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
Working with AWS SDKs .......................................................................................................... 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 .......................................................................................................... 32
    To subscribe an endpoint to an Amazon SNS topic ......................................................... 32
Message grouping
AWS CloudFormation

How does message batching work?

Message ordering and deduplication (FIFO topics)
FIFO topics use case
Message ordering details
Message grouping
Message delivery
Message filtering
Message deduplication
Message security
Message durability
Code examples
FIFO example (AWS SDKs)
FIFO example (AWS CloudFormation)

Message publishing
AWS Management Console
AWS SDKs

Large message payloads
Prerequisites
Example: Publishing messages to Amazon SNS with payload stored in Amazon S3
Other endpoint protocols

Message attributes
Message attribute items and validation
Data types
Reserved message attributes for mobile push notifications

Message batching
What is message batching?
How does message batching work?
Examples

Message filtering
Subscription filter policies
Example filter policies
Filter policy constraints
Attribute string value matching
Attribute numeric value matching
Attribute key matching
AND/OR logic

Applying a subscription filter policy
AWS Management Console
AWS CLI
AWS SDKs
Amazon SNS API
AWS CloudFormation

Removing a subscription filter policy
AWS Management Console
AWS CLI
Amazon SNS API

Message data protection
What is message data protection
Why use message data protection
Data protection policies
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).
Application-to-application (A2A) subscribers

Amazon SQS

AWS Lambda

HTTPS

Amazon SNS

Amazon Kinesis Data Firehose

Application-to-person (A2P) subscribers

Mobile text (SMS)

Mobile push
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. 153).

- **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. 217).

- **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. 43).

  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. 141).
  
  - To preserve any messages that aren’t delivered before the delivery retry policy ends, you can create a dead-letter queue (p. 146).

- **Message archiving and analytics**
  
  You can subscribe Kinesis Data Firehose delivery streams to SNS topics (p. 153), 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. 73).

- **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. 79).

- **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. 438).

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. 447).

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. 146) powered by Amazon SQS for undeliverable messages.
  - You can subscribe an Amazon SQS queue to an SNS topic (p. 174).
  - You can subscribe an Amazon SQS FIFO queue to an Amazon SNS FIFO topic (p. 43) 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. 173).
- **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. 475).
- **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. 153)
- Fanout to Lambda functions (p. 173)
- Fanout to Amazon SQS queues (p. 174)
- Fanout to HTTP(S) endpoints (p. 184)
- 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. 217).

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. 312).

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.

<table>
<thead>
<tr>
<th>SDK documentation</th>
<th>Code examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS SDK for C++</td>
<td>AWS SDK for C++ code examples</td>
</tr>
<tr>
<td>AWS SDK for Go</td>
<td>AWS SDK for Go code examples</td>
</tr>
<tr>
<td>AWS SDK for Java</td>
<td>AWS SDK for Java code examples</td>
</tr>
<tr>
<td>AWS SDK for JavaScript</td>
<td>AWS SDK for JavaScript code examples</td>
</tr>
<tr>
<td>AWS SDK for Kotlin</td>
<td>AWS SDK for Kotlin code examples</td>
</tr>
<tr>
<td>AWS SDK for .NET</td>
<td>AWS SDK for .NET code examples</td>
</tr>
<tr>
<td>AWS SDK for PHP</td>
<td>AWS SDK for PHP code examples</td>
</tr>
<tr>
<td>AWS SDK for Python (Boto3)</td>
<td>AWS SDK for Python (Boto3) code examples</td>
</tr>
<tr>
<td>AWS SDK for Ruby</td>
<td>AWS SDK for Ruby code examples</td>
</tr>
<tr>
<td>AWS SDK for Rust</td>
<td>AWS SDK for Rust code examples</td>
</tr>
<tr>
<td>AWS SDK for Swift</td>
<td>AWS SDK for Swift code examples</td>
</tr>
</tbody>
</table>

For examples specific to Amazon SNS, see Code examples for Amazon SNS using AWS SDKs (p. 347).

Example availability
Can't find what you need? Request a code example by using the Provide feedback link at the bottom of this page.
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. 43) in October, 2020. Currently, most AWS services support sending events to standard topics only.

Analytics services

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

Application integration services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon EventBridge – 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</td>
<td>Receive notifications of EventBridge events. For more information, see Amazon EventBridge targets in the Amazon EventBridge User Guide.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Benefit of using with Amazon SNS</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNS. EventBridge was formerly called CloudWatch Events.</td>
<td></td>
</tr>
<tr>
<td><strong>AWS Step Functions</strong> – Lets you combine AWS Lambda functions and other AWS services to build business-critical applications.</td>
<td>Receive notification of Step Functions events. For more information, see Call Amazon SNS with Step Functions in the AWS Step Functions Developer Guide.</td>
</tr>
</tbody>
</table>

### Billing & cost management services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Billing and Cost Management</strong> – Provides features that help you monitor your costs and pay your bill.</td>
<td>Receive budget notifications, price change notifications, and anomaly alerts. For more information, see the following pages in the AWS Billing User Guide:</td>
<td></td>
</tr>
<tr>
<td>• Creating an Amazon SNS topic for budget notifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Setting up notifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Detecting unusual spend with AWS Cost Anomaly Detection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Business applications services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Chime</strong> – Lets you meet, chat, and place business calls inside and outside of your organization.</td>
<td>Receive important meeting event notifications. For more information, see Amazon Chime SDK event notifications in the Amazon Chime Developer Guide.</td>
<td></td>
</tr>
</tbody>
</table>

### Compute services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon EC2 Auto Scaling</strong> – Helps you have the correct number of Amazon Elastic Compute Cloud (Amazon EC2) instances available for handling your application's load.</td>
<td>Receive notifications when Auto Scaling launches or terminates Amazon EC2 instances in your Auto Scaling group. For more information, see Getting Amazon SNS notifications when your Auto Scaling group scales in the Amazon EC2 Auto Scaling User Guide.</td>
<td></td>
</tr>
<tr>
<td><strong>EC2 Image Builder</strong> – 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.</td>
<td>Receive notifications when builds are complete. For more information, see Tracking the latest server images in EC2 Image Builder pipelines on the AWS Compute Blog.</td>
<td></td>
</tr>
</tbody>
</table>
Containers services

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

**AWS Service:** Amazon Simple Email Service (Amazon SES)

**Benefit of using with Amazon SNS:**
- Receive notifications of bounces, complaints, and deliveries. For more information, see Configuring Amazon SNS notifications for Amazon SES in the Amazon Simple Email Service Developer Guide.

### AWS Database Migration Service

**AWS Service:** AWS Database Migration Service

**Benefit of using with Amazon SNS:**
- Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see Working with events and notifications in AWS Database Migration Service in the AWS Database Migration Service User Guide.

### Amazon DynamoDB

**AWS Service:** Amazon DynamoDB

**Benefit of using with Amazon SNS:**
- Receive notifications when maintenance events occur. For more information, see Customizing DAX cluster settings in the Amazon DynamoDB Developer Guide.

### Amazon ElastiCache

**AWS Service:** Amazon ElastiCache

**Benefit of using with Amazon SNS:**
- Receive notifications when significant events occur. For more information, see Event notifications and Amazon SNS in the Amazon ElastiCache for Memcached User Guide.

### Amazon Neptune

**AWS Service:** Amazon Neptune

**Benefit of using with Amazon SNS:**
- Receive notifications when a Neptune event occurs. For more information, see Using Neptune event notification in the Neptune User Guide.

### Amazon Redshift

**AWS Service:** Amazon Redshift

**Benefit of using with Amazon SNS:**
- Receive notifications of Amazon Redshift events. For more information, see Amazon Redshift event notifications in the Amazon Redshift Management Guide.

### Amazon Relational Database Service

**AWS Service:** Amazon Relational Database Service

**Benefit of using with Amazon SNS:**
- Receive notifications of Amazon RDS events. For more information, see Using Amazon RDS event notification in the Amazon RDS User Guide.

### Developer tools services

### AWS CodeBuild

**AWS Service:** AWS CodeBuild

**Benefit of using with Amazon SNS:**
- Receive notifications when builds succeed, fail, or move from one build phase to another. For more information, see Build notifications sample for CodeBuild in the AWS CodeBuild User Guide.

### AWS CodeCommit

**AWS Service:** AWS CodeCommit

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

Front-end web & mobile services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Pinpoint</strong> –</td>
<td>Configure two-way SMS, which allows you to receive messages from your customers. For more information, see Using two-way SMS messaging in Amazon Pinpoint in the Amazon Pinpoint User Guide.</td>
</tr>
<tr>
<td>helps you engage your</td>
<td></td>
</tr>
<tr>
<td>customers by sending</td>
<td></td>
</tr>
<tr>
<td>them email, SMS and voice</td>
<td></td>
</tr>
<tr>
<td>messages, and push</td>
<td></td>
</tr>
<tr>
<td>notifications.</td>
<td></td>
</tr>
</tbody>
</table>

Game development services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon GameLift</strong> –</td>
<td>Receive matchmaking and queue event notifications. For more information, see the following pages:</td>
</tr>
<tr>
<td>provides solutions for</td>
<td></td>
</tr>
<tr>
<td>hosting session-based</td>
<td></td>
</tr>
<tr>
<td>multiplayer game servers</td>
<td></td>
</tr>
<tr>
<td>in the cloud, including</td>
<td></td>
</tr>
<tr>
<td>a fully managed service</td>
<td></td>
</tr>
<tr>
<td>for deploying, operating,</td>
<td></td>
</tr>
<tr>
<td>and scaling game servers.</td>
<td></td>
</tr>
<tr>
<td>• For matchmaking</td>
<td></td>
</tr>
<tr>
<td>notifications, see Set up</td>
<td></td>
</tr>
<tr>
<td>FlexMatch event</td>
<td></td>
</tr>
<tr>
<td>notification in the</td>
<td></td>
</tr>
<tr>
<td>Amazon GameLift FlexMatch</td>
<td></td>
</tr>
<tr>
<td>Developer Guide.</td>
<td></td>
</tr>
<tr>
<td>• For queue notifications, see Set up event notification for game session placement in the Amazon GameLift Developer Guide.</td>
<td></td>
</tr>
</tbody>
</table>
## Internet of Things services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS IoT Core</strong> – Provides the cloud services that connect your IoT devices to other devices and AWS Cloud services.</td>
<td>Receive notifications of AWS IoT Core events. For more information, see <a href="https://docs.aws.amazon.com/iotcore/latest/developerguide/creating-an-amazon-sns-rule.html">Creating an Amazon SNS rule</a> in the AWS IoT Developer Guide.</td>
</tr>
<tr>
<td><strong>AWS IoT Device Defender</strong> – Allows you to audit the configuration of your devices, monitor connected devices to detect abnormal behavior, and mitigate security risks.</td>
<td>Receive alarms when a device violates a behavior. For more information, see <a href="https://docs.aws.amazon.com/iot-device-defender/latest/userguide/tutorials/device-profiling.html">How to use AWS IoT Device Defender detect</a> in the AWS IoT Developer Guide.</td>
</tr>
<tr>
<td><strong>AWS IoT Events</strong> – Lets you monitor your equipment or device fleets for failures or changes in operation, and trigger actions when such events occur.</td>
<td>Receive notifications of AWS IoT Events events. For more information, see <a href="https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-events-using-sns.html">Amazon Simple Notification Service</a> in the AWS IoT Events Developer Guide.</td>
</tr>
<tr>
<td><strong>AWS IoT Greengrass</strong> – 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.</td>
<td>Receive notifications of AWS IoT Greengrass events. For more information, see <a href="https://docs.aws.amazon.com/iot-greengrass/latest/developerguide/sns-connector.html">SNS connector</a> in the AWS IoT Greengrass Version 1 Developer Guide.</td>
</tr>
</tbody>
</table>

## Machine learning services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon CodeGuru</strong> – Collects runtime performance data from your live applications, and provides recommendations that can help you fine-tune your application performance.</td>
<td>Receive notifications when anomalies occur. For more information, see <a href="https://docs.aws.amazon.com/codeguru/latest/developer-guide/codeguru-features.html">Working with anomalies and recommendation reports</a> in the Amazon CodeGuru User Guide.</td>
</tr>
<tr>
<td><strong>Amazon DevOps Guru</strong> – Generates operational insights using machine learning to help you improve the performance of your operational applications.</td>
<td>Forward insights and confirmations. For more information, see <a href="https://aws.amazon.com/about-aws/blog/devops-guru">Deliver ML-powered operational insights to your on-call teams via PagerDuty with Amazon DevOps Guru</a> on the AWS Management &amp; Governance Blog.</td>
</tr>
<tr>
<td><strong>Amazon Lookout for Metrics</strong> – Finds anomalies in your data, determines their root causes, and enables you to quickly take action.</td>
<td>Receive notifications of anomalies. For more information, see <a href="https://docs.aws.amazon.com/lookoutformetrics/latest/userguide/using-with-sns.html">Using Amazon SNS with Lookout for Metrics</a> in the Amazon Lookout for Metrics Developer Guide.</td>
</tr>
<tr>
<td><strong>Amazon Rekognition</strong> – Lets you add image and video analysis to your applications</td>
<td>Receive notifications of request results. For more information, see <a href="https://docs.aws.amazon.com/rekognition/latest/dg/video-analysis-results-notification.html">Reference: Video analysis results notification</a> in the Amazon Rekognition Developer Guide.</td>
</tr>
<tr>
<td><strong>Amazon SageMaker</strong> – Enables data scientists and developers to build and train machine learning models, and then directly deploy them into a production-ready hosted environment.</td>
<td>Receive notifications when a data object is labeled. For more information, see <a href="https://docs.aws.amazon.com/sagemaker/latest/dg/stream-labelling-jobs.html">Creating a streaming labeling job</a> in the Amazon SageMaker Developer Guide.</td>
</tr>
</tbody>
</table>
### Management & governance services

