = Aws::SNS::Resource.new(region: region)

    puts "Listing the topics."

    if list_topics?(sns_client)
    else
        puts 'The bucket was not created. Stopping program.'
        exit 1
    end
end
run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [ListTopics in AWS SDK for Ruby API Reference](#).

Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn show_topics(client: &Client) -> Result<(), Error> {
    let resp = client.list_topics().send().await?;

    println!("Topic ARNs:");

    for topic in resp.topics.unwrap_or_default() {
        println!("{}", topic.topic_arn.as_deref().unwrap_or_default());
    }

    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListTopics](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Publish an Amazon SNS SMS text message using an AWS SDK

The following code examples show how to publish SMS messages using Amazon SNS.

C++

### SDK for C++

```
/** 
 * Publish SMS: use Amazon SNS to send an SMS text message to a phone number.
 * Note: This requires additional AWS configuration prior to running example.
 *
 * NOTE: When you start using Amazon SNS to send SMS messages, your AWS account is
 * in the SMS sandbox and you can only
 * use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
 * latest/dg/sns-sms-sandbox.html.
 * NOTE: If destination is in the US, you also have an additional restriction that
 * you have use a dedicated
 * origination ID (phone number). You can request an origination number using
 * Amazon Pinpoint for a fee.
 * See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-
 * origination-numbers-with-amazon-sns/
 * for more information.
 *
 * <phone_number_value> input parameter uses E.164 format.
 * For example, in United States, this input value should be of the form:
 * +12223334444
 */
int main(int argc, char ** argv)
{
```

```
if (argc != 3)
{
    std::cout << "Usage: publish_sms <message_value> <phone_number_value> " <<
std::endl;
    return 1;
}

Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String message = argv[1];
    Aws::String phone_number = argv[2];

    Aws::SNS::Model::PublishRequest psms_req;
    psms_req.SetMessage(message);
    psms_req.SetPhoneNumber(phone_number);

    auto psms_out = sns.Publish(psms_req);

    if (psms_out.IsSuccess())
    {
        std::cout << "Message published successfully " <<
psms_out.GetResult().GetMessageId()
        << std::endl;
    }
    else
    {
        std::cout << "Error while publishing message " <<
psms_out.GetError().GetMessage()
        << std::endl;
    }
}

Aws::ShutdownAPI(options);
return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for C++ API Reference*.

## Java

### SDK for Java 2.x

```
public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .phoneNumber(phoneNumber)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out.println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Java 2.x API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Sends a a text message (SMS message) directly to a phone number using Amazon
 * SNS.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';

try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'PhoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Publish](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
```

```
"""
    self.sns_resource = sns_resource

def publish_text_message(self, phone_number, message):
    """
        Publishes a text message directly to a phone number without need for a
        subscription.

        :param phone_number: The phone number that receives the message. This must
        be
                in E.164 format. For example, a United States phone
                number might be +12065550101.
        :param message: The message to send.
        :return: The ID of the message.
    """
    try:
        response = self.sns_resource.meta.client.publish(
            PhoneNumber=phone_number, Message=message)
        message_id = response['MessageId']
        logger.info("Published message to %s.", phone_number)
    except ClientError:
        logger.exception("Couldn't publish message to %s.", phone_number)
        raise
    else:
        return message_id
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Publish to an Amazon SNS topic using an AWS SDK

The following code examples show how to publish messages to an Amazon SNS topic.

.NET

### AWS SDK for .NET

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish in AWS SDK for .NET API Reference](#).

C++

### SDK for C++

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String message = argv[1];
    Aws::String topic_arn = argv[2];

    Aws::SNS::Model::PublishRequest psms_req;
    psms_req.SetMessage(message);
    psms_req.SetTopicArn(topic_arn);
```

```
auto psms_out = sns.Publish(psms_req);

if (psms_out.IsSuccess())
{
    std::cout << "Message published successfully " << std::endl;
}
else
{
    std::cout << "Error while publishing message " <<
    psms_out.GetError().GetMessage()
        << std::endl;
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for C++ API Reference*.

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Go API Reference*.

## Java

### SDK for Java 2.x

```
public static void pubTopic(SnsClient snsClient, String message, String
topicArn) {

    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out.println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {PublishCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
var params = {
    Message: "MESSAGE_TEXT", // MESSAGE_TEXT
    TopicArn: "TOPIC_ARN", //TOPIC_ARN
};

const run = async () => {
    try {
        const data = await snsClient.send(new PublishCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
```

```
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for PHP API Reference](#).

## Python

### **SDK for Python (Boto3)**

Publish a message with attributes so that a subscription can filter based on attributes.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def publish_message(topic, message, attributes):
        """
        Publishes a message, with attributes, to a topic. Subscriptions can be
        filtered
        based on message attributes so that a subscription receives messages only
        when specified attributes are present.

        :param topic: The topic to publish to.
        :param message: The message to publish.
        :param attributes: The key-value attributes to attach to the message.
        Values
                    must be either `str` or `bytes`.
        :return: The ID of the message.
        """
        try:
            att_dict = {}
            for key, value in attributes.items():
                if isinstance(value, str):
                    att_dict[key] = {'DataType': 'String', 'StringValue': value}
                elif isinstance(value, bytes):
                    att_dict[key] = {'DataType': 'Binary', 'BinaryValue': value}
            response = topic.publish(Message=message, MessageAttributes=att_dict)
            message_id = response['MessageId']
            logger.info(
                "Published message with attributes %s to topic %s.", attributes,
                topic.arn)
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id
```

Publish a message that takes different forms based on the protocol of the subscriber.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def publish_multi_message(
            topic, subject, default_message, sms_message, email_message):
        """
        Publishes a multi-format message to a topic. A multi-format message takes
        different forms based on the protocol of the subscriber. For example,
        an SMS subscriber might receive a short, text-only version of the message
        while an email subscriber could receive an HTML version of the message.

        :param topic: The topic to publish to.
        :param subject: The subject of the message.
        :param default_message: The default version of the message. This version is
                               sent to subscribers that have protocols that are
                               not
                               otherwise specified in the structured message.
        :param sms_message: The version of the message sent to SMS subscribers.
        :param email_message: The version of the message sent to email subscribers.
        :return: The ID of the message.
        """
        try:
            message = {
                'default': default_message,
                'sms': sms_message,
                'email': email_message
            }
            response = topic.publish(
                Message=json.dumps(message), Subject=subject,
                MessageStructure='json')
            message_id = response['MessageId']
            logger.info("Published multi-format message to topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish in AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def message_sent?(sns_client, topic_arn, message)

    sns_client.publish(topic_arn: topic_arn, message: message)
```

```
rescue StandardError => e
  puts "Error while sending the message: #{e.message}"
end

def run_me

  topic_arn = 'SNS_TOPIC_ARN'
  region = 'REGION'
  message = 'MESSAGE' # The text of the message to send.

  sns_client = Aws::SNS::Client.new(region: region)

  puts "Message sending."

  if message_sent?(sns_client, topic_arn, message)
    puts 'The message was sent.'
  else
    puts 'The message was not sent. Stopping program.'
    exit 1
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [Publish](#) in [AWS SDK for Ruby API Reference](#).

## Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`, topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;
```

```
    println!("Published message: {:?}", rsp);
    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Publish](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Set a dead-letter queue for an Amazon SNS subscription using an AWS SDK

The following code example shows how to set an Amazon SQS queue as a dead-letter queue for an Amazon SNS subscription.

Java

### SDK for Java 1.x

```
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}";

// Set the specified Amazon SQS queue as a dead-letter queue
// of the specified Amazon SNS subscription by setting the RedrivePolicy attribute.
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("RedrivePolicy")
    .withAttributeValue(redrivePolicy);
sns.setSubscriptionAttributes(request);
```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Set an Amazon SNS filter policy using an AWS SDK

The following code examples show how to set an Amazon SNS filter policy.

Java

### SDK for Java 2.x

```
public static void usePolicy(SnsClient snsClient, String subscriptionArn) {
```

```
try {
    SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();

    // Add a filter policy attribute with a single value
    fp.addAttribute("store", "example_corp");
    fp.addAttribute("event", "order_placed");

    // Add a prefix attribute
    fp.addAttributePrefix("customer_interests", "bas");

    // Add an anything-but attribute
    fp.addAttributeAnythingBut("customer_interests", "baseball");

    // Add a filter policy attribute with a list of values
    ArrayList<String> attributeValues = new ArrayList<>();
    attributeValues.add("rugby");
    attributeValues.add("soccer");
    attributeValues.add("hockey");
    fp.addAttribute("customer_interests", attributeValues);

    // Add a numeric attribute
    fp.addAttribute("price_usd", "=", 0);

    // Add a numeric attribute with a range
    fp.addAttributeRange("price_usd", ">", 0, "<=", 100);

    // Apply the filter policy attributes to an Amazon SNS subscription
    fp.apply(snsClient, subscriptionArn);

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSubscriptionAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def add_subscription_filter(subscription, attributes):
        """
        Adds a filter policy to a subscription. A filter policy is a key and a
        list of values that are allowed. When a message is published, it must have
        an
        attribute that passes the filter or it will not be sent to the
        subscription.

        :param subscription: The subscription the filter policy is attached to.
        :param attributes: A dictionary of key-value pairs that define the filter.
        
```

```
"""
try:
    att_policy = {key: [value] for key, value in attributes.items()}
    subscription.set_attributes(
        AttributeName='FilterPolicy',
        AttributeValue=json.dumps(att_policy))
    logger.info("Added filter to subscription %s.", subscription.arn)
except ClientError:
    logger.exception(
        "Couldn't add filter to subscription %s.", subscription.arn)
    raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSubscriptionAttributes](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Set the default settings for sending Amazon SNS SMS messages using an AWS SDK

The following code examples show how to set the default settings for sending SMS messages using Amazon SNS.

C++

### SDK for C++

How to use Amazon SNS to set the DefaultSMSType attribute.

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String sms_type = argv[1];

    Aws::SNS::Model::SetSMSAttributesRequest ssmst_req;
    ssmst_req.AddAttributes("DefaultSMSType", sms_type);

    auto ssmst_out = sns.SetSMSAttributes(ssmst_req);

    if (ssmst_out.IsSuccess())
    {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else
    {
        std::cout << "Error while setting SMS Type: '" <<
        ssmst_out.GetError().GetMessage()
        << "'" << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSmsAttributes](#) in *AWS SDK for C++ API Reference*.

Java

**SDK for Java 2.x**

```
public static void setSNSAttributes( SnsClient snsClient, HashMap<String, String> attributes) {

    try {
        SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
            .attributes(attributes)
            .build();

        SetSmsAttributesResponse result = snsClient.setSMSAttributes(request);
        System.out.println("Set default Attributes to " + attributes + ". Status was " + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetSmsAttributes](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

**SDK for JavaScript V3**

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import {snsClient } from "./libs/snsClient.js";

// Set the parameters
const params = {
    attributes: {
        /* required */
        DefaultSMSType: "Transactional" /* highest reliability */,
        //DefaultSMSType: 'Promotional' /* lowest cost */
    },
};

const run = async () => {
    try {
        const data = await snsClient.send(new SetSMSAttributesCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    }
```

```
    } catch (err) {
      console.log("Error", err.stack);
    }
};

run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [SetSmsAttributes](#) in [AWS SDK for JavaScript API Reference](#).

## PHP

### SDK for PHP

```
$SnSclient = new SnsClient([
  'profile' => 'default',
  'region'  => 'us-east-1',
  'version'  => '2010-03-31'
]);

try {
  $result = $SnSclient->SetSMSAttributes([
    'attributes' => [
      'DefaultSMSType' => 'Transactional',
    ],
  ]);
  var_dump($result);
} catch (AwsException $e) {
  // output error message if fails
  error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for PHP Developer Guide](#).
- For API details, see [SetSmsAttributes](#) in [AWS SDK for PHP API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Set Amazon SNS topic attributes using an AWS SDK

The following code examples show how to set Amazon SNS topic attributes.

### Java

#### SDK for Java 2.x

```
public static void setTopAttr(SnsClient snsClient, String attribute, String
topicArn, String value) {

  try {
```

```
SetTopicAttributesRequest request = SetTopicAttributesRequest.builder()
    .attributeName(attribute)
    .attributeValue(value)
    .topicArn(topicArn)
    .build();

SetTopicAttributesResponse result =
snsClient.setTopicAttributes(request);
System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode() + "\n\nTopic " + request.topicArn()
+ " updated " + request.attributeName() + " to " +
request.attributeValue());

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetTopicAttributes](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SetTopicAttributesCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
    AttributeName: "ATTRIBUTE_NAME", // ATTRIBUTE_NAME
    TopicArn: "TOPIC_ARN", // TOPIC_ARN
    AttributeValue: "NEW_ATTRIBUTE_VALUE", //NEW_ATTRIBUTE_VALUE
};

const run = async () => {
    try {
        const data = await snsClient.send(new SetTopicAttributesCommand(params));
        console.log("Success.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err.stack);
    }
};
```

```
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [SetTopicAttributes](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region'  => 'us-east-1',
    'version'  => '2010-03-31'
]);
$attribute = 'Policy | DisplayName | DeliveryPolicy';
$value     = 'First Topic';
$topic     = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->setTopicAttributes([
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn'      => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [SetTopicAttributes](#) in *AWS SDK for PHP API Reference*.

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

policy  = {
```

```
"Version":"2008-10-17",
"Id": "__default_policy_ID",
"Statement":[{
    "Sid": "__default_statement_ID",
    "Effect": "Allow",
    "Principal": {
        "AWS": "*"
    },
    "Action": ["SNS:Publish"],
    "Resource": "' + MY_TOPIC_ARN + ''",
    "Condition": {
        "ArnEquals": {
            "AWS:SourceArn": "' + MY_RESOURCE_ARN + ''"
        }
    }
}]
}'
# Replace us-west-2 with the AWS Region you're using for Amazon SNS.
sns = Aws::SNS::Resource.new(region: 'REGION')

# Get topic by ARN
topic = sns.topic()

# Add policy to topic
topic.set_attributes({
    attribute_name: "POLICY_NAME",
    attribute_value: policy
})
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [SetTopicAttributes](#) in [AWS SDK for Ruby API Reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Subscribe a Lambda function to receive notifications from an Amazon SNS topic using an AWS SDK

The following code examples show how to subscribe a Lambda function so it receives notifications from an Amazon SNS topic.

C++

### SDK for C++

```
/**
 * Subscribe an AWS Lambda endpoint to a topic - demonstrates how to initiate a
 * subscription to an Amazon SNS topic with delivery
 * to an AWS Lambda function.
 *
 * NOTE: You must first configure AWS Lambda to run this example.
 *       See https://docs.amazonaws.cn/en_us/lambda/latest/dg/with-sns-
example.html for more information.
 *
 * <protocol_value> set to "lambda" provides delivery of JSON-encoded message to an
 * AWS Lambda function
```

```
*           (see https://docs.aws.amazon.com/sns/latest/api/API_Subscribe.html for
available protocols).
* <topic_arn_value> can be obtained from run_list_topics executable and includes
the "arn:" prefix.
* <lambda_function_arn> is the ARN of an AWS Lambda function.
*/
int main(int argc, char ** argv)
{
    if (argc != 4)
    {
        std::cout << "Usage: subscribe_lamda <protocol_value=lambda> <topic_arn_value>"<< " <lambda_function_arn>" << std::endl;
        return 1;
    }

    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        Aws::SNS::SNSClient sns;
        Aws::String protocol = argv[1];
        Aws::String topic_arn = argv[2];
        Aws::String endpoint = argv[3];

        Aws::SNS::Model::SubscribeRequest s_req;
        s_req.SetTopicArn(topic_arn);
        s_req.SetProtocol(protocol);
        s_req.SetEndpoint(endpoint);

        auto s_out = sns.Subscribe(s_req);

        if (s_out.IsSuccess())
        {
            std::cout << "Subscribed successfully " << std::endl;
        }
        else
        {
            std::cout << "Error while subscribing " << s_out.GetError().GetMessage()
                << std::endl;
        }
    }

    Aws::ShutdownAPI(options);
    return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for C++ API Reference*.

## Java

### SDK for Java 2.x

```
public static String subLambda(SnsClient snsClient, String topicArn, String
lambdaArn) {

    try {

        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("lambda")
            .endpoint(lambdaArn)
```

```
        .returnSubscriptionArn(true)
        .topicArn(topicArn)
        .build();

    SubscribeResponse result = snsClient.subscribe(request);
    return result.subscriptionArn();

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return "";
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SubscribeCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
  Protocol: "lambda" /* required */,
  TopicArn: "TOPIC_ARN", //TOPIC_ARN
  Endpoint: "LAMBDA_FUNCTION_ARN", //LAMBDA_FUNCTION_ARN
};

const run = async () => {
  try {
    const data = await snsClient.send(new SubscribeCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Subscribe](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Subscribe a mobile application to an Amazon SNS topic using an AWS SDK

The following code examples show how to subscribe a mobile application endpoint so it receives notifications from an Amazon SNS topic.