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

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Elastic Transcoder – Lets you convert media files that you stored in Amazon S3 into media files in the formats required by consumer playback devices.</td>
<td>Receive notifications when jobs change status. For more information, see <a href="https://docs.aws.amazon.com/elastictranscoder/latest/developerguide/jobs-status-update.html">Notifications of job status</a> in the <em>Amazon Elastic Transcoder Developer Guide</em>.</td>
</tr>
</tbody>
</table>

# Migration & transfer services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Application Discovery Service – Helps you plan your migration to the AWS Cloud by collecting usage and configuration data about your on-premises servers.</td>
<td>Receive notifications of events through AWS CloudTrail. For more information, see <a href="https://docs.aws.amazon.com/applicationdiscovery/latest/userguide/adcapi-tracking.html">Logging Application Discovery Service API calls with AWS CloudTrail</a> in the <em>Application Discovery Service User Guide</em>.</td>
</tr>
<tr>
<td>AWS Database Migration Service – Migrates data from on-premises databases into the AWS Cloud.</td>
<td>Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see <a href="https://docs.aws.amazon.com/databasemigration/latest/userguide/working-with-events-notifications.html">Working with events and notifications in AWS Database Migration Service</a> in the <em>AWS Database Migration Service User Guide</em>.</td>
</tr>
</tbody>
</table>
| AWS Snowball – 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:  
  * [Snowball notifications](https://docs.aws.amazon.com/snowball/latest/userguide/snowball-notification-types.html) in the *AWS Snowball User Guide*  
  * [Step 5: Choose your notification preferences](https://docs.aws.amazon.com/snowball/latest/edgeguide/step-5-snowball-edge-notification-preferences.html) in the *AWS Snowball Edge Developer Guide*  
  * [Step 5: Choose your notification preferences](https://docs.aws.amazon.com/snowcone/latest/userguide/step-5-snowcone-notification-preferences.html) in the *AWS Snowcone User Guide* |

# Networking & content delivery services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon API Gateway – Enables you to create and deploy your own REST and WebSocket APIs at any scale.</td>
<td>Receive messages posted to an API Gateway endpoint. For more information, see <a href="https://docs.aws.amazon.com/apigateway/latest/developerguide/tut-api-gateway-rest-api.html">Tutorial: Build an API Gateway REST API with AWS integration</a> in the <em>API Gateway Developer Guide</em>.</td>
</tr>
<tr>
<td>Amazon CloudFront – Speeds up distribution of your static and dynamic web content, such as .html, .css, .php, image, and media files.</td>
<td>Receive notifications when alarms based on specified CloudFront metrics occur. For more information, see <a href="https://docs.aws.amazon.com/cloudfront/latest/userguide/alarm-comprehend.html">Setting alarms to receive notifications</a> in the <em>Amazon CloudFront Developer Guide</em>.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Benefit of using with Amazon SNS</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>AWS Direct Connect</strong> – Links your internal network to an AWS Direct Connect location over a standard Ethernet fiber-optic cable.</td>
<td>Receive notifications when alarms that monitor the state of an AWS Direct Connect connection change state. For more information, see Creating CloudWatch alarms to monitor AWS Direct Connect connections in the AWS Direct Connect User Guide.</td>
</tr>
<tr>
<td><strong>Elastic Load Balancing</strong> – Automatically distributes your incoming traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in more or more Availability Zones.</td>
<td>Receive notifications of alarms you've created for load balancer events. For more information, see Create CloudWatch alarms for your load balancer in the User Guide for Classic Load Balancers.</td>
</tr>
<tr>
<td><strong>Amazon Route 53</strong> – Provides domain registration, DNS routing, and health checking.</td>
<td>Receive notifications when health check status is unhealthy. For more information, see To receive an Amazon SNS notification when a health check status is unhealthy (console) in the Amazon Route 53 Developer Guide.</td>
</tr>
<tr>
<td><strong>Amazon Virtual Private Cloud (Amazon VPC)</strong> – Enables you to launch AWS resources into a virtual network that you've defined.</td>
<td>Receive notifications for specific events that occur on interface endpoints. For more information, see Create and manage a notification for an endpoint service in the Amazon VPC User Guide.</td>
</tr>
</tbody>
</table>

### Security, identity, & compliance services

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

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon DynamoDB</strong> – Provides fast and predictable performance with seamless scalability in this fully managed NoSQL database service.</td>
<td>Receive notifications when maintenance events occur. For more information, see Customizing DAX cluster settings in the <em>Amazon DynamoDB Developer Guide</em>.</td>
</tr>
<tr>
<td><strong>Amazon EventBridge</strong> – 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.</td>
<td>Receive notifications of EventBridge events. For more information, see Amazon EventBridge targets in the <em>Amazon EventBridge User Guide</em>.</td>
</tr>
<tr>
<td><strong>AWS Lambda</strong> – Lets you run code without provisioning or managing servers.</td>
<td>Receive function output data by setting an SNS topic as a Lambda dead-letter queue or a Lambda destination. For more information, see Asynchronous invocation in the <em>AWS Lambda Developer Guide</em>.</td>
</tr>
</tbody>
</table>

### Storage services

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS Backup</strong> – Helps you centralize and automate the backup of data across AWS services in the cloud and on premises</td>
<td>Receive notifications of AWS Backup events. For more information, see Using Amazon SNS to track AWS Backup events in the <em>AWS Backup Developer Guide</em>.</td>
</tr>
<tr>
<td><strong>Amazon Elastic File System</strong> – Provides file storage for your Amazon EC2 instances.</td>
<td>Receive notifications of alarms you've created for Amazon EFS events. For more information, see Automated monitoring tools in the <em>Amazon Elastic File System User Guide</em>.</td>
</tr>
<tr>
<td><strong>Amazon S3 Glacier</strong> – Provides storage for infrequently used data.</td>
<td>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 Configuring vault notifications in Amazon S3 Glacier in the <em>Amazon S3 Glacier Developer Guide</em>.</td>
</tr>
<tr>
<td><strong>Amazon Simple Storage Service (Amazon S3)</strong> – Provides object storage.</td>
<td>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 Walkthrough: Configure a bucket for notifications (SNS topic or SQS queue) and Monitoring progress with replication metrics and Amazon S3 event notifications in the <em>Amazon Simple Storage Service User Guide</em>.</td>
</tr>
<tr>
<td><strong>AWS Snowball</strong> – Uses physical storage devices to transfer large amounts of data between Amazon S3 and your onsite data storage location at faster-than-internet speeds.</td>
<td>Receive notifications for Snowball jobs. For more information, see the following:</td>
</tr>
</tbody>
</table>
Additional event sources

<table>
<thead>
<tr>
<th>Source</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS IP address ranges</td>
<td>Receive notifications of changes to AWS IP ranges. For more information, see AWS IP address ranges notifications in the Amazon Web Services General Reference.</td>
</tr>
</tbody>
</table>

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. 153) and application-to-person (A2P) notifications (p. 217).

**Note**
Amazon SNS introduced FIFO topics (p. 43) 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

<table>
<thead>
<tr>
<th>Event destination</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Kinesis Data Firehose</td>
<td>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 Fanout to Kinesis Data Firehose delivery streams (p. 153).</td>
</tr>
<tr>
<td>AWS Lambda</td>
<td>Deliver events to functions for triggering the execution of custom business logic. For</td>
</tr>
</tbody>
</table>
A2P destinations

<table>
<thead>
<tr>
<th>Event destination</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon SQS</td>
<td>Deliver events to queues for application integration purposes. For more information, see Fanout to Amazon SQS queues (p. 174).</td>
</tr>
<tr>
<td>AWS Event Fork Pipelines</td>
<td>Deliver events to event backup and storage, event search and analytics, or event replay pipelines. For more information, see Fanout to AWS Event Fork Pipelines (p. 200).</td>
</tr>
<tr>
<td>HTTP/S</td>
<td>Deliver events to external webhooks. For more information, see Fanout to HTTP/S endpoints (p. 184).</td>
</tr>
</tbody>
</table>

### A2P destinations

<table>
<thead>
<tr>
<th>Event destination</th>
<th>Benefit of using with Amazon SNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS</td>
<td>Deliver events to mobile phones as text messages. For more information, see Mobile text messaging (SMS) (p. 217).</td>
</tr>
<tr>
<td>Email</td>
<td>Deliver events to inboxes as email messages. For more information, see Email notifications (p. 339).</td>
</tr>
<tr>
<td>Platform endpoint</td>
<td>Deliver events to mobile phones as native push notifications. For more information, see Mobile push notifications (p. 312).</td>
</tr>
</tbody>
</table>
| AWS Chatbot                     | Deliver events to Amazon Chime chat rooms or Slack channels. For more information, see the following pages in the AWS Chatbot Administrator Guide:  
  • Setting up AWS Chatbot with Amazon Chime  
  • Setting up AWS Chatbot with Slack  
  • Using AWS Chatbot with other AWS services |
| PagerDuty                       | Deliver operational insights to on-call teams. For more information, see Deliver ML-powered operational insights to your on-call teams via PagerDuty with Amazon DevOps Guru on the AWS Management & Governance Blog. |

**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.

   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. 437) section.
- Enable server-side encryption (p. 444) for a topic.
- Enable server-side encryption for a topic with an encrypted Amazon Simple Queue Service (Amazon SQS) queue (p. 445) subscribed.
- Subscribe AWS Event Fork Pipelines (p. 210) 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. 32)
- Deleting an Amazon SNS subscription and topic (p. 33)
- Amazon SNS topic tagging (p. 37)

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. 43), 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. 54).
5. (Optional) Expand the Encryption section and do the following. For more information, see Encryption at rest (p. 438).
   a. Choose Enable encryption.
   b. Specify the AWS KMS key. For more information, see Key terms (p. 439).

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

   Important
   If you aren't the owner of the KMS, 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 KMS on the Amazon SNS console.
   Ask the owner of the KMS 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 KMS 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 KMS for Amazon SNS for a topic, AWS KMS creates the AWS managed KMS for Amazon SNS.
   • Alternatively, the first time you use the Publish action on a topic with SSE enabled, AWS KMS creates the AWS managed KMS for Amazon SNS.

   • To use a custom KMS from your AWS account, choose the AWS KMS key field and then choose the custom KMS from the list.

   Note
   For instructions on creating custom KMSs, see Creating Keys in the AWS Key Management Service Developer Guide

   • To use a custom KMS ARN from your AWS account or from another AWS account, enter it into the AWS KMS key 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. 459) and Example cases for Amazon SNS access control (p. 470).

   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. 141).
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. 136).

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 *Amazon SNS topic tagging* (p. 37).

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:

    ```
    ```

### 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**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the **AWS Code Examples Repository**.

```csharp
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use Amazon Simple Notification Service 
/// (Amazon SNS) to add a new Amazon SNS topic. The example was created 
/// using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";

        IAmazonSimpleNotificationService client = new
        AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}"");
    }

    /// <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>
```
```csharp
/// <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;
}
```

- For API details, see CreateTopic in AWS SDK for .NET API Reference.

### C++

**SDK for C++**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);
```

- For API details, see CreateTopic in AWS SDK for C++ API Reference.

### Go

**SDK for Go V2**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
• For API details, see CreateTopic in AWS SDK for Go API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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 "";
}
```

• For API details, see CreateTopic in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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 () => {
```

28
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();

For more information, see AWS SDK for JavaScript Developer Guide.
For API details, see CreateTopic in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun createSNSTopic(topicName: String): String {
    val request = CreateTopicRequest {
        name = topicName
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}

For API details, see CreateTopic in AWS SDK for Kotlin API Reference.

PHP

SDK for PHP

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require '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:
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());
}

• 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)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

class 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

• For API details, see CreateTopic in AWS SDK for Python (Boto3) API Reference.
Ruby

SDK for Ruby

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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__
```

- 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.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```rust
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().unwrap_or_default()
    );
}
```
Subscribing to a topic

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. 32) 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. 184)
      • Email/Email-JSON (p. 339)
      • Amazon Kinesis Data Firehose (p. 153)
      • Amazon SQS (p. 174)
      Note
      To subscribe to an SNS FIFO topic (p. 43), choose this option.
      • AWS Lambda (p. 173)
      • Platform application endpoint (p. 312)
      • SMS (p. 217)
   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. 153).
   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. 129).
   f. (Optional) To configure a filter policy, expand the Subscription filter policy section. For more information, see Amazon SNS subscription filter policies (p. 79).
   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. 146).
   h. Choose Create subscription.
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. 33)
- To delete a subscription and topic using an AWS SDK (p. 33)

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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;

/// <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()
    {
        IAmazonSimpleNotificationService client = new AmazonSimpleNotificationServiceClient();

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

- For API details, see DeleteTopic in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Aws::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);

- For API details, see DeleteTopic in AWS SDK for C++ API Reference.
Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("Status was " +
            result.sdkHttpResponse().statusCode());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

For API details, see `DeleteTopic` in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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));
```
```javascript
console.log("Success.", data);
return data; // For unit tests.
} catch (err) {
    console.log("Error", err.stack);
}

run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](https://docs.aws.amazon.com/goto/WebAPI/sns/DeletionMessageFunctionOutcome).  
- For API details, see [DeleteTopic](https://docs.aws.amazon.com/goto/WebAPI/sns/DeleteTopic) in [AWS SDK for JavaScript API Reference](https://docs.aws.amazon.com/goto/WebAPI/sns/DeleteTopic).

### Kotlin

**SDK for Kotlin**

**Note**  
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**  
There’s more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-sdk-kotlin).

```kotlin
suspend fun deleteSNSTopic(topicArnVal: String) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was successfully deleted.")
    }
}
```

- For API details, see [DeleteTopic](https://docs.aws.amazon.com/goto/WebAPI/sns/DeleteTopic) in [AWS SDK for Kotlin API reference](https://docs.aws.amazon.com/goto/WebAPI/sns/DeleteTopic).

### PHP

**SDK for PHP**

**Note**  
There’s more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-sdk-php).