C++

### SDK for C++

```
/**  
 * Subscribe an app endpoint to a topic - demonstrates how to initiate a  
 subscription to an Amazon SNS topic  
 * with delivery to a mobile app.  
 *  
 * NOTE: You must first create an endpoint by registering an app and device.  
 * See https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-devicetoken.html for more information.  
 *  
 * <protocol_value> set to "application" provides delivery of JSON-encoded message  
 to an EndpointArn  
 * for a mobile app and device (see https://docs.aws.amazon.com/sns/latest/api/API\_Subscribe.html for available protocols).  
 * <topic_arn_value> can be obtained from run_list_topics executable and includes  
 the "arn:" prefix.  
 * <mobile_endpoint_arn> is the EndpointArn of a mobile app and device.  
 */  
int main(int argc, char ** argv)  
{  
    if (argc != 4)  
    {  
        std::cout << "Usage: subscribe_app <protocol_value/application>  
<topic_arn_value>"  
              " <mobile_endpoint_arn>" << std::endl;  
        return 1;  
    }  
  
    Aws::SDKOptions options;  
    Aws::InitAPI(options);  
    {  
        Aws::SNS::SNSClient sns;  
        Aws::String protocol = argv[1];  
        Aws::String topic_arn = argv[2];  
        Aws::String endpoint = argv[3];  
  
        Aws::SNS::Model::SubscribeRequest s_req;  
        s_req.SetTopicArn(topic_arn);  
        s_req.SetProtocol(protocol);  
        s_req.SetEndpoint(endpoint);  
  
        auto s_out = sns.Subscribe(s_req);  
  
        if (s_out.IsSuccess())  
        {  
            std::cout << "Subscribed successfully " << std::endl;  
        }  
        else  
        {  
    }
```

```
        std::cout << "Error while subscribing " << s_out.GetError().GetMessage()
        << std::endl;
    }

Aws::ShutdownAPI(options);
return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for C++ API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SubscribeCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
  Protocol: "application" /* required */,
  TopicArn: "TOPIC_ARN", //TOPIC_ARN
  Endpoint: "MOBILE_ENDPOINT_ARN", // MOBILE_ENDPOINT_ARN
};

const run = async () => {
  try {
    const data = await snsClient.send(new SubscribeCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Subscribe](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Subscribe an HTTP endpoint to an Amazon SNS topic using an AWS SDK

The following code examples show how to subscribe an HTTP or HTTPS endpoint so it receives notifications from an Amazon SNS topic.

Java

### SDK for Java 2.x

```
public static void subHTTPS(SnsClient snsClient, String topicArn, String url ) {  
  
    try {  
        SubscribeRequest request = SubscribeRequest.builder()  
            .protocol("http")  
            .endpoint(url)  
            .returnSubscriptionArn(true)  
            .topicArn(topicArn)  
            .build();  
  
        SubscribeResponse result = snsClient.subscribe(request);  
        System.out.println("Subscription ARN is " + result.subscriptionArn() +  
"\n\n Status is " + result.sdkHttpResponse().statusCode());  
  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Java 2.x API Reference*.

PHP

### SDK for PHP

```
require 'vendor/autoload.php';  
  
use Aws\Sns\SnsClient;  
use Aws\Exception\AwsException;  
  
/**  
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation  
 * message.  
 *  
 * This code expects that you have AWS credentials set up per:  
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
 * guide_credentials.html  
 */  
  
$SnSclient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);
```

```
$protocol = 'https';
$endpoint = 'https://';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

## Subscribe an email address to an Amazon SNS topic using an AWS SDK

The following code examples show how to subscribe an email address to an Amazon SNS topic.

C++

### SDK for C++

```
/** 
 * Subscribe an email address endpoint to a topic - demonstrates how to initiate a
 * subscription to an Amazon SNS topic with delivery
 * to an email address.
 *
 * SNS will send a subscription confirmation email to the email address provided
 * which you need to confirm to
 * receive messages.
 *
 * <protocol_value> set to "email" provides delivery of message via SMTP (see
 * https://docs.aws.amazon.com/sns/latest/api/API_Subscribe.html for available
 * protocols).
 * <topic_arn_value> can be obtained from run_list_topics executable and includes
 * the "arn:" prefix.
 */

int main(int argc, char ** argv)
{
    if (argc != 4)
    {
        std::cout << "Usage: subscribe_email <protocol_value=email> <topic_arn_value>" 
                  " <email_address>" << std::endl;
        return 1;
    }
}
```

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSClient sns;
    Aws::String protocol = argv[1];
    Aws::String topic_arn = argv[2];
    Aws::String endpoint = argv[3];

    Aws::SNS::Model::SubscribeRequest s_req;
    s_req.SetTopicArn(topic_arn);
    s_req.SetProtocol(protocol);
    s_req.SetEndpoint(endpoint);

    auto s_out = sns.Subscribe(s_req);

    if (s_out.IsSuccess())
    {
        std::cout << "Subscribed successfully " << std::endl;
    }
    else
    {
        std::cout << "Error while subscribing " << s_out.GetError().GetMessage()
        << std::endl;
    }
}

Aws::ShutdownAPI(options);
return 0;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for C++ API Reference*.

## Go

### SDK for Go V2

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Go API Reference*.

## Java

### SDK for Java 2.x

```
public static void subEmail(SnsClient snsClient, String topicArn, String email)
{
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("email")
            .endpoint(email)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is " + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Java 2.x API Reference*.

## JavaScript

### SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create SNS service object.
const snsClient = new SNSClient({ region: REGION });
export { snsClient };
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import {SubscribeCommand} from "@aws-sdk/client-sns";
import {snsClient} from "./libs/snsClient.js";

// Set the parameters
const params = {
  Protocol: "email" /* required */,
  TopicArn: "TOPIC_ARN", //TOPIC_ARN
  Endpoint: "EMAIL_ADDRESS", //EMAIL_ADDRESS
};

const run = async () => {
  try {
    const data = await snsClient.send(new SubscribeCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Subscribe](#) in *AWS SDK for JavaScript API Reference*.

## PHP

### SDK for PHP

```
require 'vendor/autoload.php';
```

```
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 * message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for PHP API Reference*.

## Python

### SDK for Python (Boto3)

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def subscribe(topic, protocol, endpoint):
        """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
        subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.

        :param topic: The topic to subscribe to.
        :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
        
```

```
:param endpoint: The endpoint that receives messages, such as a phone
number
                           (in E.164 format) for SMS messages, or an email address
for
                           email messages.
:return: The newly added subscription.
"""
try:
    subscription = topic.subscribe(
        Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True)
    logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
topic.arn)
except ClientError:
    logger.exception(
        "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
topic.arn)
    raise
else:
    return subscription
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in *AWS SDK for Python (Boto3) API Reference*.

## Ruby

### SDK for Ruby

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def subscription_created?(sns_client, topic_arn, protocol, endpoint)

    sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
endpoint)

rescue StandardError => e
    puts "Error while creating the subscription: #{e.message}"
end

# Full example call:
def run_me

protocol = 'email'
endpoint = 'EMAIL_ADDRESS'
topic_arn = 'TOPIC_ARN'
region = 'REGION'

sns_client = Aws::SNS::Client.new(region: region)

puts "Creating the subscription."

if subscription_created?(sns_client, topic_arn, protocol, endpoint)
    puts 'The subscription was created.'
else
    puts 'The subscription was not created. Stopping program.'
    exit 1
end

run_me if $PROGRAM_NAME == __FILE__
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for Ruby Developer Guide](#).
- For API details, see [Subscribe](#) in [AWS SDK for Ruby API Reference](#).

Rust

### SDK for Rust

#### Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`, topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);
    Ok(())
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [Subscribe](#) in [AWS SDK for Rust API reference](#).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see [Using Amazon SNS with an AWS SDK \(p. 6\)](#).

# Amazon SNS security

This section provides information about Amazon SNS security, authentication and access control, and the Amazon SNS Access Policy Language.

## Topics

- [Data protection \(p. 343\)](#)
- [Identity and access management in Amazon SNS \(p. 364\)](#)
- [Logging and monitoring in Amazon SNS \(p. 387\)](#)
- [Compliance validation for Amazon SNS \(p. 395\)](#)
- [Resilience in Amazon SNS \(p. 395\)](#)
- [Infrastructure security in Amazon SNS \(p. 395\)](#)
- [Amazon SNS security best practices \(p. 396\)](#)

## Data protection

The AWS [shared responsibility model](#) applies to data protection in Amazon Simple Notification 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. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR blog post](#) on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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 \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Amazon SNS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form 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.

The following sections provide additional information about data protection in Amazon SNS.

## Topics

- [Data encryption \(p. 344\)](#)
- [Internetwork traffic privacy \(p. 352\)](#)

## Data encryption

Data protection refers to protecting data while in-transit (as it travels to and from Amazon SNS) and at rest (while it is stored on disks in Amazon SNS data centers). You can protect data in transit using Secure Sockets Layer (SSL) or client-side encryption. You can protect data at rest by requesting Amazon SNS to encrypt your messages before saving them to disk in its data centers and then decrypt them when the messages are received.

### Topics

- [Encryption at rest \(p. 344\)](#)
- [Key management \(p. 345\)](#)
- [Enabling server-side encryption \(SSE\) for an Amazon SNS topic \(p. 349\)](#)
- [Enabling server-side encryption \(SSE\) for an Amazon SNS topic with an encrypted Amazon SQS queue subscribed \(p. 350\)](#)

## Encryption at rest

Server-side encryption (SSE) lets you store sensitive data in encrypted topics. SSE protects the contents of messages in Amazon SNS topics using keys managed in AWS Key Management Service (AWS KMS).

For information about managing SSE using the AWS Management Console or the AWS SDK for Java (by setting the `KmsMasterKeyId` attribute using the `CreateTopic` and `SetTopicAttributes` API actions), see [Enabling server-side encryption \(SSE\) for an Amazon SNS topic \(p. 349\)](#). For information about creating encrypted topics using AWS CloudFormation (by setting the `KmsMasterKeyId` property using the `AWS::SNS::Topic` resource), see the [AWS CloudFormation User Guide](#).

SSE encrypts messages as soon as Amazon SNS receives them. The messages are stored in encrypted form and Amazon SNS decrypts messages only when they are sent.

### Important

All requests to topics with SSE enabled must use HTTPS and [Signature Version 4](#).

For information about compatibility of other services with encrypted topics, see your service documentation.

AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. When you use Amazon SNS with AWS KMS, the [data keys \(p. 345\)](#) that encrypt your message data are also encrypted and stored with the data they protect.

The following are benefits of using AWS KMS:

- You can create and manage [customer master keys \(CMKs\) \(p. 345\)](#) yourself.
- You can also use AWS-managed KMS keys for Amazon SNS, which are unique for each account and region.
- The AWS KMS security standards can help you meet encryption-related compliance requirements.

For more information, see [What is AWS Key Management Service?](#) in the [AWS Key Management Service Developer Guide](#).

### Topics

- [Encryption scope \(p. 345\)](#)
- [Key terms \(p. 345\)](#)

## Encryption scope

SSE encrypts the body of a message in an Amazon SNS topic.

SSE doesn't encrypt the following:

- Topic metadata (topic name and attributes)
- Message metadata (subject, message ID, timestamp, and attributes)
- Per-topic metrics

### Note

- A message is encrypted only if it is sent after the encryption of a topic is enabled. Amazon SNS doesn't encrypt backlogged messages.
- Any encrypted message remains encrypted even if the encryption of its topic is disabled.

## Key terms

The following key terms can help you better understand the functionality of SSE. For detailed descriptions, see the [Amazon Simple Notification Service API Reference](#).

### Data key

The data encryption key (DEK) responsible for encrypting the contents of Amazon SNS messages.

For more information, see [Data Keys](#) in the *AWS Key Management Service Developer Guide* and [Envelope Encryption](#) in the *AWS Encryption SDK Developer Guide*.

### Customer master key ID

The alias, alias ARN, key ID, or key ARN of an AWS managed customer master key (CMK) or a custom CMK—in your account or in another account. While the alias of the AWS managed CMK for Amazon SNS is always alias/aws/sns, the alias of a custom CMK can, for example, be alias/*MyAlias*. You can use these CMKs to protect the messages in Amazon SNS topics.

### Note

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SNS for a topic, AWS KMS creates the AWS managed CMK for Amazon SNS.
- Alternatively, the first time you use the Publish action on a topic with SSE enabled, AWS KMS creates the AWS managed CMK for Amazon SNS.

You can create CMKs, define the policies that control how CMKs can be used, and audit CMK usage using the **Customer managed keys** section of the AWS KMS console or the [CreateKey](#) AWS KMS action. For more information, see [Customer Master Keys \(CMKs\)](#) and [Creating Keys](#) in the *AWS Key Management Service Developer Guide*. For more examples of CMK identifiers, see [KeyId](#) in the *AWS Key Management Service API Reference*. For information about finding CMK identifiers, see [Find the Key ID and ARN](#) in the *AWS Key Management Service Developer Guide*.

### Important

There are additional charges for using AWS KMS. For more information, see [Estimating AWS KMS costs \(p. 346\)](#) and [AWS Key Management Service Pricing](#).

## Key management

The following sections provide information about working with keys managed in AWS Key Management Service (AWS KMS).

## Topics

- [Estimating AWS KMS costs \(p. 346\)](#)
- [Configuring AWS KMS permissions \(p. 346\)](#)
- [AWS KMS errors \(p. 348\)](#)

## Estimating AWS KMS costs

To predict costs and better understand your AWS bill, you might want to know how often Amazon SNS uses your customer master key (CMK).

### Note

Although the following formula can give you a very good idea of expected costs, actual costs might be higher because of the distributed nature of Amazon SNS.

To calculate the number of API requests ( $R$ ) *per topic*, use the following formula:

$$R = B / D * (2 * P)$$

$B$  is the billing period (in seconds).

$D$  is the data key reuse period (in seconds—Amazon SNS reuses a data key for up to 5 minutes).

$P$  is the number of publishing [principals](#) that send to the Amazon SNS topic.

The following are example calculations. For exact pricing information, see [AWS Key Management Service Pricing](#).

### Example 1: Calculating the number of AWS KMS API calls for 1 publisher and 1 topic

This example assumes the following:

- The billing period is January 1-31 (2,678,400 seconds).
- The data key reuse period is 5 minutes (300 seconds).
- There is 1 topic.
- There is 1 publishing principal.

$$2,678,400 / 300 * (2 * 1) = 17,856$$

### Example 2: Calculating the number of AWS KMS API calls for multiple publishers and 2 topics

This example assumes the following:

- The billing period is February 1-28 (2,419,200 seconds).
- The data key reuse period is 5 minutes (300 seconds).
- There are 2 topics.
- The first topic has 3 publishing principals.
- The second topic has 5 publishing principals.

$$(2,419,200 / 300 * (2 * 3)) + (2,419,200 / 300 * (2 * 5)) = 129,024$$

## Configuring AWS KMS permissions

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of topics and encryption and decryption of messages. For examples and more information about AWS

KMS permissions, see [AWS KMS API Permissions: Actions and Resources Reference](#) in the *AWS Key Management Service Developer Guide*.

**Note**

You can also manage permissions for KMS keys using IAM policies. For more information, see [Using IAM Policies with AWS KMS](#).

While you can configure global permissions to send to and receive from Amazon SNS, AWS KMS requires explicitly naming the full ARN of CMKs in specific regions in the Resource section of an IAM policy.

You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. To do this, name the principals that produce and consume encrypted messages in Amazon SNS as users in the CMK key policy.

Alternatively, you can specify the required AWS KMS actions and CMK ARN in an IAM policy assigned to the principals that publish and subscribe to receive encrypted messages in Amazon SNS. For more information, see [Managing Access to AWS KMS CMKs](#) in the *AWS Key Management Service Developer Guide*.

### Allow a user to send messages to a topic with SSE

The publisher must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions for the customer master key (CMK).

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "kms:GenerateDataKey",  
        "kms:Decrypt"  
      ],  
      "Resource": "arn:aws:kms:us-  
east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
    }, {  
      "Effect": "Allow",  
      "Action": [  
        "sns:Publish"  
      ],  
      "Resource": "arn:aws:sns:*:123456789012:MyTopic"  
    }]  
}
```

### Enable compatibility between event sources from AWS services and encrypted topics

Several AWS services publish events to Amazon SNS topics. To allow these event sources to work with encrypted topics, you must perform the following steps.