```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:  
 * guide_credentials.html
 */
```
Amazon SNS topic tagging

Amazon SNS supports tagging of Amazon SNS topics. This can help you track and manage the costs associated with your topics, provide enhanced security in your AWS Identity and Access Management (IAM) policies, and lets you easily search or filter through thousands of topics. Tagging enables you to manage your Amazon SNS topics using AWS Resource Groups. For more information on Resource Groups, see the AWS Resource Groups User Guide.
Tags

Topics
• Tagging for cost allocation (p. 38)
• Tagging for access control (p. 38)
• Tagging for resource searching and filtering (p. 39)
• Configuring Amazon SNS topic tags (p. 40)

Tagging for cost allocation

To organize and identify your Amazon SNS topics for cost allocation, you can add tags that identify the purpose of a topic. This is especially useful when you have many topics. You can use cost allocation tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill to include the tag keys and values. For more information, see Setting Up a Monthly Cost Allocation Report in the AWS Billing and Cost Management User Guide.

For example, you can add tags that represent the cost center and purpose of your Amazon SNS topics, as follows:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1</td>
<td>Cost Center</td>
<td>43289</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>Order processing</td>
</tr>
<tr>
<td>Topic 2</td>
<td>Cost Center</td>
<td>43289</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>Payment processing</td>
</tr>
<tr>
<td>Topic 3</td>
<td>Cost Center</td>
<td>76585</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>Archiving</td>
</tr>
</tbody>
</table>

This tagging scheme lets you to group two topics performing related tasks in the same cost center, while tagging an unrelated activity with a different cost allocation tag.

Tagging for access control

AWS Identity and Access Management supports controlling access to resources based on tags. After tagging your resources, provide information about your resource tags in the condition element of an IAM policy to manage tag-based access. For information on how to tag your resources using the Amazon SNS console (p. 40) or the AWS SDK (p. 40), see Configuring tags (p. 40).

You can restrict access for an IAM identity. For example, you can restrict Publish and PublishBatch access to all Amazon SNS topics that include a tag with the key environment and the value production, while allowing access to all other Amazon SNS topics. In the example below, the policy restricts the ability to publish messages to topics tagged with production, while allowing messages to be published to topics tagged with development. For more information, see Controlling Access Using Tags in the IAM User Guide.

Note

Setting the IAM permission for Publish sets permission for both Publish and PublishBatch.

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "sns:Publish",
    "Condition": {
      "StringMatch": {
        "sns:TagKeys": ["environment"]
      }
    }
  }
}]
```
Tagging for resource searching and filtering

An AWS account can have tens of thousands of Amazon SNS topics (see Amazon SNS Quotas for details). By tagging your topics, you can simplify the process of searching through or filtering out topics.

For example, you may have hundreds of topics associated with your production environment. Rather than having to manually search for these topics, you can query for all topics with a given tag:

```java
import com.amazonaws.services.resourcegroups.AWSResourceGroups;
import com.amazonaws.services.resourcegroups.AWSResourceGroupsClientBuilder;
import com.amazonaws.services.resourcegroups.model.QueryType;
import com.amazonaws.services.resourcegroups.model.ResourceQuery;
import com.amazonaws.services.resourcegroups.model.SearchResourcesRequest;
import com.amazonaws.services.resourcegroups.model.SearchResourcesResult;

public class Example {
    public static void main(String[] args) {
        // Query Amazon SNS Topics with tag "keyA" as "valueA"
        final String QUERY = "{"ResourceTypeFilters": ["AWS::SNS::Topic"],
                                "TagFilters" : [{"Key": "keyA", "Values": ["valueA"]}]};";

        // Initialize ResourceGroup client
        AWSResourceGroups awsResourceGroups = AWSResourceGroupsClientBuilder
                .standard()
                .build();

        // Query all resources with certain tags from ResourceGroups
        SearchResourcesResult result = awsResourceGroups.searchResources(
                new SearchResourcesRequest().withResourceQuery(
                        new ResourceQuery()
                                .withType(QueryType.TAG_FILTERS_1_0)
                                .withQuery(QUERY)
                ));

        System.out.println("SNS Topics with certain tags are " + result.getResourceIdentifiers());
    }
}
```
Configuring Amazon SNS topic tags

This page shows how you can use the AWS Management Console, an AWS SDK, and the AWS CLI to configure tags for an Amazon SNS topic (p. 37).

**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**
- Listing, adding, and removing tags for an Amazon SNS topic using the AWS Management Console (p. 40)
- Adding tags to a topic using an AWS SDK (p. 40)
- Managing tags with Amazon SNS API actions (p. 42)
- API actions that support ABAC (p. 42)

Listing, adding, and removing 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.

Adding 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 examples show how to add tags to an Amazon SNS topic.

Java

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public static void addTopicTags(SnsClient snsClient, String topicArn) {
    try {
        Tag tag = Tag.builder()
            .key("Team")
            .value("Development")
```
Tag tag2 = Tag.builder()
.longKey("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);
}

• For API details, see TagResource in AWS SDK for Java 2.x API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun addTopicTags(topicArn: String) {
val tag = Tag {
    key = "Team"
    value = "Development"
}
val tag2 = Tag {
    key = "Environment"
    value = "Gamma"
}
val tagList = mutableListOf<Tag>()
tagList.add(tag)
tagList.add(tag2)
val request = TagResourceRequest {
    resourceArn = topicArn
    tags = tagList
}

SnsClient { region = "us-east-1" }.use { snsClient ->
snsClient.tagResource(request) }
Managing tags with Amazon SNS API actions

To manage tags using the Amazon SNS API, use the following API actions:

- `ListTagsForResource`
- `TagResource`
- `UntagResource`

API actions that support ABAC

The following is a list of API actions that support attribute-based access control (ABAC). For more details about ABAC, see What is ABAC for AWS? in the IAM User Guide.

- `AddPermission`
- `ConfirmSubscription`
- `DeleteTopic`
- `GetSubscriptionAttributes`
- `GetTopicAttributes`
- `ListSubscriptionsByTopic`
- `ListTagsForResource`
- `Publish`
- `PublishBatch`
- `RemovePermission`
- `SetSubscriptionAttributes`
- `SetTopicAttributes`
- `Subscribe`
- `TagResource`
- `Unsubscribe`
- `UntagResource`
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. 43)
- Message ordering details for FIFO topics (p. 45)
- Message grouping for FIFO topics (p. 50)
- Message delivery for FIFO topics (p. 51)
- Message filtering for FIFO topics (p. 52)
- Message deduplication for FIFO topics (p. 54)
- Message security for FIFO topics (p. 56)
- Message durability for FIFO topics (p. 56)
- Code examples for FIFO topics (p. 58)

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. 45). For code snippets that implement this use case, see Code examples for FIFO topics (p. 58).
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. 43).
Note that there is no implied ordering of the subscribers. The following example shows that message \textit{m1} is delivered first to the wholesale subscriber and then to the retail subscriber. Message \textit{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. 146) 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. 129), 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. 50).
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. 51) and Message filtering for FIFO topics (p. 52).

In the auto parts price management example use case (p. 43), 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. 52) 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. 43), 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 \(m_1\) is String with a value of "wholesale". The attribute in message \(m_3\) is String.Array with a value of "wholesale,retail". Thus, both \(m_1\) and \(m_3\) 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 \(m_2\) is String with a value of "retail". The attribute in message \(m_3\) is String.Array with a value of "wholesale,retail". Thus, both \(m_2\) and \(m_3\) 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. 79).

SNS FIFO topics don't deliver duplicate messages to subscribed endpoints. For more information, see Message deduplication for FIFO topics (p. 54).
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. 52). 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. 50).

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. 43), 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. 56).
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. 438) 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. 447) 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. 56).

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 casues 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. 141).

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. 43), 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. 146) 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. 43) with SNS FIFO topics and SQS FIFO queues.

Topics

- Using an AWS SDK (p. 58)
- Using AWS CloudFormation (p. 60)

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. 79) 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. 52).

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

```java
public static void main(String[] args) {
    final String usage = "\n" +
    "Usage: " +
    "   </topicArn>\n" +
    "Where:\n" +
    "  fifoTopicName - The name of the FIFO topic. \n" +
    "  fifoQueueARN - The ARN value of a SQS FIFO queue. You can get this value from the AWS Management Console. \n";

    if (args.length != 2) {
```
```java
System.out.println(usage);
System.exit(1);
}

String fifoTopicName = "PriceUpdatesTopic3_fifo";
String fifoQueueARN = "arn:aws:sqs:us-east-1:814548047983:MyPriceSQS_fifo";
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();

createFIFO(snsClient, fifoTopicName, fifoQueueARN);
}

public static void createFIFO(SnsClient snsClient, String topicName, String queueARN) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = new HashMap<>();
        topicAttributes.put("FifoTopic", "true");
        topicAttributes.put("ContentBasedDeduplication", "false");

        CreateTopicRequest topicRequest = CreateTopicRequest.builder()
            .name(topicName)
            .attributes(topicAttributes)
            .build();

        CreateTopicResponse response = snsClient.createTopic(topicRequest);
        String topicArn = response.topicArn();
        System.out.println("The topic ARN is"+topicArn);

        // Subscribe to the endpoint by using the SNS service client.
        // Only Amazon SQS FIFO queues can receive notifications from an Amazon
        // SNS FIFO topic.
        SubscribeRequest subscribeRequest = SubscribeRequest.builder()
            .topicArn(topicArn)
            .endpoint(queueARN)
            .protocol("sqs")
            .build();

        snsClient.subscribe(subscribeRequest);
        System.out.println("The topic is subscribed to the queue.");

        // Compose and publish a message that updates the wholesale price.
        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";

        MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
            .dataType("String")
            .stringValue(attributeValue)
            .build();

        Map<String, MessageAttributeValue> attributes = new HashMap<>();
        attributes.put(attributeName, msgAttValue);

        PublishRequest pubRequest = PublishRequest.builder()
            .topicArn(topicArn)
            .subject(subject)
            .message(payload)
            .messageGroupId(groupId)
            .messageDeduplicationId(dedupId)
            .messageAttributes(attributes)
            .build();

        snsClient.publish(pubRequest);
    }
```
Receiving messages from FIFO subscriptions

You can now receive price updates in the wholesale and retail applications. As shown in the section called “FIFO topics use case” (p. 43), 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. 79) 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.

```json
{
  "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": "sqs",
"RawMessageDelivery": "false",
"FilterPolicy": {
"business": [
"wholesale"
]
}
}
},

"RetailSubscription": {
"Type": "AWS::SNS::Subscription",
"Properties": {
"TopicArn": {
"Ref": "PriceUpdatesTopic"
},
"Endpoint": {
"Fn::GetAtt": [
"RetailQueue",
"Arn"
],
"Protocol": "sqs",
"RawMessageDelivery": "false",
"FilterPolicy": {
"business": [
"retail"
]
}
}
},

"SalesQueuesPolicy": {
"Type": "AWS::SQS::QueuePolicy",
"Properties": {
"PolicyDocument": {
"Statement": [
{
"Effect": "Allow",
"Principal": {
"Service": "sns.amazonaws.com"
},
"Action": [}
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. 32) 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. 32).

Topics
- To publish messages to Amazon SNS topics using the AWS Management Console (p. 63)
- To publish a message to a topic using an AWS SDK (p. 64)
- Publishing large messages with Amazon SNS and Amazon S3 (p. 70)
- Amazon SNS message attributes (p. 73)
- Amazon SNS message batching (p. 76)

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. 43), 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. 336), 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. 322).
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. 79) (if present) before sending the message to the subscription.

For more information, see Amazon SNS message attributes (p. 73).

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

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see Publish in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);

- For API details, see Publish in AWS SDK for C++ API Reference.

### Go

**SDK for Go V2**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

### Java

**SDK for Java 2.x**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```

- For API details, see Publish in AWS SDK for Java 2.x API Reference.

### JavaScript

**SDK for JavaScript V3**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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();

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see Publish in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun pubTopic(topicArnVal: String, messageVal: String) {
  val request = PublishRequest {
    message = messageVal
    topicArn = topicArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    val result = snsClient.publish(request)
    println("${result.messageId} message sent.")
  }
}
- For API details, see Publish in AWS SDK for Kotlin API reference.