1. Use a customer managed CMK. For more information, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*.
2. To allow the AWS service to have the `kms:GenerateDataKey*` and `kms:Decrypt` permissions, add the following statement to the CMK policy.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "service.amazonaws.com"  
      },  
      "Action": [  
        "kms:GenerateDataKey*",  
        "kms:Decrypt"  
      ]  
    }]  
}
```

```

        ],
        "Resource": "*"
    }]
}
```

Event source	Service principal
<a href="#">Amazon CloudWatch</a>	cloudwatch.amazonaws.com
<a href="#">Amazon CloudWatch Events</a>	events.amazonaws.com
<a href="#">AWS CodeCommit</a>	codecommit.amazonaws.com
<a href="#">AWS CodeStar</a>	codestar-notifications.amazonaws.com
<a href="#">AWS Database Migration Service</a>	dms.amazonaws.com
<a href="#">AWS Directory Service</a>	ds.amazonaws.com
<a href="#">Amazon DynamoDB</a>	dynamodb.amazonaws.com
<a href="#">Amazon S3 Glacier</a>	glacier.amazonaws.com
<a href="#">Amazon Inspector</a>	inspector.amazonaws.com
<a href="#">Amazon Redshift</a>	redshift.amazonaws.com
<a href="#">Amazon Simple Email Service</a>	ses.amazonaws.com
<a href="#">Amazon Simple Storage Service</a>	s3.amazonaws.com
<a href="#">AWS Snowball</a>	importexport.amazonaws.com

#### Note

Some Amazon SNS event sources require you to provide an IAM role (rather than the service principal) in the AWS KMS key policy:

- [Amazon EC2 Auto Scaling](#)
- [Amazon Elastic Transcoder](#)
- [AWS CodePipeline](#)
- [AWS Config](#)
- [AWS Elastic Beanstalk](#)
- [AWS IoT](#)

3. [Enable SSE for your topic \(p. 349\)](#) using your CMK.
4. Provide the ARN of the encrypted topic to the event source.

## AWS KMS errors

When you work with Amazon SNS and AWS KMS, you might encounter errors. The following list describes the errors and possible troubleshooting solutions.

### KMSAccessDeniedException

The ciphertext references a key that doesn't exist or that you don't have access to.

HTTP Status Code: 400

### KMSDisabledException

The request was rejected because the specified CMK isn't enabled.

HTTP Status Code: 400

### KMSInvalidStateException

The request was rejected because the state of the specified resource isn't valid for this request. For more information, see [How Key State Affects Use of a Customer Master Key](#) in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

### KMSNotFoundException

The request was rejected because the specified entity or resource can't be found.

HTTP Status Code: 400

### KMSOptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

### KMSThrottlingException

The request was denied due to request throttling. For more information about throttling, see [Limits](#) in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

## Enabling server-side encryption (SSE) for an Amazon SNS topic

You can enable server-side encryption (SSE) for a topic to protect its data. For more information about using SSE, see [Encryption at rest \(p. 344\)](#).

**Important**

All requests to topics with SSE enabled must use HTTPS and [Signature Version 4](#).

This page shows how to enable, disable, and configure SSE for an existing Amazon SNS topic using the AWS Management Console.

### To enable server-side encryption (SSE) for an Amazon SNS topic using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic and choose **Actions**, **Edit**.
4. Expand the **Encryption** section and do the following:
  - a. Choose **Enable encryption**.
  - b. Specify the customer master key (CMK). For more information, see [Key terms \(p. 345\)](#).

For each CMK type, the **Description**, **Account**, and **CMK ARN** are displayed.

**Important**

If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms>ListAliases` and `kms>DescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SNS console.

Ask the owner of the CMK to grant you these permissions. For more information, see the [AWS KMS API Permissions: Actions and Resources Reference](#) in the *AWS Key Management Service Developer Guide*.

- The AWS managed CMK for Amazon SNS (**Default**) alias/`aws/sns` is selected by default.

**Note**

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SNS for a topic, AWS KMS creates the AWS managed CMK for Amazon SNS.
- Alternatively, the first time you use the `Publish` action on a topic with SSE enabled, AWS KMS creates the AWS managed CMK for Amazon SNS.
- To use a custom CMK from your AWS account, choose the **Customer master key (CMK)** field and then choose the custom CMK from the list.

**Note**

For instructions on creating custom CMKs, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*

- To use a custom CMK ARN from your AWS account or from another AWS account, enter it into the **Customer master key (CMK)** field.

5. Choose **Save changes**.

SSE is enabled for your topic and the [MyTopic](#) page is displayed.

The topic's **Encryption** status, **AWS Account**, **Customer master key (CMK)**, **CMK ARN**, and **Description** are displayed on the **Encryption** tab.

## Enabling server-side encryption (SSE) for an Amazon SNS topic with an encrypted Amazon SQS queue subscribed

You can enable server-side encryption (SSE) for a topic to protect its data. To allow Amazon SNS to send messages to encrypted Amazon SQS queues, the customer master key (CMK) associated with the Amazon SQS queue must have a policy statement that grants Amazon SNS service-principal access to the AWS KMS API actions `GenerateDataKey` and `Decrypt`. Because AWS managed CMKs don't support policy modifications, you must use a custom CMK. For more information about using SSE, see [Encryption at rest \(p. 344\)](#).

This page shows how you can enable SSE for an Amazon SNS topic to which an encrypted Amazon SQS queue is subscribed, using the AWS Management Console.

### Step 1: To create a custom CMK

1. Sign in to the [AWS KMS console](#) with a user that has at least the `AWSKeyManagementServicePowerUser` policy.
2. Choose **Create a key**.
3. On the **Add alias and description** page, enter an **Alias** for your key (for example, `MyCustomCMK`) and then choose **Next**.
4. On the **Add tags** page, choose **Next**.
5. On the **Define key administrative permissions** page, in the **Key administrators** section, choose an IAM role or an IAM user and then choose **Next**.
6. On the **Define key usage permissions** page, in the **This account** section, choose an IAM role or an IAM user and then choose **Next**.
7. On the **Review and edit key policy** page, add the following statement to the key policy, and then choose **Finish**.

```
{  
    "Sid": "Allow Amazon SNS to use this key",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "sns.amazonaws.com"  
    },  
    "Action": [  
        "kms:Decrypt",  
        "kms:GenerateDataKey"  
    ],  
    "Resource": "*"  
}
```

Your new custom CMK appears in the list of keys.

## Step 2: To create an encrypted Amazon SNS topic

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. Choose **Create topic**.
4. On the **Create new topic** page, for **Name**, enter a topic name (for example, `MyEncryptedTopic`) and then choose **Create topic**.
5. Expand the **Encryption** section and do the following:
  - a. Choose **Enable server-side encryption**.
  - b. Specify the customer master key (CMK). For more information, see [Key terms \(p. 345\)](#).

For each CMK type, the **Description**, **Account**, and **CMK ARN** are displayed.

**Important**

If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms>ListAliases` and `kmsDescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SNS console.

Ask the owner of the CMK to grant you these permissions. For more information, see the [AWS KMS API Permissions: Actions and Resources Reference](#) in the [AWS Key Management Service Developer Guide](#).

- c. For **Customer master key (CMK)**, choose `MyCustomCMK` which you created earlier (p. 350) and then choose **Enable server-side encryption**.
6. Choose **Save changes**.

SSE is enabled for your topic and the **MyTopic** page is displayed.

The topic's **Encryption** status, **AWS Account**, **Customer master key (CMK)**, **CMK ARN**, and **Description** are displayed on the **Encryption** tab.

Your new encrypted topic appears in the list of topics.

## Step 3: To create and subscribe encrypted Amazon SQS queues

1. Sign in to the [Amazon SQS console](#).
2. Choose **Create New Queue**.
3. On the **Create New Queue** page, do the following:
  - a. Enter a **Queue Name** (for example, `MyEncryptedQueue1`).

- b. Choose **Standard Queue**, and then choose **Configure Queue**.
  - c. Choose **Use SSE**.
  - d. For **AWS KMS Customer Master Key (CMK)**, choose **MyCustomCMK** which you created earlier (p. 350), and then choose **Create Queue**.
4. Repeat the process to create a second queue (for example, named **MyEncryptedQueue2**).

Your new encrypted queues appear in the list of queues.
  5. On the Amazon SQS console, choose **MyEncryptedQueue1** and **MyEncryptedQueue2** and then choose **Queue Actions, Subscribe Queues to SNS Topic**.
  6. In the **Subscribe to a Topic** dialog box, for **Choose a Topic** select **MyEncryptedTopic**, and then choose **Subscribe**.

Your encrypted queues' subscriptions to your encrypted topic are displayed in the **Topic Subscription Result** dialog box.
  7. Choose **OK**.

#### Step 4: To publish a message to your encrypted topic

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. From the list of topics, choose **MyEncryptedTopic** and then choose **Publish message**.
4. On the **Publish a message** page, do the following:
  - a. (Optional) In the **Message details** section, enter the **Subject** (for example, Testing message publishing).
  - b. In the **Message body** section, enter the message body (for example, My message body is encrypted at rest.).
  - c. Choose **Publish message**.

Your message is published to your subscribed encrypted queues.

#### Step 5: To verify message delivery

1. Sign in to the [Amazon SQS console](#).
2. From the list of queues, choose **MyEncryptedQueue1** and then choose **Queue Actions, View/Delete Messages**.
3. On the **View/Delete Messages in MyEncryptedQueue1** page, choose **Start polling for messages**.

The message [that you sent earlier \(p. 352\)](#) is displayed.
4. Choose **More Details** to view your message.
5. When you're finished, choose **Close**.
6. Repeat the process for **MyEncryptedQueue2**.

## Internetwork traffic privacy

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon SNS is a logical entity within a VPC that allows connectivity only to Amazon SNS. The VPC routes requests to Amazon SNS and routes responses back to the VPC. The following sections provide information about working with VPC endpoints and creating VPC endpoint policies.

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and Amazon SNS. With this connection, you can publish messages to your Amazon SNS topics without sending them through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. To connect your VPC to Amazon SNS, you define an *interface VPC endpoint*. This type of endpoint enables you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to Amazon SNS without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [Interface VPC Endpoints](#) in the [Amazon VPC User Guide](#).

The information in this section is for users of Amazon VPC. For more information, and to get started with creating a VPC, see [Getting Started With Amazon VPC](#) in the [Amazon VPC User Guide](#).

**Note**

VPC endpoints don't allow you to subscribe an Amazon SNS topic to a private IP address.

**Topics**

- [Creating an Amazon VPC endpoint for Amazon SNS \(p. 353\)](#)
- [Creating an Amazon VPC endpoint policy for Amazon SNS \(p. 354\)](#)
- [Publishing an Amazon SNS message from Amazon VPC \(p. 355\)](#)

## Creating an Amazon VPC endpoint for Amazon SNS

To publish messages to your Amazon SNS topics from an Amazon VPC, create an interface VPC endpoint. Then, you can publish messages to your topics while keeping the traffic within the network that you manage with the VPC.

Use the following information to create the endpoint and test the connection between your VPC and Amazon SNS. Or, for a walkthrough that helps you start from scratch, see [Publishing an Amazon SNS message from Amazon VPC \(p. 355\)](#).

### Creating the endpoint

You can create an Amazon SNS endpoint in your VPC using the AWS Management Console, the AWS CLI, an AWS SDK, the Amazon SNS API, or AWS CloudFormation.

For information about creating and configuring an endpoint using the Amazon VPC console or the AWS CLI, see [Creating an Interface Endpoint](#) in the [Amazon VPC User Guide](#).

**Note**

When you create an endpoint, specify Amazon SNS as the service that you want your VPC to connect to. In the Amazon VPC console, service names vary based on the region. For example, if you choose US East (N. Virginia), the service name is **com.amazonaws.us-east-1.sns**.

For information about creating and configuring an endpoint using AWS CloudFormation, see the [AWS::EC2::VPCEndpoint](#) resource in the [AWS CloudFormation User Guide](#).

### Testing the connection between your VPC and Amazon SNS

After you create an endpoint for Amazon SNS, you can publish messages from your VPC to your Amazon SNS topics. To test this connection, do the following:

1. Connect to an Amazon EC2 instance that resides in your VPC. For information about connecting, see [Connect to Your Linux Instance](#) or [Connecting to Your Windows Instance](#) in the Amazon EC2 documentation.

For example, to connect to a Linux instance using an SSH client, run the following command from a terminal:

```
$ ssh -i ec2-key-pair.pem ec2-user@instance-hostname
```

Where:

- *ec2-key-pair.pem* is the file that contains the key pair that Amazon EC2 provided when you created the instance.
  - *instance-hostname* is the public hostname of the instance. To get the hostname in the [Amazon EC2 console](#): Choose **Instances**, choose your instance, and find the value for **Public DNS (IPv4)**.
2. From your instance, use the Amazon SNS [publish](#) command with the AWS CLI. You can send a simple message to a topic with the following command:

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
```

Where:

- *aws-region* is the AWS Region that the topic is located in.
- *sns-topic-arn* is the Amazon Resource Name (ARN) of the topic. To get the ARN from the [Amazon SNS console](#): Choose **Topics**, find your topic, and find the value in the **ARN** column.

If the message is successfully received by Amazon SNS, the terminal prints a message ID, like the following:

```
{  
    "MessageId": "6c96dfff-0fdf-5b37-88d7-8cba910a8b64"  
}
```

## Creating an Amazon VPC endpoint policy for Amazon SNS

You can create a policy for Amazon VPC endpoints for Amazon SNS in which you specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following example VPC endpoint policy specifies that the IAM user `MyUser` is allowed to publish to the Amazon SNS topic `MyTopic`.

```
{  
    "Statement": [  
        {  
            "Action": ["sns:Publish"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",  
            "Principal": {  
                "AWS": "arn:aws:iam:123456789012:user/MyUser"  
            }  
        }]  
}
```

}

The following are denied:

- Other Amazon SNS API actions, such as `sns:Subscribe` and `sns:Unsubscribe`.
- Other IAM users and rules which attempt to use this VPC endpoint.
- MyUser publishing to a different Amazon SNS topic.

**Note**

The IAM user can still use other Amazon SNS API actions from *outside* the VPC.

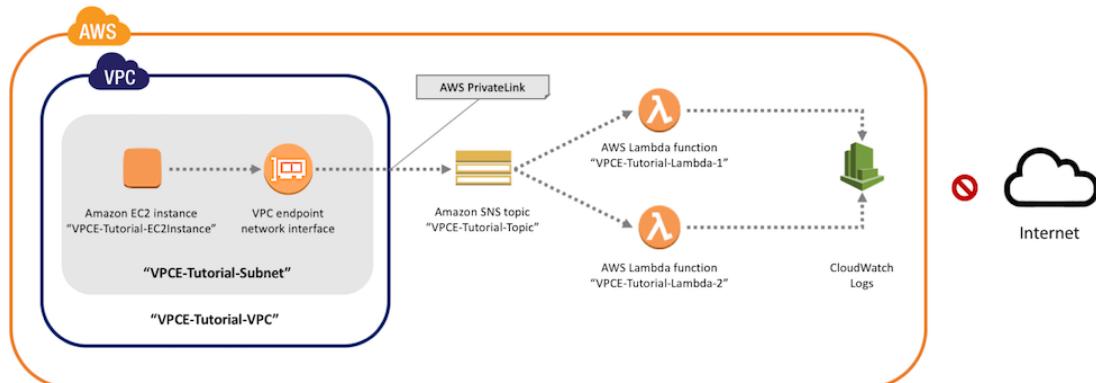
## Publishing an Amazon SNS message from Amazon VPC

This section describes how to publish to an Amazon SNS topic while keeping the messages secure in a private network. You publish a message from an Amazon EC2 instance that's hosted in Amazon Virtual Private Cloud (Amazon VPC). The message stays within the AWS network without traveling the public internet. By publishing messages privately from a VPC, you can improve the security of the traffic between your applications and Amazon SNS. This security is important when you publish personally identifiable information (PII) about your customers, or when your application is subject to market regulations. For example, publishing privately is helpful if you have a healthcare system that must comply with the Health Insurance Portability and Accountability Act (HIPAA), or a financial system that must comply with the Payment Card Industry Data Security Standard (PCI DSS).

The general steps are as follows:

- Use an AWS CloudFormation template to automatically create a temporary private network in your AWS account.
- Create a VPC endpoint that connects the VPC with Amazon SNS.
- Log in to an Amazon EC2 instance and publish a message privately to an Amazon SNS topic.
- Verify that the message was delivered successfully.
- Delete the resources that you created during this process so that they don't remain in your AWS account.

The following diagram depicts the private network that you create in your AWS account as you complete these steps:



This network consists of a VPC that contains an Amazon EC2 instance. The instance connects to Amazon SNS through an *interface VPC endpoint*. This type of endpoint connects to services that are powered by AWS PrivateLink. With this connection established, you can log in to the Amazon EC2 instance and publish messages to the Amazon SNS topic, even though the network is disconnected from the public

internet. The topic fans out the messages that it receives to two subscribing AWS Lambda functions. These functions log the messages that they receive in Amazon CloudWatch Logs.

It takes about 20 minutes to complete these steps.