### PHP

**SDK for PHP**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 */

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

$message = 'This message is sent from an 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());
}
```

- 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)**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

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

• For API details, see Publish in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK for Ruby

Note
There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

require "aws-sdk-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__

• For more information, see AWS SDK for Ruby Developer Guide.
• For API details, see Publish in AWS SDK for Ruby 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:

```java
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

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

```java
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.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.CreateQueueRequest;
import com.amazonaws.services.sns.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);
    }
}
```
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:

```json
[
  "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 SQS subscriptions, a maximum of 10 message attributes can be sent when Raw Message Delivery (p. 129) is enabled. To send more than 10 message attributes, Raw Message Delivery must be disabled.
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. 63) 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. 79).

**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. 74) 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.

  **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`. A String.Array of number or boolean type does not require quotes.

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 $-10^{30}$ to $10^{30}$, 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:

<table>
<thead>
<tr>
<th>Push notification service</th>
<th>Reserved message attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM</td>
<td>AWS.SNS.MOBILE.ADM.TTL</td>
</tr>
<tr>
<td>APNs</td>
<td>AWS.SNS.MOBILE.APNS_MDM.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_MDM_SANDBOX.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_PASSBOOK.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_PASSBOOK_SANDBOX.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_SANDBOX.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_VOIP.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_VOIP_SANDBOX.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_COLLAPSE_ID</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_PRIORITY</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS_PUSH_TYPE</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS.TOPIC</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.APNS.TTL</td>
</tr>
<tr>
<td>Baidu</td>
<td>AWS.SNS.MOBILE.BAIDU.DeployStatus</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.BAIDU.MessageKey</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.BAIDU.MessageType</td>
</tr>
</tbody>
</table>
Amazon SNS message batching

What is message batching?

An alternative to publishing messages to either Standard or FIFO topics in individual Publish API requests, is using the Amazon SNS PublishBatch API to publish up to 10 messages in a single API request. Sending messages in batches can help you reduce the costs associated with connecting distributed applications (A2A messaging (p. 153)) or sending notifications to people (A2P messaging (p. 217)) with Amazon SNS by a factor of up to 10. Amazon SNS has quotas on how many messages you can publish to a topic per second based on the region in which you operate. See the Amazon SNS endpoints and quotas page in the AWS General Reference guide for more information on API quotas.

Note

The total aggregate size of all messages that you send in a single PublishBatch API request can't exceed 262,144 bytes (256 KB).

The PublishBatch API uses the same Publish API action for IAM policies.

How does message batching work?

Publishing messages with the PublishBatch API is similar to publishing messages with the Publish API. The main difference is that each message within a PublishBatch API request needs to be assigned a unique batch ID (up to 80 characters). This way, Amazon SNS can return individual API responses for every message within a batch to confirm that each message was either published or that a failure occurred. For messages being published to FIFO topics, in addition to including assigning a unique batch ID...

---

<table>
<thead>
<tr>
<th>Push notification service</th>
<th>Reserved message attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.BAIDU.TTL</td>
</tr>
<tr>
<td>FCM</td>
<td>AWS.SNS.MOBILE.FCM.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.GCM.TTL</td>
</tr>
<tr>
<td>macOS</td>
<td>AWS.SNS.MOBILE.MACOS_SANDBOX.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.MACOS.TTL</td>
</tr>
<tr>
<td>MPNS</td>
<td>AWS.SNS.MOBILE.MPNS.NotivationClass</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.MPNS.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.MPNS.Type</td>
</tr>
<tr>
<td>WNS</td>
<td>AWS.SNS.MOBILE.WNS.CachePolicy</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.Group</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.Match</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.SuppressPopup</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.Tag</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.TTL</td>
</tr>
<tr>
<td></td>
<td>AWS.SNS.MOBILE.WNS.Type</td>
</tr>
</tbody>
</table>
ID, you still need to include a MessageDeduplicationID and MessageGroupId for each individual message.

Examples

Publishing a batch of 10 messages to a Standard topic

```java
// Imports
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.model.PublishBatchRequest;
import com.amazonaws.services.sns.model.PublishBatchRequestEntry;
import com.amazonaws.services.sns.model.PublishBatchResult;
import com.amazonaws.services.sns.model.AmazonSNSException;
import java.util.List;
import java.util.stream.Collectors;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToTopic(AmazonSNS snsClient, String topicArn) {
    try {
        // Create the batch entries to send
        List<PublishBatchRequestEntry> entries = IntStream.range(0, MAX_BATCH_SIZE)
            .mapToObj(i -> new PublishBatchRequestEntry()
                .withId("id" + i)
                .withMessage("message" + i))
            .collect(Collectors.toList());

        // Create the batch request
        PublishBatchRequest request = new PublishBatchRequest()
            .withTopicArn(topicArn)
            .withPublishBatchRequestEntries(entries);

        // Publish the batch request
        PublishBatchResult publishBatchResult = snsClient.publishBatch(request);

        // Handle the successfully sent messages
        publishBatchResult.getSuccessful().forEach(publishBatchResultEntry -> {
            System.out.println("Batch Id for successful message: " +
                publishBatchResultEntry.getId());
            System.out.println("Message Id for successful message: " +
                publishBatchResultEntry.getMessageId());
        });

        // Handle the failed messages
        publishBatchResult.getFailed().forEach(batchResultErrorEntry -> {
            System.out.println("Batch Id for failed message: " +
                batchResultErrorEntry.getId());
            System.out.println("Error Code for failed message: " +
                batchResultErrorEntry.getCode());
            System.out.println("Sender Fault for failed message: " +
                batchResultErrorEntry.getSenderFault());
            System.out.println("Failure Message for failed message: " +
                batchResultErrorEntry.getMessage());
        });
    } catch (AmazonSNSException e) {
        // Handle any exceptions from the request
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```
Publishing a batch of 10 messages to a FIFO topic

```java
// Imports
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.model.PublishBatchRequest;
import com.amazonaws.services.sns.model.PublishBatchRequestEntry;
import com.amazonaws.services.sns.model.PublishBatchResult;
import com.amazonaws.services.sns.model.AmazonSNSException;
import java.util.List;
import java.util.stream.Collectors;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToFifoTopic(AmazonSNS snsClient, String topicArn) {
    try {
        // Create the batch entries to send
        List<PublishBatchRequestEntry> entries = IntStream.range(0, MAX_BATCH_SIZE)
            .mapToObj(i -> new PublishBatchRequestEntry()
                    .withId("id" + i)
                    .withMessage("message" + i)
                    .withMessageGroupId("groupId")
                    .withMessageDeduplicationId("deduplicationId" + i))
            .collect(Collectors.toList());

        // Create the batch request
        PublishBatchRequest request = new PublishBatchRequest()
            .withTopicArn(topicArn)
            .withPublishBatchRequestEntries(entries);

        // Publish the batch request
        PublishBatchResult publishBatchResult = snsClient.publishBatch(request);

        // Handle the successfully sent messages
        publishBatchResult.getSuccessful().forEach(publishBatchResultEntry -> {
            System.out.println("Batch Id for successful message: " +
                publishBatchResultEntry.getId());
            System.out.println("Message Id for successful message: " +
                publishBatchResultEntry.getMessageId());
            System.out.println("SequenceNumber for successful message: " +
                publishBatchResultEntry.getSequenceNumber());
        });

        // Handle the failed messages
        publishBatchResult.getFailed().forEach(batchResultErrorEntry -> {
            System.out.println("Batch Id for failed message: " +
                batchResultErrorEntry.getId());
            System.out.println("Error Code for failed message: " +
                batchResultErrorEntry.getCode());
            System.out.println("Sender Fault for failed message: " +
                batchResultErrorEntry.getSenderFault());
            System.out.println("Failure Message for failed message: " +
                batchResultErrorEntry.getMessage());
        });
    } catch (AmazonSNSException e) {
        // Handle any exceptions from the request
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```
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 applications that have fan-out system notification messages, Amazon SNS topics provide a logical access point that acts as a communication channel for related endpoints. We recommend using Amazon SNS subscription filter policies (p. 79) for exceptions in Application-to-application (A2A) (p. 153) communications.

For more information, see Filter Messages Published to Topics.

Topics
- Amazon SNS subscription filter policies (p. 79)
- Applying a subscription filter policy (p. 87)
- Removing a subscription filter policy (p. 90)

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. 79).

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

**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.

```json
{
  "Type": "Notification",
  "MessageId": "a1b2c34d-567e-8f90-g1h2-1345j67klmn8",
  "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. 73).
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.

```json
{
  "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.

```json
{
  "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:

```json
{
  "key_a": ["value_one", "value_two", "value_three"],
  "key_b": ["value_one"],
  "key_c": ["value_one", "value_two"]
}
```
Attribute string value matching

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

\[ 3 \times 1 \times 2 = 6 \]

- 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. To increase this quota, submit a quota increase request.
- For exceptions in Application-to-person (A2P) (p. 217) messages, we recommend using Amazon Pinpoint. Amazon Pinpoint allows the creation of dynamic segments based on selected attribute-based criteria, and sends messages through a variety of channels (email, push notifications, SMS, and custom channels) similar to Amazon SNS.

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:
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 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:

```bash
$ 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 above 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:

```bash
$ aws sns get-subscription-attributes --subscription-arn arn:aws:sns: ...
{
```

```json
```
AWS SDKs

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

**Important**
If you are using the SDK for Java 2.x example, the class SNSMessageFilterPolicy is not available out of the box. For instructions on how to install this class, see the example from the GitHub website.

**Java**

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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());
    }
}
```
For API details, see `SetSubscriptionAttributes` in *AWS SDK for Java 2.x API Reference*.

**Python**

**SDK for Python (Boto3)**

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```python
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):
        
        #: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
```

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 an AWS CloudFormation stack. For more information, see the `FilterPolicy` property of the
AWS Simple Notification Service Developer Guide
Removing a subscription filter policy

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.

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.
Message data protection

Topics

• What is message data protection? (p. 91)
• Why should I use message data protection? (p. 91)
• Understanding data protection policies (p. 91)
• Using managed data identifiers in Amazon SNS (p. 106)

What is message data protection?

Message data protection safeguards the data that's published to your Amazon SNS topics by using data protection policies (p. 91) to audit and block the sensitive information that moves between applications or AWS services.

Message data protection scans data in motion for personally identifiable information (PII) and protected health information (PHI) using predefined data identifiers (for example, names, addresses, credit card numbers, and prescription drug codes). Using the scanned information, message data protection provides detailed audit logs, and allows you to take action to protect that data.

Message data protection supports the following actions to help protect sensitive customer information:

• Audit – Audit up to 99% of the data that's published to an Amazon SNS topic. You can then choose to send the findings to Amazon CloudWatch, Amazon S3, or Amazon Kinesis Data Firehose.
• Deny – Block the transmission of data between applications and AWS resources if sensitive data is present within the payload.

Why should I use message data protection?

By introducing message data protection into your governance, risk management, and compliance programs, you can implement data protection policies that help you to identify and prevent data leakage. This provides your teams with tools that can help to reduce financial, legal, and regulatory risks by complying with privacy regulations such as HIPAA, GDPR, PCI, and FedRAMP. It also frees your developers from the operational overhead that's associated with building and managing your own tools to protect sensitive data.

For example, you can use message data protection to create an audit policy to determine whether any of your systems are inadvertently sending or receiving sensitive data. If your audit results show that systems are sending credit card information to systems that don't require it, you can use a block policy to prevent the delivery of the data.

Note

Amazon SNS supports message data protection for Amazon SNS standard topics only. Message data protection does not support the Amazon SNS PublishBatch API for inbound messages during preview. Message delivery for PublishBatch API is still protected.

Understanding data protection policies

Topics

• What are data protection policies? (p. 92)
What are data protection policies?

Amazon SNS uses data protection policies to select the sensitive data for which you want to scan, and the actions that you want to take to protect that data from being exchanged by your Amazon SNS topics. To select the sensitive data of interest, you use data identifiers (p. 106). Amazon SNS message data protection then detects the sensitive data by using machine learning and pattern matching. To act upon data identifiers that are found, you can define an audit or deny operation. These operations let you log the sensitive data that is found (or not found), or deny message delivery.

How is the data protection policy structured?

As illustrated in the following figure, a data protection policy document includes these elements:

- Optional policy-wide information at the top of the document
- One or more individual statements

Each statement includes information about a single permission.
Only one data protection policy can be defined per Amazon SNS topic. The data protection policy can have one or more deny statements, and one audit statement.

**JSON properties for the data protection policy**

A data protection policy requires the following basic policy information for identification:

- **Name** – The policy name.
- **Description** (Optional) – The policy description.
- **Version** – The policy language version. The current version is 2021-06-01.
- **Statement** – A list of statements that specifies data protection policy actions.

```
{
  "Name": "basicPII-protection",
  "Description": "Protect basic types of sensitive data",
  "Version": "2021-06-01",
  "Statement": [
    ...
  ]
}
```

**JSON properties for a policy statement**

A policy statement sets the detection context for the data protection operation.

- **Sid** (Optional) – The statement identifier.
- **DataDirection** – Inbound (for Publish API requests) or Outbound (for notification deliveries) with respect to the Amazon SNS topic.
- **DataIdentifier** – The sensitive data for which the Amazon SNS topic should scan. For example, name, address, or phone number.
- **Principal** – The IAM principal that published to the topic, or the IAM principal that subscribed to the topic.
- **Operation** – The follow-on action, either **Deny** (block) or **Audit**, which the Amazon SNS topic executes once it finds sensitive data.

```
{
  "Sid": "basicPII-inbound-protection",
  "DataDirection": "Inbound",
  "Principal": ["*"],
  "DataIdentifier": [,
    "arn:aws:dataprotection::aws:data-identifier/Name",
    "arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US"
  ],
  "Operation": {
    ...
  }
}
```

**JSON properties for a policy statement operation**

A policy statement sets one of the following data protection operations.

- **Deny** – Blocks the Amazon SNS publish request or fails the message delivery.
Audit – Emits metrics and finding logs without interrupting message publishing or delivery.

**Deny operation**

The **Deny** operation interrupts either the Publish API request or the delivery of the message if the message contains sensitive data. The Deny operation object is empty, as it doesn't require additional configuration.

```
"Operation": {
    "Deny": {}
}
```

On an inbound message, the SNS:publish API caller receives an authorization error.

Error code: AuthorizationError ...

On an outbound message, the Amazon SNS topic does not deliver the message to the subscription. To track unauthorized deliveries, enable the topic's delivery status logging (p. 136). The following is an example of a delivery status log:

```
{
  "notification": {
    "messageMD5Sum": "29638742ff68b32cf56f42a79bcf16b",
    "messageId": "34d9b400-c6dd-5444-820d-fbebf0f1f54cf",
    "timestamp": "2022-05-12T2:12:44Z"
  },
  "delivery": {
    "deliveryId": "98236591c-56aa-51ee-a5ed-0c7d43493170",
    "destination": "arn:aws:sqs:us-east-1:123456789012:NoNameAccess",
    "providerResponse": "The topic's data protection policy prohibits this message from being delivered to <subscription-arn>>",
    "dwellTimeMs":20,
    "attempts":1,
    "statusCode": 403
  },
  "status": "FAILURE"
}
```

**Note**

Mobile endpoints are not supported.

**Audit operation**

The **Audit** operation samples topic inbound messages, and logs the sensitive data findings in an AWS destination. The sample rate can be an integer between 0–99. This operation requires one of the following types of logging destinations:

1. **FindingsDestination** – The logging destination when the Amazon SNS topic finds sensitive data in the payload.
2. **NoFindingsDestination** – The logging destination when the Amazon SNS topic doesn't find sensitive data in the payload.

You can use the following AWS services in each of the following log destination types:

- **Amazon CloudWatch Logs** (Optional) – The LogGroup must be in the topic region and the name must start with `/aws/vendedlogs/`. 
Amazon Simple Notification Service Developer Guide
Overview of data protection policy structure

- **Amazon Kinesis Data Firehose** (Optional) – The DeliveryStream must be in the topic region and have Direct PUT as the source of delivery stream. For additional details, see Source, Destination, and Name in the Amazon Kinesis Data Firehose Developer Guide.

- **Amazon S3** (Optional) – An Amazon S3 bucket name.

```json
{
  "Operation": {
    "Audit": {
      "SampleRate": "99",
      "FindingsDestination": {
        "CloudWatchLogs": {
          "LogGroup": "/aws/vendedlogs/log-group-name"
        },
        "Firehose": {
          "DeliveryStream": "delivery-stream-name"
        },
        "S3": {
          "Bucket": "bucket-name"
        }
      },
      "NoFindingsDestination": {
        "CloudWatchLogs": {
          "LogGroup": "/aws/vendedlogs/log-group-name"
        },
        "Firehose": {
          "DeliveryStream": "delivery-stream-name"
        },
        "S3": {
          "Bucket": "bucket-name"
        }
      }
    }
  }
}
```

**Required permissions when specifying log destinations**

When you specify logging destinations in the data protection policy, you must add the following permissions to the IAM identity policy of the IAM principal that is calling the Amazon SNS PutDataProtectionPolicy API, or the CreateTopic API with the --data-protection-policy parameter.

<table>
<thead>
<tr>
<th>Audit destination</th>
<th>IAM permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>logs:CreateLogDelivery</td>
</tr>
<tr>
<td></td>
<td>logs:GetLogDelivery</td>
</tr>
<tr>
<td></td>
<td>logs:UpdateLogDelivery</td>
</tr>
<tr>
<td></td>
<td>logs:DeleteLogDelivery</td>
</tr>
<tr>
<td></td>
<td>logs:ListLogDeliveries</td>
</tr>
<tr>
<td>CloudWatchLogs</td>
<td>logs:PutResourcePolicy</td>
</tr>
<tr>
<td></td>
<td>logs:DescribeResourcePolicies</td>
</tr>
<tr>
<td></td>
<td>logs:DescribeLogGroups</td>
</tr>
</tbody>
</table>
### Audit destination log example

You can use `callerPrincipal` to identify the source of the sensitive content, and use the `messageID` as a reference to check against the Publish API response.
Audit operation metrics

When an audit operation has specified the FindingsDestination or the NoFindingsDestination property, the topic owners also receive CloudWatch MessagesWithFindings and MessagesWithNoFindings metrics.

How do I determine the IAM principals for my data protection policy?

Message data protection uses two IAM principals that interact with Amazon SNS.

1. **Publish API Principal** (Inbound) – The authenticated IAM principal calling the Amazon SNS Publish API.

2. **Subscription Principal** (Outbound) – The authenticated IAM principal that called the Subscribe API during subscription creation.

The SubscriptionPrincipal is a publicly available Amazon SNS subscription property that can be retrieved from the GetSubscriptionAttributes API.
Creating data protection policies to secure message data in Amazon SNS

Data protection policies (p. 91) help you safeguard the data that’s published to your Amazon SNS topics by auditing and blocking sensitive information that moves between applications or AWS services. You can use AWS API, AWS CLI, AWS CloudFormation, or AWS Management Console to create data protection policies in Amazon SNS. Only one policy can be defined per Amazon SNS topic. Each data protection policy can have one or more deny statements, however only one audit statement is supported for each data protection policy.

Topics
- Creating data protection policies to secure message data (API) (p. 98)
- Creating data protection policies to secure message data (CLI) (p. 99)
- Creating data protection policies to secure message data (CloudFormation) (p. 99)
- Creating data protection policies to secure message data (Console) (p. 99)
- Creating data protection policies to secure message data (SDK) (p. 101)

Creating data protection policies to secure message data (API)

The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

Creating data protection policies (AWS API)

You can create an Amazon SNS data protection policy using the AWS API.

To create a data protection policy together with an Amazon SNS topic (AWS API)

Use the DataProtectionPolicy property of a standard Amazon SNS topic:

- CreateTopic

To retrieve or create a data protection policy for an existing Amazon SNS topic (AWS API)

Call one of the following operations:

- GetDataProtectionPolicy
- PutDataProtectionPolicy
Creating data protection policies to secure message data (CLI)

The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

Creating data protection policies (AWS CLI)

You can create an Amazon SNS data protection policy using the AWS Command Line Interface.

To create a data protection policy together with an Amazon SNS topic (AWS CLI)

Use this option to create a new data protection policy together with a standard Amazon SNS topic:

• create-topic

To create or retrieve a data protection policy for an existing Amazon SNS topic (AWS CLI)

Call one of the following operations:

• get-data-protection-policy
• put-data-protection-policy

Creating data protection policies to secure message data (CloudFormation)

The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

Creating data protection policies (CloudFormation)

You can create an Amazon SNS data protection policy using AWS CloudFormation.

To create a data protection policy together with an Amazon SNS topic (CloudFormation)

Use this option to create a new data protection policy together with a standard Amazon SNS topic:

• AWS::SNS::Topic

Creating data protection policies to secure message data (Console)

The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

To create a data protection policy together with an Amazon SNS topic (Console)

Use this option to create a new data protection policy together with a standard Amazon SNS topic:

1. Sign in to the Amazon SNS console.
2. Choose a topic or create a new one. For more details on creating topics, see Creating an Amazon SNS topic (p. 24).
3. On the Create topic page, in the Details section, choose Standard.
   a. Enter a Name for the topic.
b. (Optional) Enter a **Display name** for the topic.

4. Expand **Data protection policy**.

5. Choose a **Configuration mode**:
   - **Basic** – Define a data protection policy using a simple menu.
   - **Advanced** – Define a custom data protection policy using JSON.

6. Choose the statement(s) that you’d like to add to your data protection policy. You can add both **Deny** and **Audit** statement types to the same data protection policy.

   a. **Add deny statement** – Configure which sensitive data to prevent from moving through your topic, and which principals to prevent from delivering that data.
      i. Select the **Data direction** of the messages to which this deny statement applies:
         - **Inbound messages** – Apply this deny statement to messages that are sent to the topic.
         - **Outbound messages** – Apply this deny statement to messages that the topic delivers to subscription endpoints.
      ii. Select the **data identifiers** to define the sensitive data that you want to deny.
      iii. Select the **IAM principals** to which this deny statement applies. You can apply it to **all AWS accounts**, or to **specific AWS accounts** or **IAM entities** (account roots, roles, or users) that use account IDs or IAM entity ARNs. Separate multiple IDs or ARNs using a comma (, ). The following **IAM principals** are supported:
         - **IAM account principals** (for example, arn:aws:iam::AWS-account-ID:root)
         - **IAM role principals** (for example, arn:aws:iam::AWS-account-ID:role/role-name)
         - **IAM user principals** (for example, arn:aws:iam::AWS-account-ID:user/user-name)
      iv. (Optional) Continue to add deny statements as needed.

   b. **Add audit statement** – Configure which sensitive data to audit, what percentage of messages you want to audit for that data, and where to send audit logs.

      **Note**
      Only one audit statement is allowed per data protection policy or topic.
      i. Select **data identifiers** to define the sensitive data that you want to audit.
      ii. For **Audit sample rate**, enter the percentage of messages to audit for sensitive information, up to a maximum of 99%.
      iii. For **Audit destination**, select which AWS services to send the audit finding results, and enter a destination name for each AWS service that you use. You can select from the following Amazon Web Services:
         - **Amazon CloudWatch** – CloudWatch Logs is the AWS standard logging solution. Using CloudWatch Logs, you can perform log analytics using Logs Insights (see samples here) and create metrics and alarms. CloudWatch Logs is where many services publish logs, which makes it easier to aggregate all logs using one solution. For information about Amazon CloudWatch, see the [Amazon CloudWatch User Guide](#).
         - **Amazon Kinesis Data Firehose** – Kinesis Data Firehose satisfies the demands for real-time streaming to Splunk, OpenSearch, and Amazon Redshift for further log analytics. For information about Amazon Kinesis Data Firehose, see the [Amazon Kinesis Data Firehose User Guide](#).
         - **Amazon Simple Storage Service** – Amazon S3 is an economical log destination for archival purposes. You may be required to retain logs for a period of years. In this case, you can put logs into Amazon S3 to save costs. For information about Amazon Simple Storage Service, see the [Amazon Simple Storage Service User Guide](#).
Creating data protection policies to secure message data (SDK)

The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

Creating data protection policies (AWS SDK)

You can create an Amazon SNS data protection policy using the AWS SDK.

To create a data protection policy together with an Amazon SNS topic (AWS SDK)

Use the following options to create a new data protection policy together with a standard Amazon SNS topic:

Java

```java
/**
 * For information regarding CreateTopic see this documentation topic:
 * https://docs.aws.amazon.com/code-samples/latest/catalog/javav2-sns-src-main-java-com-example-sns-CreateTopic.java.html
 */

public static String createSNSTopicWithDataProtectionPolicy(SnsClient snsClient, String topicName, String dataProtectionPolicy) {
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .dataProtectionPolicy(dataProtectionPolicy)
                .build();

        CreateTopicResponse result = snsClient.createTopic(request);
        return result.topicArn();
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

JavaScript

```javascript
// 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", DataProtectionPolicy: "DATA_PROTECTION_POLICY" };

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();
```
To create or retrieve a data protection policy for an existing Amazon SNS topic (AWS SDK)

Use the following options to create or retrieve a new data protection policy together with a standard Amazon SNS topic:

Java

```java
public static void putDataProtectionPolicy(SnsClient snsClient, String topicName,
    String dataProtectionPolicy) {
    try {
        PutDataProtectionPolicyRequest request =
            PutDataProtectionPolicyRequest.builder()
                .resourceArn(topicName)
                .dataProtectionPolicy(dataProtectionPolicy)
                .build();

        PutDataProtectionPolicyResponse result =
            snsClient.putDataProtectionPolicy(request);
        System.out.println("
        Status was "+ result.sdkHttpResponse().statusCode()
            + "\n        Topic " + request.resourceArn() + " DataProtectionPolicy "
            + request.dataProtectionPolicy());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```java
public static void getDataProtectionPolicy(SnsClient snsClient, String topicName) {
    try {
        GetDataProtectionPolicyRequest request =
            GetDataProtectionPolicyRequest.builder()
                .resourceArn(topicName)
                .build();

        GetDataProtectionPolicyResponse result =
            snsClient.getDataProtectionPolicy(request);
        System.out.println("
        Status is " + result.sdkHttpResponse().statusCode()
            + "\n        DataProtectionPolicy: \n        " + result.dataProtectionPolicy());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

JavaScript

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

// Set the parameters

const runPut = async () => {
    try {
        const data = await snsClient.send(new PutDataProtectionPolicyCommand(putParams));
        console.log("Success.", data);
    }
}
```

Deleting data protection policies in Amazon SNS

You can delete Amazon SNS data protection policies using the AWS API, AWS CLI, AWS CloudFormation, or AWS Management Console.

For general information about Amazon SNS data protection policies, see Understanding data protection policies (p. 91).

The number and size of Amazon SNS data protection policy resources in an AWS account are limited. For more information, see Amazon SNS API throttling in AWS General Reference.

Topics
- Deleting data protection policies (Console) (p. 103)
- Deleting a data protection policy using an empty JSON string (p. 103)
- Deleting a data protection policy using the AWS CLI (p. 103)

Deleting data protection policies (Console)

To delete a managed data protection policy (Console)

1. Sign in to the Amazon SNS console.
2. Choose the topic that contains the data protection policy that you want to delete.
3. Choose Edit.
4. Expand the Data protection policy section.
5. Choose Remove next to the data protection policy statement that you want to remove.
6. Choose Save changes.

Deleting a data protection policy using an empty JSON string

You can delete a data protection policy by updating it to an empty JSON string.

Deleting a data protection policy using the AWS CLI

You can delete a data protection policy using the AWS CLI.
Data protection policy examples

The following are examples of data protection policies that you can use to audit and deny sensitive data. For a complete tutorial that includes an example application, see the Introducing message data protection for Amazon SNS blog post.

Topics
- Example policy for auditing (p. 104)
- Example policy inbound deny statement (p. 105)
- Example policy for outbound deny statement (p. 105)

Example policy for auditing

Audit policies allow you to audit up to 99% of inbound messages and send findings to Amazon CloudWatch, Amazon Kinesis Data Firehose, and Amazon S3.

For example, you can create an audit policy to evaluate whether any of your systems are inadvertently sending or receiving sensitive data. If your audit results show that systems are sending credit card information to systems that don't require it, you can implement a data protection policy to block the delivery of the data.

The following example audits 99% of the messages that flow through the topic by looking for credit card numbers and sending the findings to CloudWatch Logs, Kinesis Data Firehose, and Amazon S3.