### Topics

- [Before you begin \(p. 356\)](#)
- [Step 1: Create an Amazon EC2 key pair \(p. 356\)](#)
- [Step 2: Create the AWS resources \(p. 357\)](#)
- [Step 3: Confirm that your Amazon EC2 instance lacks internet access \(p. 358\)](#)
- [Step 4: Create an Amazon VPC endpoint for Amazon SNS \(p. 359\)](#)
- [Step 5: Publish a message to your Amazon SNS topic \(p. 353\)](#)
- [Step 6: Verify your message deliveries \(p. 361\)](#)
- [Step 7: Clean up \(p. 363\)](#)
- [Related resources \(p. 364\)](#)

## Before you begin

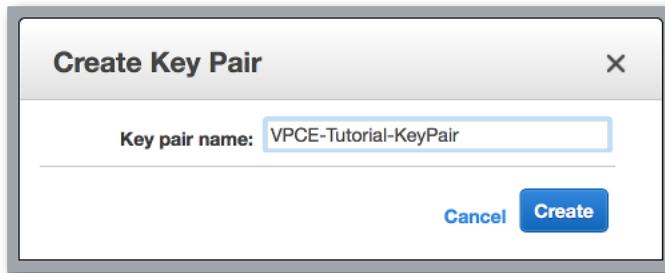
Before you start, you need an Amazon Web Services (AWS) account. When you sign up, your account is automatically signed up for all services in AWS, including Amazon SNS and Amazon VPC. If you haven't created an account already, go to <https://aws.amazon.com/>, and then choose **Create a Free Account**.

## Step 1: Create an Amazon EC2 key pair

A *key pair* is used to log in to an Amazon EC2 instance. It consists of a public key that's used to encrypt your login information, and a private key that's used to decrypt it. When you create a key pair, you download a copy of the private key. Later, you use the key pair to log in to an Amazon EC2 instance. To log in, you specify the name of the key pair, and you provide the private key.

### To create the key pair

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation menu on the left, find the **Network & Security** section. Then, choose **Key Pairs**.
3. Choose **Create Key Pair**.
4. In the **Create Key Pair** window, for **Key pair name**, type **VPCE-Tutorial-KeyPair**. Then, choose **Create**.



5. The private key file is automatically downloaded by your browser. Save it in a safe place. Amazon EC2 gives the file an extension of **.pem**.
6. (Optional) If you're using an SSH client on a Mac or Linux computer to connect to your instance, use the `chmod` command to set the permissions of your private key file so that only you can read it:
  - a. Open a terminal and navigate to the directory that contains the private key:

```
$ cd /filepath_to_private_key/
```

- b. Set the permissions using the following command:

```
$ chmod 400 VPCE-Tutorial-KeyPair.pem
```

## Step 2: Create the AWS resources

To set up the infrastructure, you use an AWS CloudFormation *template*. A template is a file that acts as a blueprint for building AWS resources, such as Amazon EC2 instances and Amazon SNS topics. The template for this process is provided on GitHub for you to download.

You provide the template to AWS CloudFormation, and AWS CloudFormation provisions the resources that you need as a *stack* in your AWS account. A stack is a collection of resources that you manage as a single unit. When you finish these steps, you can use AWS CloudFormation to delete all of the resources in the stack at once. These resources don't remain in your AWS account, unless you want them to.

The stack for this process includes the following resources:

- A VPC and the associated networking resources, including a subnet, a security group, an internet gateway, and a route table.
- An Amazon EC2 instance that's launched into the subnet in the VPC.
- An Amazon SNS topic.
- Two AWS Lambda functions. These functions receive messages that are published to the Amazon SNS topic, and they log events in CloudWatch Logs.
- Amazon CloudWatch metrics and logs.
- An IAM role that allows the Amazon EC2 instance to use Amazon SNS, and an IAM role that allows the Lambda functions to write to CloudWatch logs.

### To create the AWS resources

1. Download the [template file](#) from the GitHub website.
2. Sign in to the [AWS CloudFormation console](#).
3. Choose **Create Stack**.
4. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.
5. On the **Specify Details** page, specify stack and key names:
  - a. For **Stack name**, type **VPCE-Tutorial-Stack**.
  - b. For **KeyName**, choose **VPCE-Tutorial-KeyPair**.
  - c. For **SSHLocation**, keep the default value of **0.0.0.0/0**.

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

KeyName	<input type="text" value="VPCE-Tutorial-KeyPair"/>
Name of an existing EC2 KeyPair to enable SSH access to the instance	
SSHLlocation	<input type="text" value="0.0.0.0/0"/>
The IP address range that can be used to SSH to the EC2 instance	

- d. Choose **Next**.
6. On the **Options** page, keep all of the default values, and choose **Next**.
7. On the **Review** page, verify the stack details.
8. Under **Capabilities**, acknowledge that AWS CloudFormation might create IAM resources with custom names.
9. Choose **Create**.

The AWS CloudFormation console opens the **Stacks** page. The VPCE-Tutorial-Stack has a status of **CREATE\_IN\_PROGRESS**. In a few minutes, after the creation process completes, the status changes to **CREATE\_COMPLETE**.

Create Stack		Actions	Design template
Filter: Active ▾		By Stack Name	
Stack Name	Created Time	Status	Description
VPCE-Tutorial-Stack	2018-05-18 16:38:06 UTC-0700	CREATE_COMPLETE	CloudFormation Template for SNS VPC Endpoints Tutorial

### Tip

Choose the **Refresh** button to see the latest stack status.

## Step 3: Confirm that your Amazon EC2 instance lacks internet access

The Amazon EC2 instance that was launched in your VPC in the previous step lacks internet access. It disallows outbound traffic, and it's unable to publish messages to Amazon SNS. Verify this by logging in to the instance. Then, attempt to connect to a public endpoint, and attempt to message Amazon SNS.

At this point, the publish attempt fails. In a later step, after you create a VPC endpoint for Amazon SNS, your publish attempt succeeds.

### To connect to your Amazon EC2 instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation menu on the left, find the **Instances** section. Then, choose **Instances**.
3. In the list of instances, select **VPCE-Tutorial-EC2Instance**.
4. Copy the hostname that's provided in the **Public DNS (IPv4)** column.

Launch Instance		Connect	Actions		
Filter by tags and attributes or search by keyword					
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
VPCE-Tutorial-EC2Instance	i-0011111111111111	t2.micro	us-east-1c	running	2/2 checks ... None

5. Open a terminal. From the directory that contains the key pair, connect to the instance using the following command, where `instance-hostname` is the hostname that you copied from the Amazon EC2 console:

```
$ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
```

#### To verify that the instance lacks internet connectivity

- In your terminal, attempt to connect to any public endpoint, such as amazon.com:

```
$ ping amazon.com
```

Because the connection attempt fails, you can cancel at any time (Ctrl + C on Windows or Command + C on macOS).

#### To verify that the instance lacks connectivity to Amazon SNS

1. Sign in to the [Amazon SNS console](#).
2. In the navigation menu on the left, choose **Topics**.
3. On the **Topics** page, copy the Amazon Resource Name (ARN) for the topic **VPCE-Tutorial-Topic**.
4. In your terminal, attempt to publish a message to the topic:

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
```

Because the publish attempt fails, you can cancel at any time.

### Step 4: Create an Amazon VPC endpoint for Amazon SNS

To connect the VPC to Amazon SNS, you define an interface VPC endpoint. After you add the endpoint, you can log in to the Amazon EC2 instance in your VPC, and from there you can use the Amazon SNS API. You can publish messages to the topic, and the messages are published privately. They stay within the AWS network, and they don't travel the public internet.

#### Note

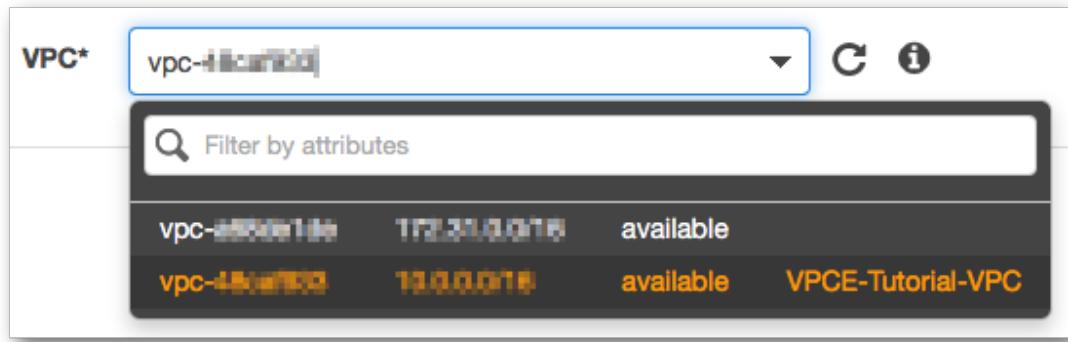
The instance still lacks access to other AWS services and endpoints on the internet.

#### To create the endpoint

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation menu on the left, choose **Endpoints**.
3. Choose **Create Endpoint**.
4. On the **Create Endpoint** page, for **Service category**, keep the default choice of **AWS services**.
5. For **Service Name**, choose the service name for Amazon SNS.

The service names vary based on the chosen region. For example, if you chose US East (N. Virginia), the service name is **com.amazonaws.us-east-1.sns**.

6. For **VPC**, choose the VPC that has the name **VPCE-Tutorial-VPC**.



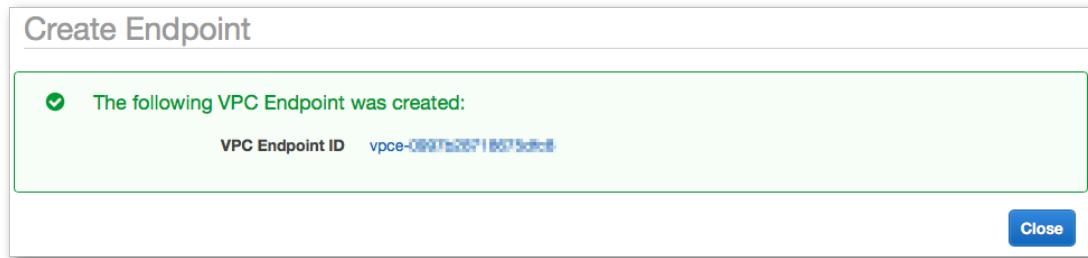
- For **Subnets**, choose the subnet that has *VPCE-Tutorial-Subnet* in the subnet ID.

Subnets	subnet-00000000	i
	Availability Zone	Subnet ID
<input checked="" type="checkbox"/>	us-east-1a	subnet-00000000 (VPCE-Tutorial-Subnet)
<input type="checkbox"/>	us-east-1b	No subnet available
<input type="checkbox"/>	us-east-1c	No subnet available
<input type="checkbox"/>	us-east-1d	No subnet available
<input type="checkbox"/>	us-east-1e	No subnet available
<input type="checkbox"/>	us-east-1f	No subnet available

- For **Enable Private DNS Name**, select **Enable for this endpoint**.
- For **Security group**, choose **Select security group**, and choose **VPCE-Tutorial-SecurityGroup**.

Security group	sg-00000000	Create a new security group
<a href="#">Select security groups ▾</a>		
Filter by attributes or search by keyword		
Name	Group ID	Group Name
-	sg-00000000	default
<input checked="" type="checkbox"/>	sg-11111111	Tutorial Secu...

- Choose **Create endpoint**. The Amazon VPC console confirms that a VPC endpoint was created.



11. Choose **Close**.

The Amazon VPC console opens the **Endpoints** page. The new endpoint has a status of **pending**. In a few minutes, after the creation process completes, the status changes to **available**.

Create Endpoint		Actions		
Filter by attributes or search by keyword				
Endpoint ID	VPC ID	Service name	Endpoint type	Status
vpce-0997b26716575d0f	vpc-0997b267   V...	com.amazonaws.us-east-1.sns	Interface	available

## Step 5: Publish a message to your Amazon SNS topic

Now that your VPC includes an endpoint for Amazon SNS, you can log in to the Amazon EC2 instance and publish messages to the topic.

### To publish a message

1. If your terminal is no longer connected to your Amazon EC2 instance, connect again:

```
$ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
```

2. Run the same command that you did previously to publish a message to your Amazon SNS topic. This time, the publish attempt succeeds, and Amazon SNS returns a message ID:

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"  
{  
    "MessageId": "5b111270-d169-5be6-9042-410dfc9e86de"  
}
```

## Step 6: Verify your message deliveries

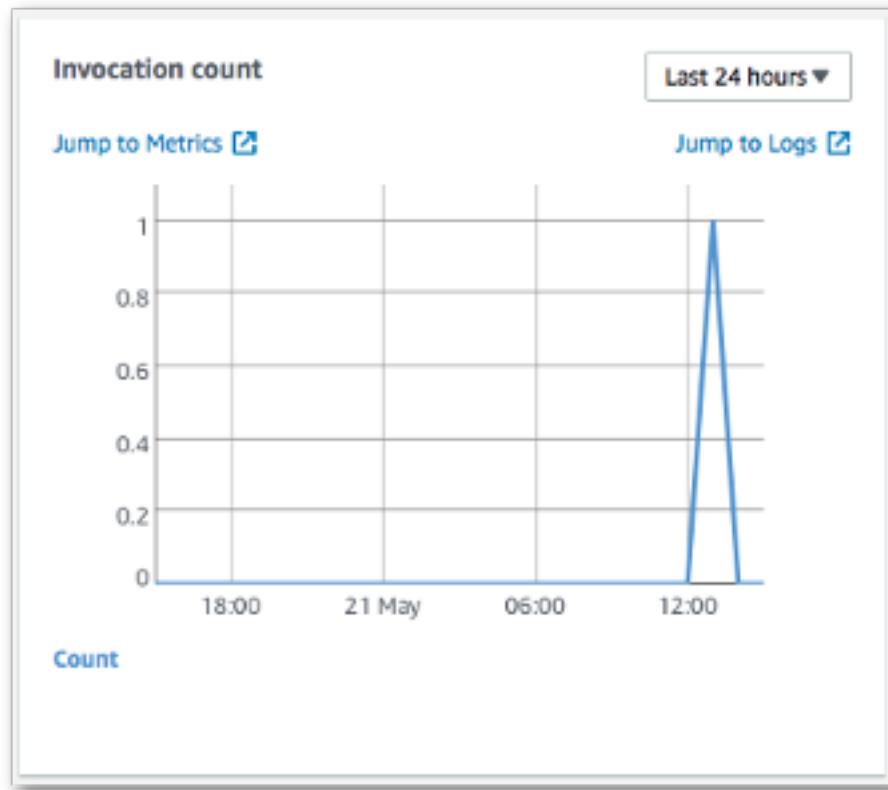
When the Amazon SNS topic receives a message, it fans out the message by sending it to the two subscribing Lambda functions. When these functions receive the message, they log the event to CloudWatch logs. To verify that your message delivery succeeded, check that the functions were invoked, and check that the CloudWatch logs were updated.

### To verify that the Lambda functions were invoked

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. On the **Functions** page, choose **VPCE-Tutorial-Lambda-1**.
3. Choose **Monitoring**.

4. Check the **Invocation count** graph. This graph shows the number of times that the Lambda function has been run.

The invocation count matches the number of times you published a message to the topic.



#### To verify that the CloudWatch logs were updated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation menu on the left, choose **Logs**.
3. Check the logs that were written by the Lambda functions:
  - a. Choose the **/aws/lambda/VPCE-Tutorial-Lambda-1/** log group.
  - b. Choose the log stream.
  - c. Check that the log includes the entry `From SNS: Hello.`

The screenshot shows a log viewer interface with a 'Filter events' bar at the top. Below it is a table with two columns: 'Time (UTC +00:00)' and 'Message'. A date filter '2018-05-21' is applied. The log entries are:

Time (UTC +00:00)	Message
2018-05-21 20:27:35	Loading function
2018-05-21 20:27:35	From SNS: Hello
	From SNS: Hello
2018-05-21 20:27:35	START RequestId: 654321
2018-05-21 20:27:35	END RequestId: 654321
2018-05-21 20:27:35	REPORT RequestId: 654321

- d. Choose **Log Groups** at the top of the console to return the **Log Groups** page. Then, repeat the preceding steps for the /aws/lambda/VPCE-Tutorial-Lambda-2/ log group.

Congratulations! By adding an endpoint for Amazon SNS to a VPC, you were able to publish a message to a topic from within the network that's managed by the VPC. The message was published privately without being exposed to the public internet.

## Step 7: Clean up

Unless you want to retain the resources that you created, you can delete them now. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

First, delete your VPC endpoint using the Amazon VPC console. Then, delete the other resources that you created by deleting the stack in the AWS CloudFormation console. When you delete a stack, AWS CloudFormation removes the stack's resources from your AWS account.

### To delete your VPC endpoint

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation menu on the left, choose **Endpoints**.
3. Select the endpoint that you created.
4. Choose **Actions**, and then choose **Delete Endpoint**.
5. In the **Delete Endpoint** window, choose **Yes, Delete**.

The endpoint status changes to **deleting**. When the deletion completes, the endpoint is removed from the page.

### To delete your AWS CloudFormation stack

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.

2. Select the stack **VPCE-Tutorial-Stack**.
3. Choose **Actions**, and then choose **Delete Stack**.
4. In the **Delete Stack** window, choose **Yes, Delete**.