Data protection policy:

```json
{
    "Name": "__example_data_protection_policy",
    "Description": "Example data protection policy",
    "Version": "2021-06-01",
    "Statement": [
        {
            "DataDirection": "Inbound",
            "Principal": ["*"],
            "DataIdentifier": [
                "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
            ],
            "Operation": {
                "Audit": {
                    "SampleRate": "99",
                    "FindingsDestination": {
                        "CloudWatchLogs": {
                            "LogGroup": "<example log name>"
                        },
                        "Firehose": {
                            "DeliveryStream": "<example stream name>"
                        },
                        "S3": {
                            "Bucket": "<example bucket name>"
                        }
                    }
                }
            }
        }
    ]
}
```
Audit results format example:

```
{
    "messageId": "...",
    "callerPrincipal": "arn:aws:sts::123456789012:assumed-role/ExampleRole",
    "resourceArn": "arn:aws:sns:us-east-1:123456789012:ExampleArn",
    "dataIdentifiers": [
        {
            "name": "CreditCardNumber",
            "count": 1,
            "detections": [
                { "start": 1, "end": 2 }
            ]
        }
    ],
    "timestamp": "2021-04-20T00:33:40.241Z"
}
```

Example policy inbound deny statement

The following example blocks a user from publishing a message to a topic with CreditCardNumber in the message content. Denied payloads in the API response have a status code of "403 AuthorizationError".

```
{
    "Name": "__example_data_protection_policy",
    "Description": "Example data protection policy",
    "Version": "2021-06-01",
    "Statement": [
        {
            "DataDirection": "Inbound",
            "Principal": [
                "arn:aws:iam::123456789012:user/ExampleUser"
            ],
            "DataIdentifier": [
                "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
            ],
            "Operation": {
                "Deny": {}
            }
        }
    ]
}
```

Example policy for outbound deny statement

The following example blocks an AWS account from receiving messages that contain CreditCardNumber.

```
{
    "Name": "__example_data_protection_policy",
    "Description": "Example data protection policy",
    "Version": "2021-06-01",
    "Statement": [
        {
            "DataDirection": "Outbound",
            "Principal": [
                "arn:aws:iam::123456789012:user/ExampleUser"
            ],
            "DataIdentifier": [
```
"arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
],
"Operation": {
  "Deny": {}
}
]
]

Outbound deny results example, logged in Amazon CloudWatch:

{
  "notification": {
    "messageMD5Sum": "2e8f58ff2e6ed723b56b15493fb35a5",
    "messageId": "8747a956-ebf1-59da-b291-2c2e4b87c9c",
    "timestamp": "2022-09-08 15:40:57.144"
  },
  "delivery": {
    "deliveryId": "6a422437-78cc-5171-ad64-7fa3778507aa",
    "providerResponse": "The topic's data protection policy prohibits this message from being delivered to <subscription arn>",
    "dwellTimeMs": 22,
    "attempts": 1,
    "statusCode": 403
  },
  "status": "FAILURE"
}

Using managed data identifiers in Amazon SNS

Topics
- What are managed data identifiers? (p. 106)
- Sensitive data types: Credentials (p. 109)
- Sensitive data types: Devices (p. 110)
- Sensitive data types: Financial (p. 110)
- Sensitive data types: Protected health information (PHI) (p. 113)
- Sensitive data types: Personally identifiable information (PII) (p. 117)

What are managed data identifiers?

Amazon SNS uses a combination of criteria and techniques, including machine learning and pattern matching, to detect sensitive data. These criteria and techniques, collectively referred to as managed data identifiers, can detect a large and growing list of sensitive data types for many countries and regions, including multiple types of financial data, personal health information (PHI), and personally identifiable information (PII).

Each managed data identifier is designed to detect a specific type of sensitive data, such as credit card numbers, AWS secret access keys, or passport numbers for a particular country or region. When you create a data protection policy, you can configure Amazon SNS to use these identifiers to analyze messages going through the topic, and take actions when they are detected.

Amazon SNS can detect the following categories of sensitive data by using managed data identifiers:
What are managed data identifiers?

- Credentials, such as private keys or AWS secret access keys.
- Device identifiers, such as IP address or MAC address.
- Financial information, such as credit card numbers.
- Health information, for PHI such as health insurance or medical identification numbers.
- Personal information, for PII such as driver's licenses or social security numbers.

Within each category, Amazon SNS can detect multiple types of sensitive data. The topics in this section list and describe each type and any relevant requirements for detecting it. For each type, they also indicate the unique identifier (ID) for the managed data identifier that's designed to detect the data. When you create a data protection policy, you can use this ID to include the managed data identifier for message data protection to detect.

**Keyword requirements**

To detect certain types of sensitive data, Amazon SNS scans for keywords in proximity of the data. If this is the case for a particular type of data, a subsequent topic in this section indicates specific keyword requirements for that data.

Keywords aren't case sensitive. In addition, if a keyword contains a space, Amazon SNS automatically matches keyword variations that don't contain the space, or contain an underscore (_) or a hyphen (-) instead of the space. In certain cases, Amazon SNS also expands or abbreviates a keyword to address common variations of the keyword.

**Amazon SNS managed data identifiers for sensitive data types**

The following table lists and describes the types of credential, device, financial, medical, and personal health information (PHI) that Amazon SNS can detect using managed data identifiers. These are in addition to certain types of data that might also qualify as personally identifiable information (PII).

Region-dependent data identifiers require the identifier name with a dash, and the two letter (ISO 3166-1 alpha-2) codes. For example, DriversLicense-US.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Category</th>
<th>Countries/Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>BankAccountNumber</td>
<td>Financial</td>
<td>DE, ES, FR, GB, IT</td>
</tr>
<tr>
<td>CepCode</td>
<td>Personal</td>
<td>BR</td>
</tr>
<tr>
<td>Cnpj</td>
<td>Personal</td>
<td>BR</td>
</tr>
<tr>
<td>CpfCode</td>
<td>Personal</td>
<td>BR</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>Personal</td>
<td>AT, AU, BE, BG, CA, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IT, LT, LU, LV, MT, NL, PL, PT, RO, SE, SI, SK, US</td>
</tr>
<tr>
<td>DrugEnforcementAgencyNumber</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>ElectoralRollNumber</td>
<td>Personal</td>
<td>GB</td>
</tr>
<tr>
<td>HealthInsuranceCardNumber</td>
<td>Health</td>
<td>EU</td>
</tr>
<tr>
<td>HealthInsuranceClaimNumber</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>HealthInsuranceNumber</td>
<td>Health</td>
<td>FR</td>
</tr>
</tbody>
</table>
### What are managed data identifiers?

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Category</th>
<th>Countries/Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthcareProcedureCode</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>IndividualTaxIdentificationNumber</td>
<td>Personal</td>
<td>US</td>
</tr>
<tr>
<td>InseeCode</td>
<td>Personal</td>
<td>FR</td>
</tr>
<tr>
<td>MedicareBeneficiaryNumber</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>NationalDrugCode</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>NationalIdentificationNumber</td>
<td>Personal</td>
<td>DE, ES, IT</td>
</tr>
<tr>
<td>NationalInsuranceNumber</td>
<td>Personal</td>
<td>GB</td>
</tr>
<tr>
<td>NationalProviderId</td>
<td>Health</td>
<td>US</td>
</tr>
<tr>
<td>NhsNumber</td>
<td>Health</td>
<td>GB</td>
</tr>
<tr>
<td>NieNumber</td>
<td>Personal</td>
<td>ES</td>
</tr>
<tr>
<td>NifNumber</td>
<td>Personal</td>
<td>ES</td>
</tr>
<tr>
<td>PassportNumber</td>
<td>Personal</td>
<td>CA, DE, ES, FR, GB, IT, US</td>
</tr>
<tr>
<td>PermanentResidenceNumber</td>
<td>Personal</td>
<td>CA</td>
</tr>
<tr>
<td>PersonalHealthNumber</td>
<td>Health</td>
<td>CA</td>
</tr>
<tr>
<td>PhoneNumber</td>
<td>Personal</td>
<td>BR, DE, ES, FR, GB, IT, US</td>
</tr>
<tr>
<td>PostalCode</td>
<td>Personal</td>
<td>CA</td>
</tr>
<tr>
<td>RgNumber</td>
<td>Personal</td>
<td>BR</td>
</tr>
<tr>
<td>SocialInsuranceNumber</td>
<td>Personal</td>
<td>CA</td>
</tr>
<tr>
<td>Ssn</td>
<td>Personal</td>
<td>ES, US</td>
</tr>
<tr>
<td>TaxId</td>
<td>Personal</td>
<td>DE, ES, FR, GB</td>
</tr>
<tr>
<td>ZipCode</td>
<td>Personal</td>
<td>US</td>
</tr>
</tbody>
</table>

#### Supported Identifiers that are language/region independent

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Personal</td>
</tr>
<tr>
<td>AWS Secret Key</td>
<td>Credentials</td>
</tr>
<tr>
<td>CreditCardExpiration</td>
<td>Financial</td>
</tr>
<tr>
<td>CreditCardNumber</td>
<td>Financial</td>
</tr>
<tr>
<td>CreditCardSecurityCode</td>
<td>Financial</td>
</tr>
<tr>
<td>EmailAddress</td>
<td>Personal</td>
</tr>
<tr>
<td>IpAddress</td>
<td>Personal</td>
</tr>
<tr>
<td>Identifier</td>
<td>Category</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>LatLong</td>
<td>Personal</td>
</tr>
<tr>
<td>Name</td>
<td>Personal</td>
</tr>
<tr>
<td>OpenSshPrivateKey</td>
<td>Credentials</td>
</tr>
<tr>
<td>PgpPrivateKey</td>
<td>Credentials</td>
</tr>
<tr>
<td>PkcsPrivateKey</td>
<td>Credentials</td>
</tr>
<tr>
<td>PuttyPrivateKey</td>
<td>Credentials</td>
</tr>
<tr>
<td>VehicleIdentificationNumber</td>
<td>Personal</td>
</tr>
</tbody>
</table>

**Sensitive data types: Credentials**

The following table lists and describes the types of credentials that Amazon SNS can detect using managed data identifiers.

<table>
<thead>
<tr>
<th>Detection type</th>
<th>Managed data identifier ID</th>
<th>Keyword required</th>
<th>Countries and regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS secret access key</td>
<td>AwsSecretKey</td>
<td>aws_secret_access_key, credentials, secret access key, secret key, set-awscredential</td>
<td>Any</td>
</tr>
<tr>
<td>OpenSSH private key</td>
<td>OpenSshPrivateKey</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td>PGP private key</td>
<td>PgpPrivateKey</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td>Public-Key Cryptography Standard (PKCS) private key</td>
<td>PkcsPrivateKey</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td>PuTTY private key</td>
<td>PuttyPrivateKey</td>
<td>No</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Data identifier ARNs for credential data types**

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

<table>
<thead>
<tr>
<th>Credential data identifier ARNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/AwsSecretKey</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/OpenSshPrivateKey</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PgpPrivateKey</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PkcsPrivateKey</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PuttyPrivateKey</td>
</tr>
</tbody>
</table>
Sensitive data types: Devices

The following table lists and describes the types of device identifiers that Amazon SNS can detect using managed data identifiers.

<table>
<thead>
<tr>
<th>Detection type</th>
<th>Managed data identifier ID</th>
<th>Keyword required</th>
<th>Countries and regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Address</td>
<td>IpAddress</td>
<td>No</td>
<td>Any</td>
</tr>
</tbody>
</table>

Data identifier ARNs for device data types

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

<table>
<thead>
<tr>
<th>Device data identifier ARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/IpAddress</td>
</tr>
</tbody>
</table>

Sensitive data types: Financial

The following table lists and describes the types of financial information that Amazon SNS can detect using managed data identifiers.

<table>
<thead>
<tr>
<th>Detection type</th>
<th>Managed data identifier ID</th>
<th>Keyword required</th>
<th>Additional information</th>
<th>Countries and regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank account number</td>
<td>BankAccountNumber</td>
<td>Yes, see Keywords for bank account numbers (p. 111).</td>
<td>This includes: International Bank Account Numbers (IBANs) that consist of up to 34 alphanumeric characters, including elements such as country code.</td>
<td>France, Germany, Italy, Spain, UK</td>
</tr>
<tr>
<td>Credit card expiration date</td>
<td>CreditCardExpiration</td>
<td>exp d, exp m, exp y, expiration, expiry</td>
<td>–</td>
<td>Any</td>
</tr>
<tr>
<td>Credit card number</td>
<td>CreditCardNumber</td>
<td>account number, american express, amex, bank card, card, card num, card number, cc #, ccn, check card, credit, credit card#, dankort, debit,</td>
<td>Detection requires the data to be a 13–19 digit sequence that adheres to the Luhn check formula, and uses a standard card number prefix</td>
<td>Any</td>
</tr>
</tbody>
</table>
## Sensitive data types: Financial

<table>
<thead>
<tr>
<th>Detection type</th>
<th>Managed data identifier ID</th>
<th>Keyword required</th>
<th>Additional information</th>
<th>Countries and regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>debit card, diners club, discover,</td>
<td>for any of the following types of credit cards:</td>
<td>Amercian Express, Dankort, Diner’s Club, Discover, Electron,</td>
<td>Any</td>
<td></td>
</tr>
<tr>
<td>electron, japanese card bureau, jcb</td>
<td>debit card, discover, electron, japanese card bureau, jcb, mastercard, mc, pan, payment</td>
<td>Japanese Card Bureau (JCB), Mastercard, UnionPay, and Visa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>account number, payment card number,</td>
<td></td>
<td>(superscript link below 1).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pcn, union pay, visa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit card verification code</td>
<td>CreditCardSecurityCode, credit card id, card identification code, card identification code,</td>
<td>–</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>card identification number, card security code, card validation code, card validation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>number, card verification data, card verification value, cvc, cvc2, cvv, cvv2, elo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>verification code</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Amazon SNS doesn't report occurrences of the following sequences, which credit card issuers have reserved for public testing:

   1220000000000003, 2222405343248877, 2222990905257051, 2223007648726984, 2223577120017656, 30569309025904, 343434344343434, 3528000700000000, 35301133330000, 3566002020360505, 36148900647913, 36701020000000, 371449635398431, 378282246310005, 378734493671000, 38520000023237, 4012888888881881, 411111111111111, 422222222222, 4444333322221111, 4462030000000000, 4484070000000000, 4911830000000000, 4917300800000000, 4917610000000000, 4917610000000000, 5019717010103742, 510510510510510, 5111010030175156, 5185540810000019, 5200828282828210, 5204230080000017, 5204740009900014, 5420923878724339, 5454545454545454, 5506900460000018, 55069004900000436, 55069004900000444, 5506900500000234, 5506920809243667, 55069224010634930, 5506927427317625, 553042241984105, 555555000048194, 555555555555444, 5610591081018250, 6011000990139424, 6011000400000000, 60111111111111, 630490017740292441, 6304950600000000, 6331101999990016, 6759649826438453, 679999010000000019, and 76009244561.