The stack status changes to **DELETE\_IN\_PROGRESS**. When the deletion completes, the stack is removed from the page.

## Related resources

For more information, see the following resources.

- [AWS Security Blog: Securing messages published to Amazon SNS with AWS PrivateLink](#)
- [What Is Amazon VPC?](#)
- [VPC Endpoints](#)
- [What Is Amazon EC2?](#)
- [AWS CloudFormation Concepts](#)

# Identity and access management in Amazon SNS

Access to Amazon SNS requires credentials that AWS can use to authenticate your requests. These credentials must have permissions to access AWS resources, such as an Amazon SNS topics and messages. The following sections provide details on how you can use [AWS Identity and Access Management \(IAM\)](#) and Amazon SNS to help secure your resources by controlling access to them.

## Topics

- [Authentication \(p. 364\)](#)
- [Access control \(p. 365\)](#)
- [Overview of managing access in Amazon SNS \(p. 366\)](#)
- [Using identity-based policies with Amazon SNS \(p. 380\)](#)
- [Using temporary security credentials with Amazon SNS \(p. 385\)](#)
- [Amazon SNS API permissions: Actions and resources reference \(p. 385\)](#)

## Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An [IAM user](#) is an identity within your AWS account that has specific custom permissions (for example, permissions to create a topic in Amazon SNS). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon SNS supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the [AWS General Reference](#).

- **IAM role** – An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:
  - **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
  - **AWS service access** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

## Access control

Amazon SNS has its own resource-based permissions system that uses policies written in the same language used for AWS Identity and Access Management (IAM) policies. This means that you can achieve similar things with Amazon SNS policies and IAM policies.

### Note

It is important to understand that all AWS accounts can delegate their permissions to users under their accounts. Cross-account access allows you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, see [Enabling Cross-Account Access](#) in the *IAM User Guide*.

## Overview of managing access in Amazon SNS

This section describes basic concepts you need to understand to use the access policy language to write policies. It also describes the general process for how access control works with the access policy language, and how policies are evaluated.

### Topics

- [When to use access control \(p. 366\)](#)
- [Key concepts \(p. 366\)](#)
- [Architectural overview \(p. 369\)](#)
- [Using the Access Policy Language \(p. 370\)](#)
- [Evaluation logic \(p. 371\)](#)
- [Example cases for Amazon SNS access control \(p. 374\)](#)

## When to use access control

You have a great deal of flexibility in how you grant or deny access to a resource. However, the typical use cases are fairly simple:

- You want to grant another AWS account a particular type of topic action (for example, Publish). For more information, see [Grant AWS account access to a topic \(p. 375\)](#).
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see [Limit subscriptions to HTTPS \(p. 375\)](#).
- You want to allow Amazon SNS to publish messages to your Amazon SQS queue. For more information, see [Publish messages to an Amazon SQS queue \(p. 376\)](#).

## Key concepts

The following sections describe the concepts you need to understand to use the access policy language. They're presented in a logical order, with the first terms you need to know at the top of the list.

### Topics

- [Permission \(p. 367\)](#)
- [Statement \(p. 367\)](#)
- [Policy \(p. 367\)](#)
- [Issuer \(p. 367\)](#)
- [Principal \(p. 367\)](#)
- [Action \(p. 367\)](#)
- [Resource \(p. 368\)](#)
- [Conditions and keys \(p. 368\)](#)
- [Requester \(p. 368\)](#)
- [Evaluation \(p. 368\)](#)
- [Effect \(p. 368\)](#)
- [Default deny \(p. 368\)](#)
- [Allow \(p. 369\)](#)
- [Explicit deny \(p. 369\)](#)

## Permission

A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, Jane (A) has permission to *publish* (B) to TopicA (C) as long as *she uses the HTTP protocol* (D). Whenever Jane publishes to TopicA, the service checks to see if she has permission and if the request satisfies the conditions set forth in the permission.

## Statement

A *statement* is the formal description of a single permission, written in the access policy language. You always write a statement as part of a broader container document known as a *policy* (see the next concept).

## Policy

A *policy* is a document (written in the access policy language) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can subscribe using the email protocol, and another that states that Bob cannot publish to TopicA. As shown in the following figure, an equivalent scenario would be to have two policies, one that states that Jane can subscribe using the email protocol, and another that states that Bob cannot publish to TopicA.



## Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

## Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (that is, you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address.

## Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. For example, Jane sends a request to Amazon SNS with Action=Subscribe. You can specify one or multiple actions in a policy.

## Resource

The *resource* is the object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies."

## Conditions and keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (for example, the request must arrive before a specific day)
- IP address (for example, the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called `aws:CurrentTime` and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The AWS service itself (for example, Amazon SQS or Amazon SNS) might also define service-specific keys. For more information, see [Amazon SNS API permissions: Actions and resources reference \(p. 385\)](#).

## Requester

The *requester* is the person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Will you allow me to do B to C where D applies?"

## Evaluation

*Evaluation* is the process the AWS service uses to determine if an incoming request should be denied or allowed based on the applicable policies. For information about the evaluation logic, see [Evaluation logic \(p. 371\)](#).

## Effect

The *effect* is the result that you want a policy statement to return at evaluation time. You specify this value when you write the statements in a policy, and the possible values are *deny* and *allow*.

For example, you could write a policy that has a statement that *denies* all requests that come from Antarctica (`effect=deny` grants that the request uses an IP address allocated to Antarctica). Alternately, you could write a policy that has a statement that *allows* all requests that *don't* come from Antarctica (`effect=allow`, grants that the request doesn't come from Antarctica). Although the two statements sound like they do the same thing, in the access policy language logic, they are different. For more information, see [Evaluation logic \(p. 371\)](#).

Although there are only two possible values you can specify for the effect (allow or deny), there can be three different results at policy evaluation time: *default deny*, *allow*, or *explicit deny*. For more information, see the following concepts and [Evaluation logic \(p. 371\)](#).

## Default deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

## Allow

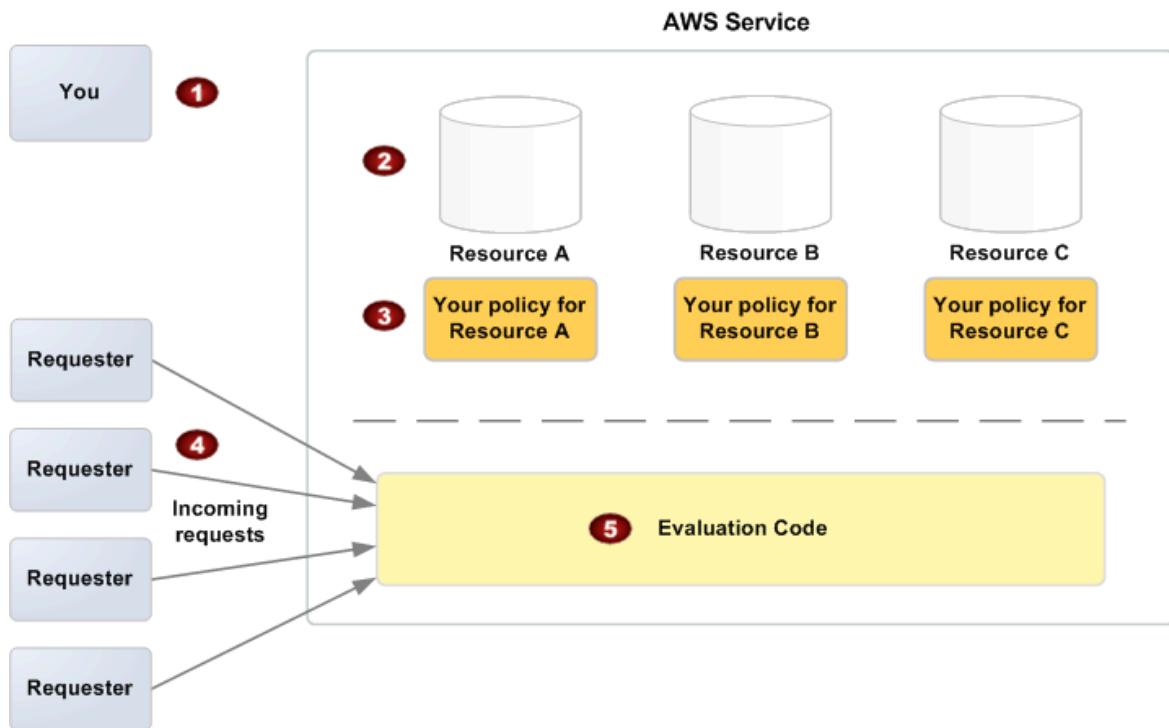
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

## Explicit deny

An *explicit deny* results from a statement that has effect=deny, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

## Architectural overview

The following figure and table describe the main components that interact to provide access control for your resources.

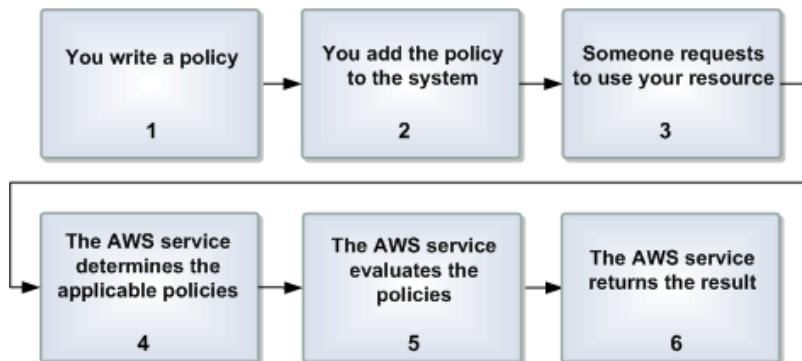


1	You, the resource owner.
2	Your resources (contained within the AWS service; for example, Amazon SQS queues).
3	Your policies.  Typically you have one policy per resource, although you could have multiple. The AWS service itself provides an API you use to upload and manage your policies.
4	Requesters and their incoming requests to the AWS service.
5	The access policy language evaluation code.

This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see [Evaluation logic \(p. 371\)](#).

## Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.



### Process for using access control with the Access Policy Language

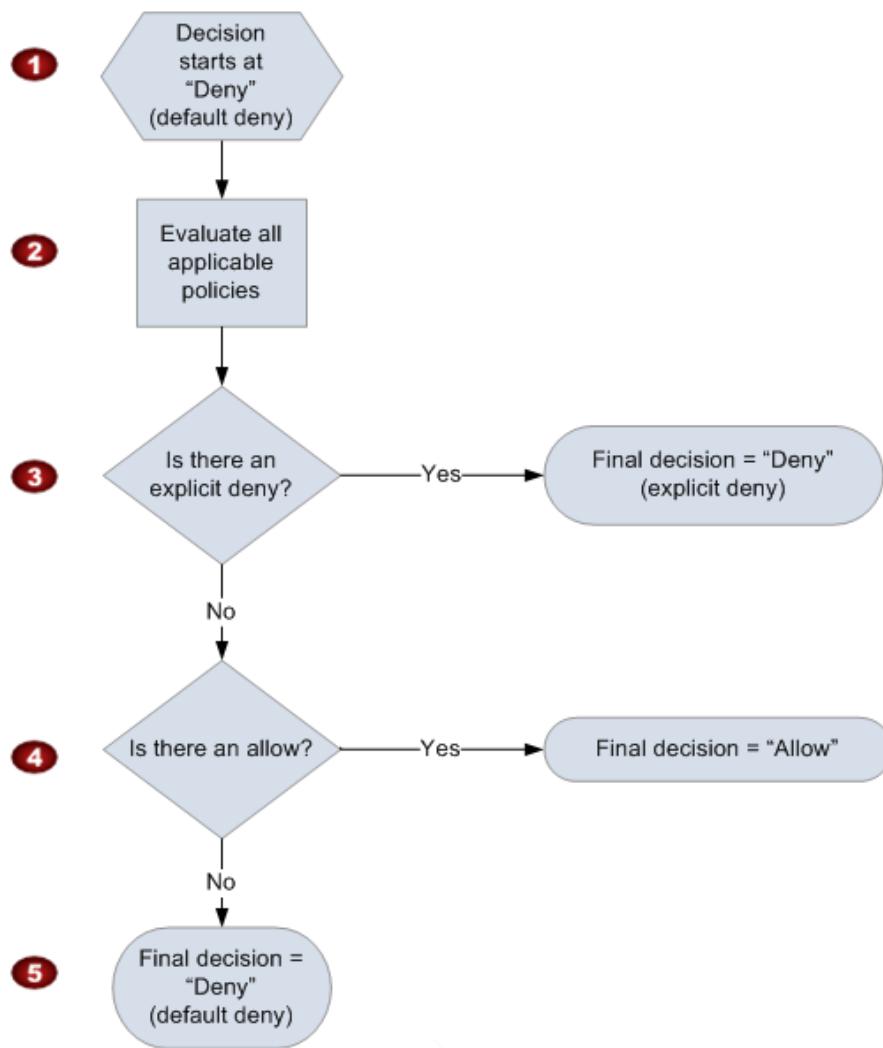
1	You write a policy for your resource.  For example, you write a policy to specify permissions for your Amazon SNS topics.
2	You upload your policy to AWS.  The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SNS SetTopicAttributes action to upload a policy for a particular Amazon SNS topic.
3	Someone sends a request to use your resource.  For example, a user sends a request to Amazon SNS to use one of your topics.
4	The AWS service determines which policies are applicable to the request.  For example, Amazon SNS looks at all the available Amazon SNS policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies.  For example, Amazon SNS evaluates the policies and determines if the requester is allowed to use your topic or not. For information about the decision logic, see <a href="#">Evaluation logic (p. 371)</a> .
6	The AWS service either denies the request or continues to process it.  For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

## Evaluation logic

The goal at evaluation time is to decide whether a grant request should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.



1	The decision starts with a default deny.
2	The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions).

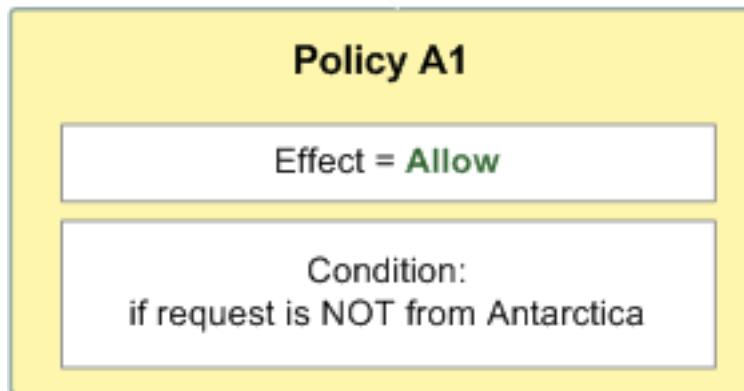
	The order in which the enforcement code evaluates the policies is not important.
3	<p>In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request.</p> <p>If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see <a href="#">Explicit deny (p. 369)</a>).</p>
4	<p>If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request.</p> <p>If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).</p>
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a <i>default deny</i> (for more information, see <a href="#">Default deny (p. 368)</a> )).

## The interplay of explicit and default denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if a user requests to use Amazon SNS, but the policy on the topic doesn't refer to the user's AWS account at all, then that policy results in a default deny.

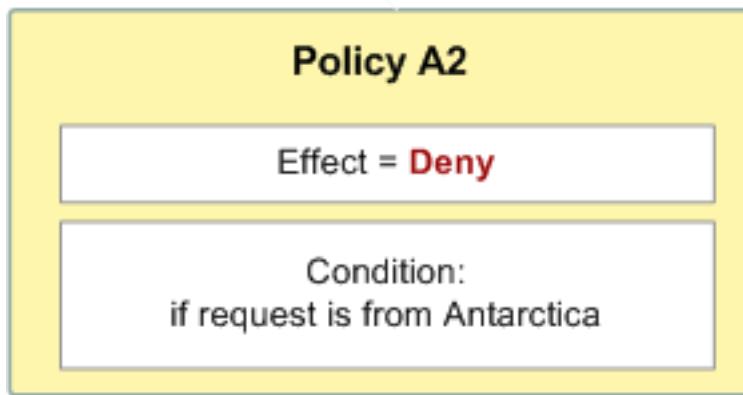
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met, and so the default result in that case is a default deny.

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



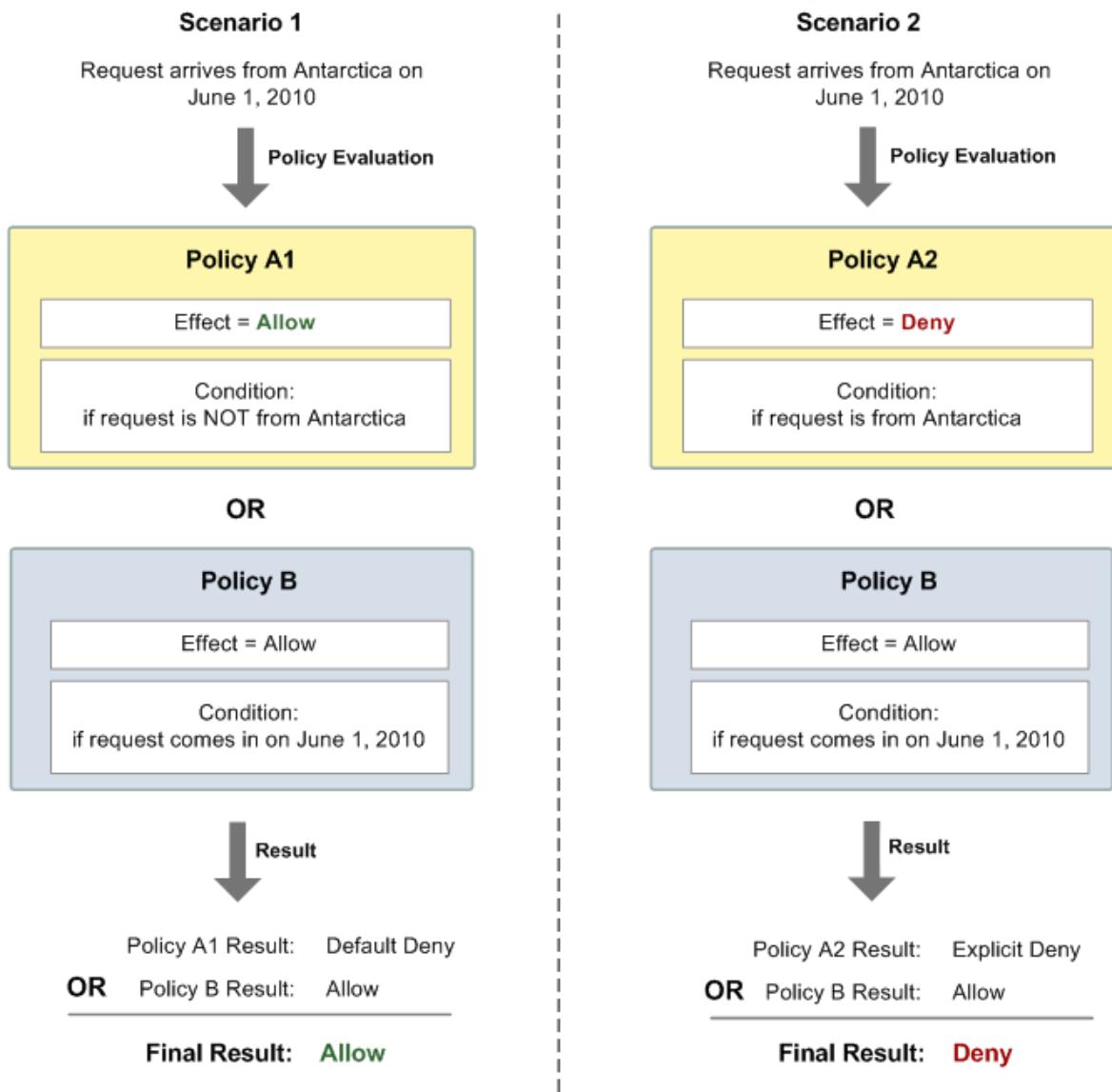
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.



In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy A2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

## Example cases for Amazon SNS access control

### Topics

- [Grant AWS account access to a topic \(p. 375\)](#)
- [Limit subscriptions to HTTPS \(p. 375\)](#)
- [Publish messages to an Amazon SQS queue \(p. 376\)](#)

- [Allow Amazon S3 event notifications to publish to a topic \(p. 376\)](#)
- [Allow Amazon SES to publish to a topic that is owned by another account \(p. 377\)](#)
- [aws:SourceAccount versus aws:SourceOwner \(p. 378\)](#)
- [Allow accounts in an organization in AWS Organizations to publish to a topic in a different account \(p. 379\)](#)
- [Allow any CloudWatch alarm to publish to a topic in a different account \(p. 379\)](#)
- [Restrict publication to an Amazon SNS topic only from a specific VPC endpoint \(p. 379\)](#)

This section describes a few examples of typical use cases for access control.

## Grant AWS account access to a topic

Let's say you have a topic in the Amazon SNS system. In the simplest case, you want to allow one or more AWS accounts access to a specific topic action (for example, Publish).

You can do this using the Amazon SNS API action `AddPermission`. It takes a topic, a list of AWS account IDs, a list of actions, and a label, and automatically creates a new statement in the topic's access control policy. In this case, you don't write a policy yourself, because Amazon SNS automatically generates the new policy statement for you. You can remove the policy statement later by calling `RemovePermission` with its label.

For example, if you called `AddPermission` on the topic `arn:aws:sns:us-east-2:444455556666:MyTopic`, with AWS account ID `1111-2222-3333`, the `Publish` action, and the label `grant-1234-publish`, Amazon SNS would generate and insert the following access control policy statement:

```
{  
  "Statement": [{  
    "Sid": "grant-1234-publish",  
    "Effect": "Allow",  
    "Principal": {  
      "AWS": "111122223333"  
    },  
    "Action": [ "sns:Publish" ],  
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic"  
  }]  
}
```

Once this statement is added, the user with AWS account `1111-2222-3333` can publish messages to the topic.

## Limit subscriptions to HTTPS

In the following example, you limit the notification delivery protocol to HTTPS.

You need to know how to write your own policy for the topic because the Amazon SNS `AddPermission` action doesn't let you specify a protocol restriction when granting someone access to your topic. In this case, you would write your own policy, and then use the `SetTopicAttributes` action to set the topic's `Policy` attribute to your new policy.

The following example of a full policy grants the AWS account ID `1111-2222-3333` the ability to subscribe to notifications from a topic.

```
{  
  "Statement": [{  
    "Sid": "Statement1",  
    "Effect": "Allow",  
    "Principal": {  
      "AWS": "111122223333"  
    },  
    "Action": [ "sns:Subscribe" ],  
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",  
    "Condition": {  
      "StringEquals": {  
        "aws:sns:Protocol": "https"  
      }  
    }  
  }]  
}
```

```
        },
        "Action": [ "sns:Subscribe" ],
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "StringEquals": {
                "sns:Protocol": "https"
            }
        }
    }]
}
```

## Publish messages to an Amazon SQS queue

In this use case, you want to publish messages from your topic to your Amazon SQS queue. Like Amazon SNS, Amazon SQS uses Amazon's access control policy language. To allow Amazon SNS to send messages, you'll need to use the Amazon SQS action `SetQueueAttributes` to set a policy on the queue.

Again, you'll need to know how to write your own policy because the Amazon SQS `AddPermission` action doesn't create policy statements with conditions.

### Note

The example presented below is an Amazon SQS policy (controlling access to your queue), not an Amazon SNS policy (controlling access to your topic). The actions are Amazon SQS actions, and the resource is the Amazon Resource Name (ARN) of the queue. You can determine the queue's ARN by retrieving the queue's `QueueArn` attribute with the `GetQueueAttributes` action.

```
{
    "Statement": [
        {
            "Sid": "Allow-SNS-SendMessage",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": [ "sns:SendMessage" ],
            "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:sqs:us-east-2:444455556666:MyQueue"
                }
            }
        ]
    }
}
```

This policy uses the `aws:SourceArn` condition to restrict access to the queue based on the source of the message being sent to the queue. You can use this type of policy to allow Amazon SNS to send messages to your queue only if the messages are coming from one of your own topics. In this case, you specify a particular one of your topics, whose ARN is `arn:aws:sns:us-east-2:444455556666:MyTopic`.

The preceding policy is an example of the Amazon SQS policy you could write and add to a specific queue. It would grant access to Amazon SNS and other AWS services. Amazon SNS grants a default policy to all newly created topics. The default policy grants access to your topic to all other AWS services. This default policy uses an `aws:SourceArn` condition to ensure that AWS services access your topic only on behalf of AWS resources you own.

## Allow Amazon S3 event notifications to publish to a topic

In this case, you want to configure a topic's policy so that another AWS account's Amazon S3 bucket can publish to your topic. For more information about publishing notifications from Amazon S3, go to [Setting Up Notifications of Bucket Events](#).

This example assumes that you write your own policy and then use the `SetTopicAttributes` action to set the topic's `Policy` attribute to your new policy.

The following example statement uses the `SourceAccount` condition to ensure that only the Amazon S3 owner account can access the topic. In this example, the topic owner is 111122223333 and the Amazon S3 owner is 444455556666. The example states that any Amazon S3 bucket owned by 444455556666 is allowed to publish to `MyTopic`.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sns:Publish",  
            "Resource": "arn:aws:sns:us-east-2:111122223333:MyTopic",  
            "Condition": {  
                "StringEquals": {  
                    "AWS:SourceAccount": "444455556666"  
                }  
            }  
        }  
    ]  
}
```

When publishing events to Amazon SNS, the following services support `aws:SourceAccount`:

- Amazon API Gateway
- Amazon CloudWatch
- Amazon DevOps Guru
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon GameLift
- Amazon Pinpoint SMS and Voice API
- Amazon RDS
- Amazon Redshift
- Amazon Simple Storage Service
- Amazon S3 Glacier
- Amazon SES
- AWS CodeCommit
- AWS Directory Service
- AWS Lambda
- AWS Systems Manager Incident Manager

## Allow Amazon SES to publish to a topic that is owned by another account

You can allow another AWS service to publish to a topic that is owned by another AWS account. Suppose that you signed into the 111122223333 account, opened Amazon SES, and created an email. To publish notifications about this email to a Amazon SNS topic that the 444455556666 account owns, you'd create a policy like the following. To do so, you need to provide information about the principal (the other service) and each resource's ownership. The `Resource` statement provides the topic ARN, which includes the account ID of the topic owner, 444455556666. The "`aws:SourceOwner`": "111122223333" statement specifies that your account owns the email.

```
{  
    "Version": "2008-10-17",  
    "Id": "__default_policy_ID",  
    "Statement": [  
        {  
            "Sid": "__default_statement_ID",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "ses.amazonaws.com"  
            },  
            "Action": "SNS:Publish",  
            "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceOwner": "111122223333"  
                }  
            }  
        }  
    ]  
}
```

When publishing events to Amazon SNS, the following services support `aws:SourceOwner`:

- Amazon API Gateway
- Amazon CloudWatch
- Amazon DevOps Guru
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon GameLift
- Amazon Pinpoint SMS and Voice API
- Amazon RDS
- Amazon Redshift
- Amazon Simple Storage Service
- Amazon S3 Glacier
- Amazon SES
- AWS CodeCommit
- AWS Directory Service
- AWS Lambda
- AWS Systems Manager Incident Manager

#### **aws:SourceAccount versus aws:SourceOwner**

The `aws:SourceAccount` and `aws:SourceOwner` condition keys are each set by some AWS services when they publish to an Amazon SNS topic. When supported, the value will be the 12-digit AWS account ID on whose behalf the service is publishing data. Some services support one, and some support the other.

- See [Allow Amazon S3 event notifications to publish to a topic \(p. 376\)](#) for how Amazon S3 notifications use `aws:SourceAccount` and a list of AWS services that support that condition.
- See [Allow Amazon SES to publish to a topic that is owned by another account \(p. 377\)](#) for how Amazon SES uses `aws:SourceOwner` and a list of AWS services that support that condition.

## Allow accounts in an organization in AWS Organizations to publish to a topic in a different account

The AWS Organizations service helps you to centrally manage billing, control access and security, and share resources across your AWS accounts.

You can find your organization ID in the [Organizations console](#). For more information, see [Viewing details of an organization from the management account](#).

In this example, any AWS account in organization `myOrgId` can publish to Amazon SNS topic `MyTopic` in account `444455556666`. The policy checks the organization ID value using the `aws:PrincipalOrgID` global condition key.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {"AWS": "*"},  
            "Action": "SNS:Publish",  
            "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",  
            "Condition": {"StringEquals": {"aws:PrincipalOrgID": "myOrgId"}},  
        }  
    ]  
}
```

## Allow any CloudWatch alarm to publish to a topic in a different account

In this case, any CloudWatch alarms in account `111122223333` are allowed to publish to an Amazon SNS topic in account `444455556666`.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {"AWS": "*"},  
            "Action": "SNS:Publish",  
            "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",  
            "Condition": {"ArnLike": {"aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:/*"}},  
        }  
    ]  
}
```

## Restrict publication to an Amazon SNS topic only from a specific VPC endpoint

In this case, the topic in account `444455556666` is allowed to publish only from the VPC endpoint with the ID `vpce-1ab2c34d`.

```
{
```

```
    "Statement": [{
        "Effect": "Deny",
        "Principal": "*",
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "StringNotEquals": {
                "aws:sourceVpce": "vpce-1ab2c34d"
            }
        }
    }]
}
```

## Using identity-based policies with Amazon SNS

### Topics

- [IAM and Amazon SNS policies together \(p. 380\)](#)
- [Amazon SNS resource ARN format \(p. 382\)](#)
- [Amazon SNS API actions \(p. 383\)](#)
- [Amazon SNS policy keys \(p. 383\)](#)
- [Example policies for Amazon SNS \(p. 383\)](#)

Amazon Simple Notification Service integrates with AWS Identity and Access Management (IAM) so that you can specify which Amazon SNS actions a user in your AWS account can perform with Amazon SNS resources. You can specify a particular topic in the policy. For example, you could use variables when creating an IAM policy that grants certain users in your organization permission to use the Publish action with specific topics in your AWS account. For more information, see [Policy Variables](#) in the *Using IAM* guide.

### Important

Using Amazon SNS with IAM doesn't change how you use Amazon SNS. There are no changes to Amazon SNS actions, and no new Amazon SNS actions related to users and access control.

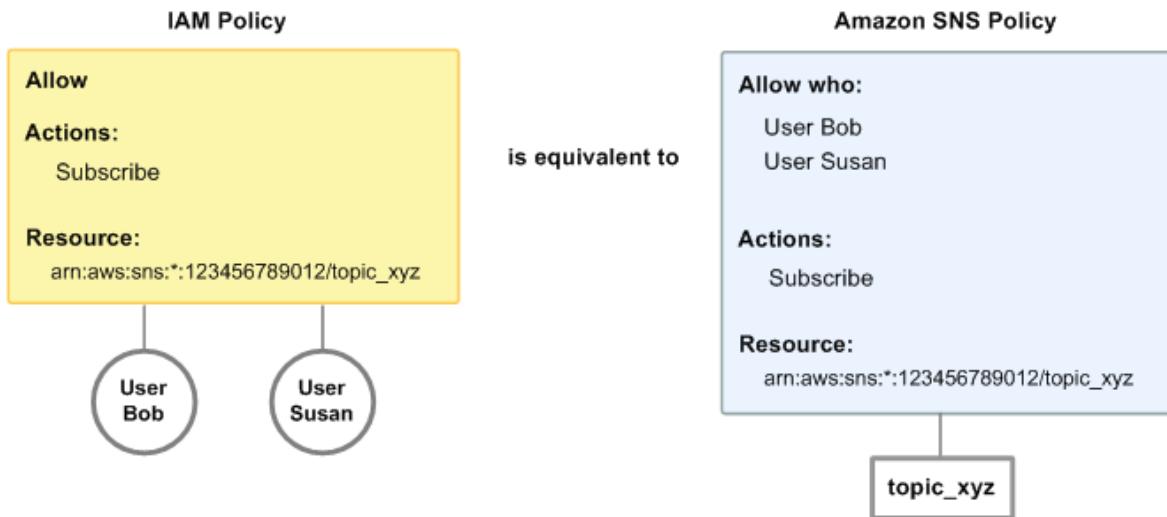
For examples of policies that cover Amazon SNS actions and resources, see [Example policies for Amazon SNS \(p. 383\)](#).

## IAM and Amazon SNS policies together

You use an IAM policy to restrict your users' access to Amazon SNS actions and topics. An IAM policy can restrict access only to users within your AWS account, not to other AWS accounts.

You use an Amazon SNS policy with a particular topic to restrict who can work with that topic (for example, who can publish messages to it, who can subscribe to it, etc.). Amazon SNS policies can grant access to other AWS accounts, or to users within your own AWS account.

To grant your users permissions for your Amazon SNS topics, you can use IAM policies, Amazon SNS policies, or both. For the most part, you can achieve the same results with either. For example, the following diagram shows an IAM policy and an Amazon SNS policy that are equivalent. The IAM policy allows the Amazon SNS Subscribe action for the topic called `topic_xyz` in your AWS account. The IAM policy is attached to the users Bob and Susan (which means that Bob and Susan have the permissions stated in the policy). The Amazon SNS policy likewise grants Bob and Susan permission to access `Subscribe` for `topic_xyz`.



#### Note

The preceding example shows simple policies with no conditions. You could specify a particular condition in either policy and get the same result.

There is one difference between AWS IAM and Amazon SNS policies: The Amazon SNS policy system lets you grant permission to other AWS accounts, whereas the IAM policy doesn't.

It's up to you how you use both of the systems together to manage your permissions, based on your needs. The following examples show how the two policy systems work together.

#### Example 1

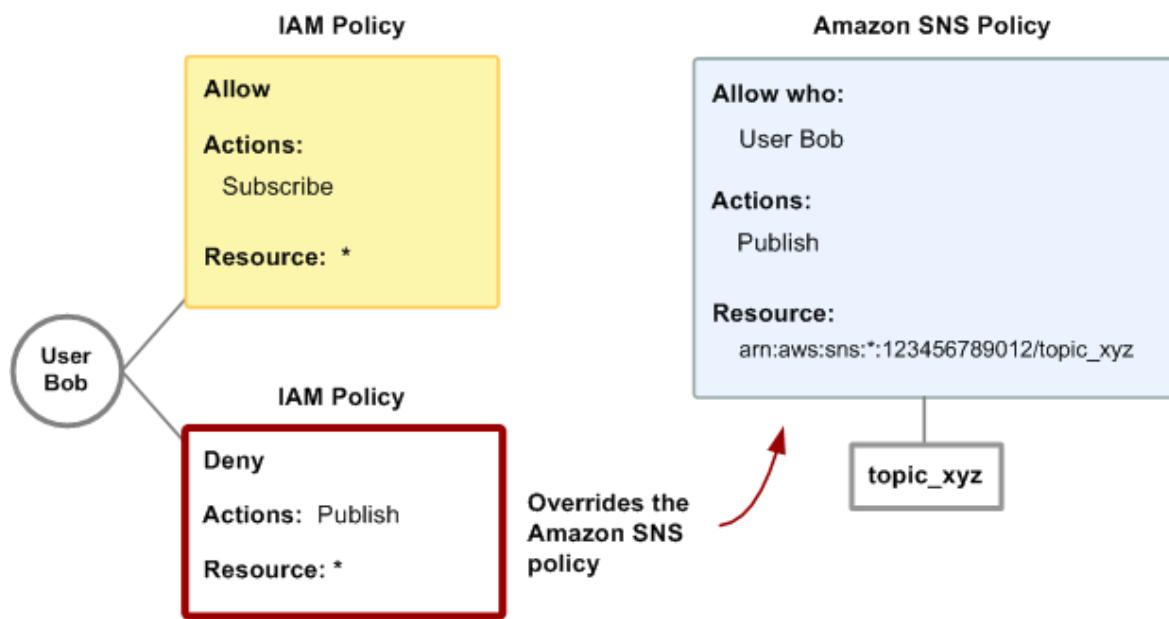
In this example, both an IAM policy and an Amazon SNS policy apply to Bob. The IAM policy grants him permission for **Subscribe** on any of the AWS account's topics, whereas the Amazon SNS policy grants him permission to use **Publish** on a specific topic (**topic\_xyz**). The following diagram illustrates the concept.



If Bob were to send a request to subscribe to any topic in the AWS account, the IAM policy would allow the action. If Bob were to send a request to publish a message to topic\_xyz, the Amazon SNS policy would allow the action.

### Example 2

In this example, we build on example 1 (where Bob has two policies that apply to him). Let's say that Bob publishes messages to topic\_xyz that he shouldn't have, so you want to entirely remove his ability to publish to topics. The easiest thing to do is to add an IAM policy that denies him access to the Publish action on all topics. This third policy overrides the Amazon SNS policy that originally gave him permission to publish to topic\_xyz, because an explicit deny always overrides an allow (for more information about policy evaluation logic, see [Evaluation logic \(p. 371\)](#)). The following diagram illustrates the concept.



For examples of policies that cover Amazon SNS actions and resources, see [Example policies for Amazon SNS \(p. 383\)](#). For more information about writing Amazon SNS policies, go to the [technical documentation for Amazon SNS](#).

## Amazon SNS resource ARN format

For Amazon SNS, topics are the only resource type you can specify in a policy. The following is the Amazon Resource Name (ARN) format for topics.