**Keywords for bank account numbers**

Use the following keywords to detect International Bank Account Numbers (IBANs) that consist of up to 34 alphanumeric characters, including elements such as country code.
<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>account code, account number, accountno#, accountnumber#, bban, code bancaire, compte bancaire, customer account id, customer account number, customer bank account id, iban, numéro de compte</td>
</tr>
<tr>
<td>Germany</td>
<td>account code, account number, accountno#, accountnumber#, bankleitzahl, bban, customer account id, customer account number, customer bank account id, geheimzahl, iban, kartennummer, kontonummer, kreditkartennummer, sepa</td>
</tr>
<tr>
<td>Italy</td>
<td>account code, account number, accountno#, accountnumber#, bban, codice bancario, conto bancario, customer account id, customer account number, customer bank account id, iban, numero di conto</td>
</tr>
<tr>
<td>Spain</td>
<td>account code, account number, accountno#, accountnumber#, bban, código cuenta, código cuenta bancaria, cuenta cliente id, customer account ID, customer account number,</td>
</tr>
</tbody>
</table>
### Sensitive data types: Protected health information (PHI)

The following table lists and describes the types of protected health information (PHI) that Amazon SNS can detect using managed data identifiers.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>customer bank account id, iban, número cuenta bancaria cliente, número cuenta cliente</td>
</tr>
<tr>
<td>UK</td>
<td>account code, account number, accountno#, accountnumber#, bban, customer account id, customer account number, customer bank account id, iban, sepa</td>
</tr>
</tbody>
</table>

### Data identifier ARNs for financial data types

The following lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

<table>
<thead>
<tr>
<th>Financial data identifier ARNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-IT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/CreditCardExpiration</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/CreditCardNumber</td>
</tr>
<tr>
<td>Detection type</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>Drug Enforcement Agency (DEA) Registration Number</td>
</tr>
<tr>
<td>Health Insurance Card Number (EHIC)</td>
</tr>
<tr>
<td>Health Insurance Claim Number (HICN)</td>
</tr>
<tr>
<td>Health insurance or medical identification number</td>
</tr>
</tbody>
</table>
Keywords for health insurance and medical identification numbers

To detect various types of health insurance and medical identification numbers, Amazon SNS requires a keyword to be in proximity of the numbers. This includes European Health Insurance Card numbers (EU, Finland), health insurance numbers (France), Medicare Beneficiary Identifiers (US), National Insurance numbers (UK), NHS numbers (UK), and Personal Health Numbers (Canada).

The following table lists the keywords that Amazon SNS recognizes for specific countries and regions.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>Canada healthcare number, msp number, personal healthcare number, phn, soins de santé</td>
</tr>
<tr>
<td>EU</td>
<td>assicurazione sanitaria numero, carta assicurazione numero, carte d’assurance maladie, carte européenne d’assurance maladie, ceam, ehic, ehic#, finlandehicnumber#, gesundheitskarte, hälsokort, health card, health card number, health insurance card, health insurance number, insurance card number, krankenversicherungskarte, krankenversicherungsnummer, medical account number, numero conto medico, numero d’assurance maladie, numéro de carte d’assurance, numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanhoitokortin,</td>
</tr>
<tr>
<td>Country or region</td>
<td>Keywords</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Finland</td>
<td>ehic, ehic#, finland health insurance card, finlandehicnumber#, finska sjukförsäkringskort, hälsokort, health card, health card number, health insurance card, health insurance number, sairaanhoitokortin, sairaanhoitokortin, sairausvakuutuskortti, sairausvakuutusnumero, sjukförsäkring numero, sjukförsäkringskort, suomen sairausvakuutuskortti, suomi ehic-numero, terveyskortti</td>
</tr>
<tr>
<td>France</td>
<td>carte d'assuré social, carte vitale, insurance card</td>
</tr>
<tr>
<td>UK</td>
<td>national health service, NHS</td>
</tr>
<tr>
<td>US</td>
<td>mbi, medicare beneficiary</td>
</tr>
</tbody>
</table>

**Data identifier ARNs for protected health information data types (PHI)**

The following lists the data identifier Amazon Resource Names (ARNs) that can be used in PHI data protection policies.

<table>
<thead>
<tr>
<th>PHI data identifier ARNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DrugEnforcementAgencyNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/HealthInsuranceCardNumber-EU</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/HealthInsuranceClaimNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/HealthInsuranceNumber-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/MedicareBeneficiaryNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NationalInsuranceNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NationalProviderId-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NhsNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PersonalHealthNumber-CA</td>
</tr>
</tbody>
</table>
## Sensitive data types: Personally identifiable information (PII)

The following table lists and describes the types of personally identifiable information (PII) that Amazon SNS can detect using managed data identifiers.

<table>
<thead>
<tr>
<th>Detection type</th>
<th>Managed data identifier ID</th>
<th>Keyword required</th>
<th>Additional information</th>
<th>Countries and regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Código de Endereçamento Postal (CEP)</td>
<td>CepCode</td>
<td>cep, código de endereçamento postal, codigo de endereçamento postal</td>
<td>–</td>
<td>Brazil</td>
</tr>
<tr>
<td>Cadastro Nacional da Pessoa Jurídica (CNPJ)</td>
<td>Cnpj</td>
<td>cadastro nacional da pessoa jurídica, cadastro nacional da pessoa jurídica, cnpj</td>
<td>–</td>
<td>Brazil</td>
</tr>
<tr>
<td>Cadastro de Pessoas Físicas (CPF)</td>
<td>CpfCode</td>
<td>Cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro de pessoa física, cadastro de pessoa física, cpf</td>
<td>–</td>
<td>Brazil</td>
</tr>
<tr>
<td>Driver's license identification number</td>
<td>DriversLicense</td>
<td>Yes, see Keywords for driver's license identification numbers (p. 122).</td>
<td>–</td>
<td>Australia, Austria, Belgium, Bulgaria, Canada, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, UK, US</td>
</tr>
<tr>
<td>Electoral roll number</td>
<td>ElectoralRollNumber</td>
<td>electoral #, electoral number, electoral roll #, electoral roll no., electoral roll number, electoralrollno</td>
<td>–</td>
<td>UK</td>
</tr>
<tr>
<td>Detection type</td>
<td>Managed data identifier ID</td>
<td>Keyword required</td>
<td>Additional information</td>
<td>Countries and regions</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
<td>-----------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Individual taxpayer identification</td>
<td>IndividualTaxIdentificationNumber</td>
<td>Yes, see Keywords for taxpayer identification and reference numbers (p. 125).</td>
<td>–</td>
<td>US</td>
</tr>
<tr>
<td>National Institute for Statistics and Economic Studies (INSEE)</td>
<td>InseeCode</td>
<td>Yes, see Keywords for national identification numbers (p. 124).</td>
<td>–</td>
<td>France</td>
</tr>
<tr>
<td>National identification number</td>
<td>NationalIdentificationNumber</td>
<td>Yes, see Keywords for national identification numbers (p. 124).</td>
<td>This includes Documento Nacional de Identidad (DNI) identifiers (Spain), Codice fiscale codes (Italy), and National Identity Card numbers (German).</td>
<td>Germany, Italy, Spain</td>
</tr>
<tr>
<td>National Insurance Number (NINO)</td>
<td>NationalInsuranceNumber</td>
<td>insurance no., insurance number, insurance#, national insurance number, nationalinsurance#, nationalinsurancenumber, nin, nino</td>
<td>–</td>
<td>UK</td>
</tr>
<tr>
<td>Número de identidad de extranjero (NIE)</td>
<td>NieNumber</td>
<td>Yes, see Keywords for taxpayer identification and reference numbers (p. 125).</td>
<td>–</td>
<td>Spain</td>
</tr>
<tr>
<td>Número de Identificación Fiscal (NIF)</td>
<td>NifNumber</td>
<td>Yes, see Keywords for taxpayer identification and reference numbers (p. 125).</td>
<td>–</td>
<td>Spain</td>
</tr>
<tr>
<td>Passport number</td>
<td>PassportNumber</td>
<td>Yes, see Keywords for passport numbers (p. 125).</td>
<td>–</td>
<td>Canada, France, Germany, Italy, Spain, UK, US</td>
</tr>
<tr>
<td>Detection type</td>
<td>Managed data identifier ID</td>
<td>Keyword required</td>
<td>Additional information</td>
<td>Countries and regions</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Permanent residence number</td>
<td>PermanentResidenceNumber</td>
<td>permanent, numéro carte résident permanent, numéro résident permanent, permanent resident card, permanent resident card number, permanent resident no, permanent resident no., permanent resident number, pr no, pr no., pr non, pr number, résident permanent no., résident permanent non</td>
<td>–</td>
<td>Canada</td>
</tr>
<tr>
<td>Phone number</td>
<td>PhoneNumber</td>
<td>Brazil: keywords also include: cel, celular, fone, móvil, número residencial, numero residencial, telefone Others: cell, contact, fax, fax number, mobile, phone, phone number, tel, telephone, telephone number</td>
<td>This includes toll-free numbers in the US and fax numbers. If a keyword is in proximity of the data, the number doesn’t have to include a country code. If a keyword isn’t in proximity of the data, the number has to include a country code.</td>
<td>Brazil, Canada, France, Germany, Italy, Spain, UK, US</td>
</tr>
<tr>
<td>Postal Code</td>
<td>PostalCode</td>
<td>No</td>
<td>–</td>
<td>Canada</td>
</tr>
<tr>
<td>Registro Geral (RG)</td>
<td>RgNumber</td>
<td>Yes, see Keywords for national identification numbers (p. 124)</td>
<td>–</td>
<td>Brazil</td>
</tr>
<tr>
<td>Detection type</td>
<td>Managed data identifier</td>
<td>Keyword required</td>
<td>Additional information</td>
<td>Countries and regions</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Social Insurance Number (SIN)</td>
<td>SocialInsuranceNumber</td>
<td>canadian id, numéro d'assurance sociale, social insurance number, sin</td>
<td>--</td>
<td>Canada</td>
</tr>
<tr>
<td>Social Security number (SSN)</td>
<td>Ssn</td>
<td>Spain – número de la seguridad social, social security no., social security number, socialsecurityno#, ssn, ssn#</td>
<td>--</td>
<td>Spain, US</td>
</tr>
<tr>
<td>Taxpayer identification or reference number</td>
<td>TaxId</td>
<td>Yes, see Keywords for taxpayer identification and reference numbers (p. 125).</td>
<td>This includes TIN (France); Steueridentifikationsnummer (Germany); CIF (Spain); and TRN, UTR (UK).</td>
<td>France, Germany, Spain, UK</td>
</tr>
<tr>
<td>US postal code</td>
<td>ZipCode</td>
<td>zip code, zip+4</td>
<td>--</td>
<td>US</td>
</tr>
<tr>
<td>Mailing address</td>
<td>Address</td>
<td>No</td>
<td>Although a keyword isn't required, detection requires the address to include the name of a city or place and a ZIP or Postal Code.</td>
<td>Australia, Canada, France, Germany, Italy, Spain, UK, US</td>
</tr>
<tr>
<td>Electronic mail address</td>
<td>EmailAddress</td>
<td>No</td>
<td>--</td>
<td>Any</td>
</tr>
<tr>
<td>Detection type</td>
<td>Managed data identifier ID</td>
<td>Keyword required</td>
<td>Additional information</td>
<td>Countries and regions</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------</td>
<td>-------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Global Positioning System (GPS)</td>
<td>LatLong</td>
<td>coordinate, coordinates, lat long, latitude longitude, position</td>
<td>Amazon SNS can detect GPS coordinates if the latitude and longitude coordinates are stored as a pair and they're in Decimal Degrees (DD) format, for example, 41.948614,-87.655311. Support doesn't include coordinates in Degrees Decimal Minutes (DDM) format, for example 41°56.9168'N 87°39.3187'W, or Degrees, Minutes, Seconds (DMS) format, for example 41°56'55.0104&quot;N 87°39'19.1196&quot;W.</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full name</td>
<td>Name</td>
<td>No</td>
<td>Amazon SNS can detect full names only. Support is limited to Latin character sets.</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vehicle identification number (VIN)</td>
<td>VehicleIdentification</td>
<td>Fahrgestellnummer, niv, numarul de identificare, numarul seriei de sasiu, numer VIN, Número de Identificação do Veículo, Número de Identificación de Automóviles, numéro d'identification du véhicule, vehicle identification number, vin, VIN numeris</td>
<td>Amazon SNS can detect VINs that consist of a 17-character sequence and adhere to the ISO 3779 and 3780 standards. These standards were designed for worldwide use.</td>
<td>Any</td>
</tr>
</tbody>
</table>
Keywords for driver’s license identification numbers