```
arn:aws:sns:region:account_ID:topic_name
```

For more information about ARNs, go to [ARNs](#) in *IAM User Guide*.

### Example

The following is an ARN for a topic named MyTopic in the us-east-2 region, belonging to AWS account 123456789012.

```
arn:aws:sns:us-east-2:123456789012:MyTopic
```

### Example

If you had a topic named MyTopic in each of the different Regions that Amazon SNS supports, you could specify the topics with the following ARN.

```
arn:aws:sns:*:123456789012:MyTopic
```

You can use \* and ? wildcards in the topic name. For example, the following could refer to all the topics created by Bob that he has prefixed with bob\_.

```
arn:aws:sns:*:123456789012:bob_*
```

As a convenience to you, when you create a topic, Amazon SNS returns the topic's ARN in the response.

## Amazon SNS API actions

In an IAM policy, you can specify any actions that Amazon SNS offers. However, the ConfirmSubscription and Unsubscribe actions do not require authentication, which means that even if you specify those actions in a policy, IAM won't restrict users' access to those actions.

Each action you specify in a policy must be prefixed with the lowercase string sns:. To specify all Amazon SNS actions, for example, you would use sns:\*. For a list of the actions, go to the [Amazon Simple Notification Service API Reference](#).

## Amazon SNS policy keys

Amazon SNS implements the following AWS wide policy keys, plus some service-specific keys.

For a list of condition keys supported by each AWS service, see [Actions, resources, and condition keys for AWS services](#) in the *IAM User Guide*. For a list of condition keys that can be used in multiple AWS services, see [AWS global condition context keys](#) in the *IAM User Guide*.

Amazon SNS uses the following service-specific keys. Use these keys in policies that restrict access to `Subscribe` requests.

- **sns:endpoint**—The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example policies for Amazon SNS \(p. 383\)](#)) to restrict access to specific endpoints (for example, \*@yourcompany.com).
- **sns:protocol**—The `protocol` value from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example policies for Amazon SNS \(p. 383\)](#)) to restrict publication to specific delivery protocols (for example, https).

## Example policies for Amazon SNS

This section shows several simple policies for controlling user access to Amazon SNS.

### Note

In the future, Amazon SNS might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

### Example 1: Allow a group to create and manage topics

In this example, we create a policy that grants access to `CreateTopic`, `ListTopics`, `SetTopicAttributes`, and `DeleteTopic`.

```
{  
  "Statement": [  
    {"Effect": "Allow",  
     "Action": ["sns:CreateTopic", "sns>ListTopics", "sns:SetTopicAttributes",  
               "sns>DeleteTopic"],  
     "Resource": "*"  
   ]  
}
```

### Example 2: Allow the IT group to publish messages to a particular topic

In this example, we create a group for IT, and assign a policy that grants access to Publish on the specific topic of interest.

```
{  
  "Statement": [  
    {"Effect": "Allow",  
     "Action": "sns:Publish",  
     "Resource": "arn:aws:sns:*:123456789012:MyTopic"  
   ]  
}
```

### Example 3: Give users in the AWS account ability to subscribe to topics

In this example, we create a policy that grants access to the Subscribeaction, with string matching conditions for the sns:Protocol and sns:Endpoint policy keys.

```
{  
  "Statement": [  
    {"Effect": "Allow",  
     "Action": ["sns:Subscribe"],  
     "Resource": "*",  
     "Condition": {  
       "StringLike": {  
         "SNS:Endpoint": "*@example.com"  
       },  
       "StringEquals": {  
         "sns:Protocol": "email"  
       }  
     }  
   ]  
}
```

### Example 4: Allow a partner to publish messages to a particular topic

You can use an Amazon SNS policy or an IAM policy to allow a partner to publish to a specific topic. If your partner has an AWS account, it might be easier to use an Amazon SNS policy. However, anyone in the partner's company who possesses the AWS security credentials could publish messages to the topic. This example assumes that you want to limit access to a particular person (or application). To do this you need to treat the partner like a user within your own company, and use a IAM policy instead of an Amazon SNS policy.

For this example, we create a group called WidgetCo that represents the partner company; we create a user for the specific person (or application) at the partner company who needs access; and then we put the user in the group.

We then attach a policy that grants the group Publish access on the specific topic named *WidgetPartnerTopic*.

We also want to prevent the WidgetCo group from doing anything else with topics, so we add a statement that denies permission to any Amazon SNS actions other than Publish on any topics other

than WidgetPartnerTopic. This is necessary only if there's a broad policy elsewhere in the system that grants users wide access to Amazon SNS.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sns:Publish",  
            "Resource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"  
        },  
        {  
            "Effect": "Deny",  
            "NotAction": "sns:Publish",  
            "NotResource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"  
        }  
    ]  
}
```

## Using temporary security credentials with Amazon SNS

In addition to creating IAM users with their own security credentials, IAM also enables you to grant temporary security credentials to any user allowing this user to access your AWS services and resources. You can manage users who have AWS accounts; these users are IAM users. You can also manage users for your system who do not have AWS accounts; these users are called federated users. Additionally, "users" can also be applications that you create to access your AWS resources.

You can use these temporary security credentials in making requests to Amazon SNS. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials Amazon SNS denies the request.

For more information about IAM support for temporary security credentials, go to [Granting Temporary Access to Your AWS Resources](#) in *Using IAM*.

### Example Using temporary security credentials to authenticate an Amazon SNS request

The following example demonstrates how to obtain temporary security credentials to authenticate an Amazon SNS request.

```
http://sns.us-east-2.amazonaws.com/  
?Name=My-Topic  
&Action/CreateTopic  
&Signature=gfzIF53exFVdpSNb8AiwN3Lv%2FNYXh6S%2Br3yySK70oX4%3D  
&SignatureVersion=2  
&SignatureMethod=HmacSHA256  
&Timestamp=2010-03-31T12%3A00%3A00.000Z  
&SecurityToken=SecurityTokenValue  
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
```

## Amazon SNS API permissions: Actions and resources reference

The following list grants information specific to the Amazon SNS implementation of access control:

- Each policy must cover only a single topic (when writing a policy, don't include statements that cover different topics)
- Each policy must have a unique policy ID
- Each statement in a policy must have a unique statement sid

## Policy quotas

The following table lists the maximum quotas for a policy statement.

Name	Maximum quota
Bytes	30 kb
Statements	100
Principals	1 to 200 (0 is invalid.)
Resource	1 (0 is invalid. The value must match the ARN of the policy's topic.)

## Valid Amazon SNS policy actions

Amazon SNS supports the actions shown in the following table.

Action	Description
sns:AddPermission	Grants permission to add permissions to the topic policy.
sns:DeleteTopic	Grants permission to delete a topic.
sns:GetTopicAttributes	Grants permission to receive all of the topic attributes.
sns>ListSubscriptionsByTopic	Grants permission to retrieve all the subscriptions to a specific topic.
sns:Publish	Grants permission to publish to a topic or endpoint. For more information, see <a href="#">Publish</a> in the Amazon Simple Notification Service API Reference
sns:RemovePermission	Grants permission to remove any permissions in the topic policy.
sns:SetTopicAttributes	Grants permission to set a topic's attributes.
sns:Subscribe	Grants permission to subscribe to a topic.

## Service-specific keys

Amazon SNS uses the following service-specific keys. You can use these in policies that restrict access to `Subscribe` requests.

- **sns:endpoint**—The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example policies for Amazon SNS \(p. 383\)](#)) to restrict access to specific endpoints (for example, `*@example.com`).
- **sns:protocol**—The `protocol` value from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example policies for Amazon SNS \(p. 383\)](#)) to restrict publication to specific delivery protocols (for example, `https`).

### Important

When you use a policy to control access by `sns:Endpoint`, be aware that DNS issues might affect the endpoint's name resolution in the future.

# Logging and monitoring in Amazon SNS

This section provides information about logging and monitoring Amazon SNS topics.

## Topics

- [Logging Amazon SNS API calls using CloudTrail \(p. 387\)](#)
- [Monitoring Amazon SNS topics using CloudWatch \(p. 390\)](#)

## Logging Amazon SNS API calls using CloudTrail

Amazon SNS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon SNS. CloudTrail captures API calls for Amazon SNS as events. The calls captured include calls from the Amazon SNS console and code calls to the Amazon SNS API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon SNS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SNS, 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](#).

## Amazon SNS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon SNS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Amazon SNS, 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](#)

Amazon SNS supports logging the following actions as events in CloudTrail log files:

- [AddPermission](#)
- [CheckIfPhoneNumberIsOptedOut](#)
- [ConfirmSubscription](#)
- [CreatePlatformApplication](#)
- [CreatePlatformEndpoint](#)
- [CreateSMSSandboxPhoneNumber](#)
- [CreateTopic](#)
- [DeleteEndpoint](#)

- [DeletePlatformApplication](#)
- [DeleteSMSSandboxPhoneNumber](#)
- [DeleteTopic](#)
- [GetEndpointAttributes](#)
- [GetPlatformApplicationAttributes](#)
- [GetSMSAttributes](#)
- [GetSMSSandboxAccountStatus](#)
- [GetSubscriptionAttributes](#)
- [GetTopicAttributes](#)
- [ListEndpointsByPlatformApplication](#)
- [ListOriginationNumbers](#)
- [ListPhoneNumbersOptedOut](#)
- [ListPlatformApplications](#)
- [ListSMSSandboxPhoneNumbers](#)
- [ListSubscriptions](#)
- [ListSubscriptionsByTopic](#)
- [ListTagsForResource](#)
- [ListTopics](#)
- [OptInPhoneNumber](#)
- [RemovePermission](#)
- [SetEndpointAttributes](#)
- [SetPlatformApplicationAttributes](#)
- [SetSMSAttributes](#)
- [SetSubscriptionAttributes](#)
- [SetTopicAttributes](#)
- [Subscribe](#)
- [TagResource](#)
- [Unsubscribe](#)
- [UntagResource](#)
- [VerifySMSSandboxPhoneNumber](#)

**Note**

When you are not logged in to Amazon Web Services (unauthenticated mode) and either the [ConfirmSubscription](#) or [Unsubscribe](#) actions are invoked, then they will not be logged to CloudTrail. Such as, when you choose the provided link in an email notification to confirm a pending subscription to a topic, the [ConfirmSubscription](#) action is invoked in unauthenticated mode. In this example, the [ConfirmSubscription](#) action would not be logged to CloudTrail.

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](#).

## Example: Amazon SNS log file entries

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 example shows a CloudTrail log entry that demonstrates the `ListTopics`, `CreateTopic`, and `DeleteTopic` actions.

```
{  
  "Records": [  
    {  
      "eventVersion": "1.02",  
      "userIdentity": {  
        "type": "IAMUser",  
        "userName": "Bob"  
      },  
      "principalId": "EX_PRINCIPAL_ID",  
      "arn": "arn:aws:iam::123456789012:user/Bob",  
      "accountId": "123456789012",  
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE"  
    },  
    {  
      "eventTime": "2014-09-30T00:00:00Z",  
      "eventSource": "sns.amazonaws.com",  
      "eventName": "ListTopics",  
      "awsRegion": "us-west-2",  
      "sourceIPAddress": "127.0.0.1",  
      "userAgent": "aws-sdk-java/unknown-version",  
      "requestParameters": {  
        "nextToken": "ABCDEF1234567890EXAMPLE=="  
      },  
      "responseElements": null,  
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",  
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",  
      "eventType": "AwsApiCall",  
      "recipientAccountId": "123456789012"  
    },  
    {  
      "eventVersion": "1.02",  
      "userIdentity": {  
        "type": "IAMUser",  
        "userName": "Bob"  
      },  
      "principalId": "EX_PRINCIPAL_ID",  
      "arn": "arn:aws:iam::123456789012:user/Bob",  
      "accountId": "123456789012",  
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE"  
    },  
    {  
      "eventTime": "2014-09-30T00:00:00Z",  
      "eventSource": "sns.amazonaws.com",  
      "eventName": "CreateTopic",  
      "awsRegion": "us-west-2",  
      "sourceIPAddress": "127.0.0.1",  
      "userAgent": "aws-sdk-java/unknown-version",  
      "requestParameters": {  
        "name": "hello"  
      },  
      "responseElements": {  
        "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"  
      },  
      "requestID": "example7-5cd3-5323-8a00-f1889011fee9",  
      "eventID": "examplec-4f2f-4625-8378-130ac89660b1",  
      "eventType": "AwsApiCall",  
      "recipientAccountId": "123456789012"  
    }  
  ]  
}
```

```
},
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "userName": "Bob",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2014-09-30T00:00:00Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "DeleteTopic",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
  },
  "responseElements": null,
  "requestID": "example5-4faa-51d5-aab2-803a8294388d",
  "eventId": "example8-6443-4b4d-abfd-1b867280d964",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
},
]
```

## Monitoring Amazon SNS topics using CloudWatch

Amazon SNS and Amazon CloudWatch are integrated so you can collect, view, and analyze metrics for every active Amazon SNS notification. Once you have configured CloudWatch for Amazon SNS, you can gain better insight into the performance of your Amazon SNS topics, push notifications, and SMS deliveries. For example, you can set an alarm to send you an email notification if a specified threshold is met for an Amazon SNS metric, such as `NumberOfNotificationsFailed`. For a list of all the metrics that Amazon SNS sends to CloudWatch, see [Amazon SNS metrics \(p. 391\)](#). For more information about Amazon SNS push notifications, see [Mobile push notifications \(p. 234\)](#).

### Note

The metrics you configure with CloudWatch for your Amazon SNS topics are automatically collected and pushed to CloudWatch at *1-minute* intervals. These metrics are gathered on all topics that meet the CloudWatch guidelines for being active. A topic is considered active by CloudWatch for up to six hours from the last activity (that is, any API call) on the topic. There is no charge for the Amazon SNS metrics reported in CloudWatch; they are provided as part of the Amazon SNS service.

## View CloudWatch metrics for Amazon SNS

You can monitor metrics for Amazon SNS using the CloudWatch console, CloudWatch's own command line interface (CLI), or programmatically using the CloudWatch API. The following procedures show you how to access the metrics using the AWS Management Console.

### To view metrics using the CloudWatch console

1. Sign in to the [CloudWatch console](#).
2. On the navigation panel, choose **Metrics**.
3. On the **All metrics** tab, choose **SNS**, and then choose one of the following dimensions:
  - **Country, SMS Type**

- **PhoneNumber**
  - **Topic Metrics**
  - **Metrics with no dimensions**
4. To view more detail, choose a specific item. For example, if you choose **Topic Metrics** and then choose **NumberOfMessagesPublished**, the average number of published Amazon SNS messages for a 1-minute period throughout the time range of 6 hours is displayed.

## Set CloudWatch alarms for Amazon SNS metrics

CloudWatch also allows you to set alarms when a threshold is met for a metric. For example, you could set an alarm for the metric, **NumberOfNotificationsFailed**, so that when your specified threshold number is met within the sampling period, then an email notification would be sent to inform you of the event.

### To set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Alarms**, and then choose the **Create Alarm** button. This launches the **Create Alarm** wizard.
3. Scroll through the Amazon SNS metrics to locate the metric you want to place an alarm on. Select the metric to create an alarm on and choose **Continue**.
4. Fill in the **Name**, **Description**, **Threshold**, and **Time** values for the metric, and then choose **Continue**.
5. Choose **Alarm** as the alarm state. If you want CloudWatch to send you an email when the alarm state is reached, choose either an existing Amazon SNS topic or choose **Create New Email Topic**. If you choose **Create New Email Topic**, you can set the name and email addresses for a new topic. This list will be saved and appear in the drop-down box for future alarms. Choose **Continue**.

#### Note

If you use **Create New Email Topic** to create a new Amazon SNS topic, the email addresses must be verified before they will receive notifications. Emails are sent only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they will not receive a notification.

6. At this point, the **Create Alarm** wizard gives you a chance to review the alarm you're about to create. If you need to make any changes, you can use the **Edit** links on the right. Once you are satisfied, choose **Create Alarm**.

For more information about using CloudWatch and alarms, see the [CloudWatch Documentation](#).

## Amazon SNS metrics

Amazon SNS sends the following metrics to CloudWatch.

Metric	Description
NumberOfMessagesPublished	The number of messages published to your Amazon SNS topics.  Units: <i>Count</i>  Valid Dimensions: Application, PhoneNumber, Platform, and TopicName  Valid Statistics: Sum

Metric	Description
NumberOfNotificationsDelivered	<p>The number of messages successfully delivered from your Amazon SNS topics to subscribing endpoints.</p> <p>For a delivery attempt to succeed, the endpoint's subscription must accept the message. A subscription accepts a message if a.) it lacks a filter policy or b.) its filter policy includes attributes that match those assigned to the message. If the subscription rejects the message, the delivery attempt isn't counted for this metric.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum</p>
NumberOfNotificationsFailed	<p>The number of messages that Amazon SNS failed to deliver.</p> <p>For Amazon SQS, email, SMS, or mobile push endpoints, the metric increments by 1 when Amazon SNS stops attempting message deliveries. For HTTP or HTTPS endpoints, the metric includes every failed delivery attempt, including retries that follow the initial attempt. For all other endpoints, the count increases by 1 when the message fails to deliver (regardless of the number of attempts).</p> <p>This metric does not include messages that were rejected by subscription filter policies.</p> <p>You can control the number of retries for HTTP endpoints. For more information, see <a href="#">Amazon SNS message delivery retries (p. 92)</a>.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFilteredOut	<p>The number of messages that were rejected by subscription filter policies. A filter policy rejects a message when the message attributes don't match the policy attributes.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>

Metric	Description
NumberOfNotificationsFilteredOut-InvalidAttributes	<p>The number of messages that were rejected by subscription filter policies because the messages' attributes are invalid – for example, because the attribute JSON is incorrectly formatted.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFilteredOut-NoMessageAttributes	<p>The number of messages that were rejected by subscription filter policies because the messages have no attributes.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsRedrivenToDlq	<p>The number of messages that have been moved to a dead-letter queue.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFailedToRedrive	<p>The number of messages that couldn't be moved to a dead-letter queue.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Sum, Average</p>
PublishSize	<p>The size of messages published.</p> <p>Units: <i>Bytes</i></p> <p>Valid Dimensions: Application, PhoneNumber, Platform, and TopicName</p> <p>Valid Statistics: Minimum, Maximum, Average and Count</p>

Metric	Description
SMSMonthToDateSpentUSD	<p>The charges you have accrued since the start of the current calendar month for sending SMS messages.</p> <p>You can set an alarm for this metric to know when your month-to-date charges are close to the monthly SMS spend quota for your account. When Amazon SNS determines that sending an SMS message would incur a cost that exceeds this quota, it stops publishing SMS messages within minutes.</p> <p>For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see <a href="#">Setting SMS messaging preferences (p. 189)</a>.</p> <p>Units: <i>USD</i></p> <p>Valid Dimensions: <i>PhoneNumber</i></p> <p>Valid Statistics: Maximum</p>
SMSSuccessRate	<p>The rate of successful SMS message deliveries.</p> <p>Units: <i>Count</i></p> <p>Valid Dimensions: <i>PhoneNumber</i></p> <p>Valid Statistics: Sum, Average, Data Samples</p>

## Dimensions for Amazon SNS metrics

Amazon Simple Notification Service sends the following dimensions to CloudWatch.

Dimension	Description
Application	Filters on application objects, which represent an app and device registered with one of the supported push notification services, such as APNs and FCM.
Application, Platform	Filters on application and platform objects, where the platform objects are for the supported push notification services, such as APNs and FCM.
Country	Filters on the destination country or region of an SMS message. The country or region is represented by its ISO 3166-1 alpha-2 code.
PhoneNumber	Filters on the phone number when you publish SMS directly to a phone number (without a topic).
Platform	Filters on platform objects for the push notification services, such as APNs and FCM.
TopicName	Filters on Amazon SNS topic names.
SMSType	Filters on the message type of SMS message. Can be <i>promotional</i> or <i>transactional</i> .

## Compliance validation for Amazon SNS

Third-party auditors assess the security and compliance of Amazon SNS as part of multiple AWS compliance programs, including the Health Insurance Portability and Accountability Act (HIPAA).

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Amazon SNS 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:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules in the AWS Config Developer Guide](#) – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

## Resilience in Amazon SNS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures. For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

## Infrastructure security in Amazon SNS

As a managed service, Amazon SNS is protected by the AWS global network security procedures described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

Use AWS API actions to access Amazon SNS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

You must sign requests using both an access key ID and a secret access key associated with an IAM principal. Alternatively, you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials for signing requests.

You can call these API actions from any network location, but Amazon SNS supports resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon SNS policies to control access from specific Amazon VPC endpoints or specific VPCs. This effectively isolates network access to a given Amazon SNS queue from only the specific VPC within the AWS network. For more information, see [Restrict publication to an Amazon SNS topic only from a specific VPC endpoint \(p. 379\)](#).

## Amazon SNS security best practices

AWS provides many security features for Amazon SNS. Review these security features in the context of your own security policy.

### Note

The guidance for these security features applies to common use cases and implementations. We recommend that you review these best practices in the context of your specific use case, architecture, and threat model.

## Preventative best practices

The following are preventative security best practices for Amazon SNS.

### Topics

- [Ensure topics aren't publicly accessible \(p. 396\)](#)
- [Implement least-privilege access \(p. 396\)](#)
- [Use IAM roles for applications and AWS services which require Amazon SNS access \(p. 397\)](#)
- [Implement server-side encryption \(p. 397\)](#)
- [Enforce encryption of data in transit \(p. 397\)](#)
- [Consider using VPC endpoints to access Amazon SNS \(p. 398\)](#)
- [Ensure subscriptions are not configured to deliver to raw http endpoints \(p. 398\)](#)

### Ensure topics aren't publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your Amazon SNS topic, you should ensure that your topic isn't publicly accessible (accessible by everyone in the world or by any authenticated AWS user).

- Avoid creating policies with `Principal` set to "".
- Avoid using a wildcard (\*). Instead, name a specific user or users.

### Implement least-privilege access

When you grant permissions, you decide who receives them, which topics the permissions are for, and specific API actions that you want to allow for these topics. Implementing the principle of least privilege is important to reducing security risks. It also helps to reduce the negative effect of errors or malicious intent.

Follow the standard security advice of granting least privilege. That is, grant only the permissions required to perform a specific task. You can implement least privilege by using a combination of security policies pertaining to user access.

Amazon SNS uses the publisher-subscriber model, requiring three types of user account access:

- **Administrators** – Access to creating, modifying, and deleting topics. Administrators also control topic policies.
- **Publishers** – Access to sending messages to topics.
- **Subscribers** – Access to subscribing to topics.

For more information, see the following sections:

- [Identity and access management in Amazon SNS \(p. 364\)](#)
- [Amazon SNS API permissions: Actions and resources reference \(p. 385\)](#)

## Use IAM roles for applications and AWS services which require Amazon SNS access

For applications or AWS services, such as Amazon EC2, to access Amazon SNS topics, they must use valid AWS credentials in their AWS API requests. Because these credentials aren't rotated automatically, you shouldn't store AWS credentials directly in the application or EC2 instance.

You should use an IAM role to manage temporary credentials for applications or services that need to access Amazon SNS. When you use a role, you don't need to distribute long-term credentials (such as a user name, password, and access keys) to an EC2 instance or AWS service, such as AWS Lambda. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources.

For more information, see [IAM Roles](#) and [Common Scenarios for Roles: Users, Applications, and Services](#) in the *IAM User Guide*.

## Implement server-side encryption

To mitigate data leakage issues, use encryption at rest to encrypt your messages using a key stored in a different location from the location that stores your messages. Server-side encryption (SSE) provides data encryption at rest. Amazon SNS encrypts your data at the message level when it stores it, and decrypts the messages for you when you access them. SSE uses keys managed in AWS Key Management Service. When you authenticate your request and have access permissions, there is no difference between accessing encrypted and unencrypted topics.

For more information, see [Encryption at rest \(p. 344\)](#) and [Key management \(p. 345\)](#).

## Enforce encryption of data in transit

It's possible, but not recommended, to publish messages that are not encrypted during transit by using HTTP. You can't, however, use HTTP when publishing to an encrypted SNS topic.

AWS recommends that you use HTTPS instead of HTTP. When you use HTTPS, messages are automatically encrypted during transit, even if the SNS topic itself isn't encrypted. Without HTTPS, a network-based attacker can eavesdrop on network traffic or manipulate it using an attack such as man-in-the-middle.

To enforce only encrypted connections over HTTPS, add the `aws:SecureTransport` condition in the IAM policy that's attached to unencrypted SNS topics. This forces message publishers to use HTTPS instead of HTTP. You can use the following example policy as a guide:

```
{  
  "Id": "ExamplePolicy",  
  "Version": "2012-10-17",  
  "Statement": [
```

```
{  
    "Sid": "AllowPublishThroughSSLOnly",  
    "Action": "SNS:Publish",  
    "Effect": "Deny",  
    "Resource": [  
        "arn:aws:sns:us-east-1:1234567890:test-topic"  
    ],  
    "Condition": {  
        "Bool": {  
            "aws:SecureTransport": "false"  
        }  
    },  
    "Principal": "*"  
}  
]  
}
```

## Consider using VPC endpoints to access Amazon SNS

If you have topics that you must be able to interact with, but these topics must absolutely not be exposed to the internet, use VPC endpoints to limit topic access to only the hosts within a particular VPC. You can use topic policies to control access to topics from specific Amazon VPC endpoints or from specific VPCs.

Amazon SNS VPC endpoints provide two ways to control access to your messages:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint.
- You can control which VPCs or VPC endpoints have access to your topic using a topic policy.

For more information, see [Creating the endpoint \(p. 353\)](#) and [Creating an Amazon VPC endpoint policy for Amazon SNS \(p. 354\)](#).

## Ensure subscriptions are not configured to deliver to raw http endpoints

Avoid configuring subscriptions to deliver to a raw http endpoints. Always have subscriptions delivering to an endpoint domain name. For example, a subscription configured to deliver to an endpoint, `http://1.2.3.4/my-path`, should be changed to `http://my.domain.name/my-path`.

# Troubleshooting Amazon SNS topics

This section provides information about troubleshooting Amazon SNS topics.

## Troubleshooting Amazon SNS topics using AWS X-Ray

AWS X-Ray collects data about requests that your application serves, and lets you view and filter data to identify potential issues and opportunities for optimization. For any traced request to your application, you can see detailed information about the request, the response, and the calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

You can use X-Ray with Amazon SNS to trace and analyze the messages that travel through your application. You can use the AWS Management Console to view the map of connections between Amazon SNS and other services that your application uses. You can also use the console to view metrics such as average latency and failure rates. For more information, see [Amazon SNS and AWS X-Ray](#) in the [AWS X-Ray Developer Guide](#).

# Documentation history

The following table describes recent changes to the *Amazon Simple Notification Service Developer Guide*.

update-history-change	update-history-description	update-history-date
<a href="#">New senders of SMS messages are placed in the SMS sandbox</a>	The SMS sandbox exists to help prevent fraud and abuse, and to help protect your reputation as a sender. While your AWS account is in the SMS sandbox, you can send SMS messages only to verified destination phone numbers.	June 1, 2021
<a href="#">New attributes for sending SMS messages to recipients in India</a>	Two new attributes, <b>Entity ID</b> and <b>Template ID</b> , are now required for sending SMS messages to recipients in India.	April 22, 2021
<a href="#">Updates to message filtering operators</a>	A new operator, <code>cidr</code> , is available for matching message source IP addresses and subnets. You can now also check for the absence of an attribute key, and use a prefix with the <code>anything-but</code> operator for attribute string matching.	April 7, 2021
<a href="#">Ending support for P2P long codes for US destinations</a>	Effective June 1, 2021, US telecom providers no longer support using person-to-person (P2P) long codes for application-to-person (A2P) communications to US destinations. Instead, you can use short codes, toll-free numbers, or a new type of origination number called <code>10DLC</code> .	February 16, 2021
<a href="#">Support for 1-minute Amazon CloudWatch metrics</a>	The 1-minute CloudWatch metric for Amazon SNS is now available in all AWS Regions.	January 28, 2021
<a href="#">Support for Amazon Kinesis Data Firehose endpoints</a>	You can subscribe Kinesis Data Firehose delivery streams to SNS topics. This allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more.	January 12, 2021

<a href="#">Origination numbers are available</a>	You can use origination numbers when sending text messages (SMS).	October 23, 2020
<a href="#">Support for Amazon SNS FIFO topics</a>	To integrate distributed applications that require data consistency in near-real time, you can use Amazon SNS first-in, first-out (FIFO) topics with Amazon SQS FIFO queues.	October 22, 2020
<a href="#">The Amazon SNS Extended Client Library for Java is available</a>	You can use this library to publish large Amazon SNS messages.	August 25, 2020
<a href="#">SSE is available in the China Regions</a>	Server-side encryption (SSE) for Amazon SNS is available in the China Regions.	January 20, 2020
<a href="#">Support for using DLQs to capture undeliverable messages</a>	To capture undeliverable messages, you can use an Amazon SQS dead-letter queue (DLQ) with an Amazon SNS subscription.	November 14, 2019
<a href="#">Support for specifying custom APNs header values</a>	You can specify a custom APNs header value.	October 18, 2019
<a href="#">Support for the 'apns-push-type' header field for APNs</a>	You can use the apns-push-type header field for mobile notifications sent through APNs.	September 10, 2019
<a href="#">Support for topic troubleshooting using AWS X-Ray</a>	You can use X-Ray to troubleshoot messages passing through SNS topics.	July 24, 2019
<a href="#">Support for attribute key matching using the 'exists' operator</a>	To check whether an incoming message has an attribute whose key is listed in the filter policy, you can use the exists operator.	July 5, 2019
<a href="#">Support for anything-but matching of multiple numeric values</a>	In addition to multiple strings, Amazon SNS allows anything-but matching of multiple numeric values.	July 5, 2019
<a href="#">Amazon SNS release notes are available as an RSS feed</a>	Following the title on this page ( <b>Documentation history</b> ), choose <b>RSS</b> .	June 22, 2019

# AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.