To detect various types of driver’s license identification numbers, Amazon SNS requires a keyword to be in proximity of the numbers. The following table lists the keywords that Amazon SNS recognizes for specific countries and regions.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>dl#, dl:, dlno#, driver licence, driver license, driver permit, drivers lic., drivers licence, driver's licence, drivers license, driver's license, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit</td>
</tr>
<tr>
<td>Austria</td>
<td>führerschein, fuhrerschein, führerschein republik österreich, fuhrerschein republik oesterreich</td>
</tr>
<tr>
<td>Belgium</td>
<td>fuehrerschein, fuehrerschein-nr, fuehrerscheinnummer, fuhrerschein, führerschein, fuehrerschein-nr, führerscheinnummer, fuehrerscheinnummer, numéro permis conduire, permis de conduire, rijbewijs, rijbewijsnummer</td>
</tr>
<tr>
<td>Bulgaria</td>
<td>превозно средство, свидетелство за управление на моторно, свидетелство за управление на мпс, сумпс, шофьорска книжка</td>
</tr>
<tr>
<td>Canada</td>
<td>dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver's licence, drivers licence, driver's license, drivers license, driver's permit, drivers permit, driver's permit number, driving licence, driving license, driving permit, permis de conduire</td>
</tr>
<tr>
<td>Croatia</td>
<td>vozačka dozvola</td>
</tr>
<tr>
<td>Cyprus</td>
<td>άδεια οδήγησης</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>číslo licence, číslo licence řidiče, číslo řidičského průkazu, ovladače lic., povolení k jízdě, povolení řidiče, řidiči povolení, řidičský průkaz</td>
</tr>
<tr>
<td>Denmark</td>
<td>kørekort, kørekortnummer</td>
</tr>
<tr>
<td>Estonia</td>
<td>juhi litsentsi number, juhiloa number, juhiluba, juhiluba number</td>
</tr>
<tr>
<td>Finland</td>
<td>ajokortin numero, ajokortti, förare lic., körkort, körkort nummer, kuljettaja lic., permis de conduire</td>
</tr>
<tr>
<td>France</td>
<td>permis de conduire</td>
</tr>
<tr>
<td>Germany</td>
<td>fuehrerschein, fuehrerschein-nr, fuehrerscheinnummer, fuhrerschein,</td>
</tr>
<tr>
<td>Country or region</td>
<td>Keywords</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Greece</td>
<td>δεια οδήγησης, adeia odigisis</td>
</tr>
<tr>
<td>Hungary</td>
<td>illesztőprogramok lic, jogosítvány, jogsi, licencszám, vezető engedély, vezetői engedély</td>
</tr>
<tr>
<td>Ireland</td>
<td>ceadúnas tioamána</td>
</tr>
<tr>
<td>Italy</td>
<td>patente di guida, patente di guida numero, patente guida, patente guida numero</td>
</tr>
<tr>
<td>Latvia</td>
<td>autovadītāja apliecība, licences numurs, vadītāja apliecība, vadītāja apliecības numurs, vadītāja atļauja, vadītāja licences numurs, vadītāji lic.</td>
</tr>
<tr>
<td>Lithuania</td>
<td>vairuotojo pažymėjimas</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>fahrerlaubnis, führerschäin</td>
</tr>
<tr>
<td>Malta</td>
<td>ličenzja tas-sewqan</td>
</tr>
<tr>
<td>Netherlands</td>
<td>permis de conduire, rijbewijs, rijbewijsnummer</td>
</tr>
<tr>
<td>Poland</td>
<td>numer licencyjny, prawo jazdy, zezwolenie na prowadzenie</td>
</tr>
<tr>
<td>Portugal</td>
<td>carta de condução, carteira de habilitação, carteira de motorist, carteira habilitação, carteira motorist, licença condução, licença de condução, número de licença, número licença, permissão condução, permissão de condução</td>
</tr>
<tr>
<td>Romania</td>
<td>numărul permisului de conducere, permis de conducere</td>
</tr>
<tr>
<td>Slovakia</td>
<td>číslo licencie, číslo vodičského preukazu, ovládače lic., povolenia vodičov, povolenie jazdu, povolenie na jazdu, povolenie vodiča, vodičský preukaz</td>
</tr>
<tr>
<td>Slovenia</td>
<td>vozniško dovoljenje</td>
</tr>
<tr>
<td>Spain</td>
<td>carnet conductor, el carnet de conductor, licencia conductor, licencia de manejo, número carnet conductor, número de carnet de conductor, número de permiso conductor, número de permiso de conductor, número licencia conductor, número permiso conductor, permiso conducción, permiso conducer, permiso de conducción</td>
</tr>
<tr>
<td>Sweden</td>
<td>ajokortin numero, dlno# ajokortti, driveer lic., förare lic., körkort, körkort nummer, körkortsnummer, kuljettaajat lic.</td>
</tr>
</tbody>
</table>
### Keywords for national identification numbers

To detect various types of national identification numbers, Amazon SNS requires a keyword to be in close proximity to the numbers. This includes Documento Nacional de Identidad (DNI) identifiers (Spain), French National Institute for Statistics and Economic Studies (INSEE) codes, German National Identity Card numbers, and Registro Geral (RG) numbers (Brazil).

The following table lists the keywords that Amazon SNS recognizes for specific countries and regions.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>registro geral, rg</td>
</tr>
<tr>
<td>France</td>
<td>assurance sociale, carte nationale d'identité, cni, code sécurité sociale, French social security number, fssn#, insee, insurance number, national id number, nationalid#, numéro d’assurance, sécurité sociale, sécurité sociale non., sécurité sociale numéro, social, social security, social security number, socialsecuritynumber, ss#, ssn, ssn#</td>
</tr>
<tr>
<td>Germany</td>
<td>ausweisnummer, id number, identification number, identity number, insurance number, personal id, personalausweis</td>
</tr>
<tr>
<td>Italy</td>
<td>codice fiscal, dati anagrafici, ehic, health card, health insurance card, p. iva, partita i.v.a., personal data, tax code, tessera sanitaria</td>
</tr>
<tr>
<td>Spain</td>
<td>dni, dni#, dninúmero#, documento nacional de identidad, identidad único, identidadúnico#, insurance number, national identification number, national identity, nationalid#, nationalidno#, número nacional identidad, personal identification number, personal identity no, unique identity number, uniqueid#</td>
</tr>
</tbody>
</table>
Keywords for passport numbers

To detect various types of passport numbers, Amazon SNS requires a keyword to be in proximity of the numbers. The following table lists the keywords that Amazon SNS recognizes for specific countries and regions.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>passeport, passeport#, passport, passport#, passportno, passportno#</td>
</tr>
<tr>
<td>France</td>
<td>numéro de passeport, passeport #, passeport n °, passeport non</td>
</tr>
<tr>
<td>Germany</td>
<td>ausstellungsdatum, ausstellungsort, geburtsdatum, passport, passports, reisepass, reisepass-nr, reisepassnummer</td>
</tr>
<tr>
<td>Italy</td>
<td>italian passport number, numéro passeport, numéro passeport italien, passaporto, passaporto italiano, passaporto numero, passport number, repubblica italiana passaporto</td>
</tr>
<tr>
<td>Spain</td>
<td>españa pasaporte, libreta pasaporte, número pasaporte, pasaporte, passport, passport book, passport no, passport number, spain passport</td>
</tr>
<tr>
<td>UK</td>
<td>passeport #, passeport n °, passeport non, passeportn °, passport #, passport no, passport number, passport#, passportid</td>
</tr>
<tr>
<td>US</td>
<td>passport, travel document</td>
</tr>
</tbody>
</table>

Keywords for taxpayer identification and reference numbers

To detect various types of taxpayer identification and reference numbers, Amazon SNS requires a keyword to be in proximity of the numbers. The following table lists the keywords that Amazon SNS recognizes for specific countries and regions.

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>cadastro de pessoa física, cadastro de pessoa física, cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro nacional da pessoa jurídica, cadastro nacional da pessoa jurídica, cnpj, cpf</td>
</tr>
<tr>
<td>France</td>
<td>numéro d'identification fiscal, tax id, tax identification number, tax number, tin, tin#</td>
</tr>
<tr>
<td>Germany</td>
<td>identifikationsnummer, steuer id, steueridentifikationsnummer, steuernummer, tax id, tax identification number, tax number</td>
</tr>
</tbody>
</table>
| Spain             | cif, cif número, cifnúmero#, nie, nif, número de contribuyente, número de identidad de extranjero, número de identificación fiscal, número de
## Sensitive data types: Personally identifiable information (PII)

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>impuesto corporativo, personal tax number, tax id, tax identification number, tax number, tin, tin#</td>
</tr>
<tr>
<td>UK</td>
<td>paye, tax id, tax id no., tax id number, tax identification, tax identification#, tax no., tax number, tax reference, tax#, taxa#d#, temporary reference number, tin, trn, unique tax reference, unique taxpayer reference, utr</td>
</tr>
</tbody>
</table>

### Data identifier ARNs for personally identifiable information (PII)

The following table lists the Amazon Resource Names (ARNs) for the data identifiers that you can add to your data protection policies.

<table>
<thead>
<tr>
<th>PII data identifier ARNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/Address</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/CepCode-BR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/Cnpj-BR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/CpfCode-BR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-AT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-AU</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-BE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-BG</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-CA</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-CY</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-CZ</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-DK</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-EE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-FI</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-GR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-HR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-HU</td>
</tr>
<tr>
<td>PII data identifier ARNs</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-IE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-IT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-LT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-LU</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-LV</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-MT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-NL</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-PL</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-PT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-RO</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-SE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-SI</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-SK</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/DriversLicense-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/ElectoralRollNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/EmailAddress</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/IndividualTaxIdentificationNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/LatLong</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/Name</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-IT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NieNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/NifNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-CA</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-IT</td>
</tr>
<tr>
<td>PII data identifier ARNs</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PassportNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PermanentResidenceNumber-CA</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-BR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-IT</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/PostalCode-CA</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/RgNumber-BR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/SocialInsuranceNumber-CA</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/Ssn-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/Ssn-US</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/TaxId-DE</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/TaxId-ES</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/TaxId-FR</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/TaxId-GB</td>
</tr>
<tr>
<td>arn:aws:dataprotection::aws:data-identifier/VehicleIdentificationNumber</td>
</tr>
</tbody>
</table>
Amazon SNS message delivery

This section describes how message delivery works.

Topics
- Amazon SNS raw message delivery (p. 129)
- Sending Amazon SNS messages to an Amazon SQS queue in a different account (p. 130)
- Sending Amazon SNS messages to an Amazon SQS queue or AWS Lambda function in a different Region (p. 134)
- Amazon SNS message delivery status (p. 136)
- Amazon SNS message delivery retries (p. 141)
- Amazon SNS dead-letter queues (DLQs) (p. 146)

Amazon SNS raw message delivery

To avoid having Amazon Kinesis Data Firehose (p. 153), Amazon SQS (p. 174), and HTTP/S (p. 184) 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.
- When you enable raw message delivery for HTTP/S endpoints, the message body, client IP, and the required headers are delivered. When you specify message attributes, it won't be sent.
- When you enable raw message delivery for Kinesis Data Firehose endpoints, the message body is delivered. When you specify message attributes, it won't be sent.

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**

  ```json
  {
    "Type": "Notification",
    "MessageId": "dc1e94df-56cc-5e96-808d-cc7f68faa162",
    "Subject": "TestSubject",
    "Message": "This is a test message."
  }
  ```

- **Raw message delivery is enabled**

  ```json
  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. 174)). The major difference is how you handle subscription confirmation, and that depends on how you subscribe the queue to the topic.

It is a best practice to follow the steps referenced in the Queue owner creates subscription (p. 130) section when possible, because confirmation is automatic when the queue owner creates the subscription.

**Topics**

- Queue owner creates subscription (p. 130)
- A user who does not own the queue creates a subscription (p. 132)
- How do I force a subscription to require authentication on unsubscribe requests? (p. 134)

**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**

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose Topics.
3. Select a topic and then choose Edit.
5. Enter the following policy:

   ```json
   ```

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

   A user with the credentials for account 111122223333 can subscribe to MyTopic. This permission allows the account ID to delegate permission to their IAM user/role. Only the root account or administrator users are allowed to call sns:Subscribe. The IAM user/role must also have sns:subscribe to allow their queue to subscribe.

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. 175).

1. Sign in to the Amazon SQS console.
2. On the navigation panel, choose Queues.
3. From the list of queues, choose the queue to subscribe to the Amazon SNS topic.
4. From Actions, choose Subscribe to Amazon SNS topic.
5. From the Specify an Amazon SNS topic available for this queue menu, choose the Amazon SNS topic for your queue.
6. Choose Enter Amazon SNS topic ARN and then enter the topic's Amazon Resource Name (ARN).
7. Choose Save.
A user who does not own the queue creates a 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 retrieve the subscription confirmation URL, and the subscription owner must confirm the subscription using the subscription confirmation 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. 175).

**Step 1: 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. 175).

1. Sign in to the Amazon SNS console.
2. On the navigation panel, 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**, 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.

The following is an example policy statement that allows the Amazon SNS topic to send a message to the Amazon SQS queue.

```json
{
   "Sid": "Stmt1234",
   "Effect": "Allow",
}
```
A user who does not own the queue creates a subscription

```
"Principal": "*",
"Action": "sqs:SendMessage",
"Condition": {
  " ArnEquals": {
  }
}
```

**Step 2: 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 Details dialog box, find and note the SubscribeURL value. This is your subscription link (example below). For additional details on API token validation, see ConfirmSubscription in the Amazon SNS API Reference.

   ```
   https://sns.us-west-2.amazonaws.com/?
   Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-2:123456789012:MyTopic&Token=2336412f37fb...
   ```
   c. Make a note of the subscription confirmation link. The URL must be passed from the queue owner to the subscription owner. The subscription owner must enter the URL into the Amazon SNS console.
5. Log in as the subscription owner to the Amazon SNS console. The subscription owner performs the confirmation.
6. Choose the relevant topic.
7. Choose the relevant subscription in the topic's subscription listings table. It is labeled as "Pending confirmation".
8. Choose Confirm subscription.
9. A modal appears prompting the subscription confirmation link. Paste the subscription confirmation link.
10. Select the Confirm subscription in the modal.
   An XML response is displayed, for example:

   ```
   <ConfirmSubscriptionResponse>
   <ConfirmSubscriptionResult>
     <SubscriptionArn>arn:aws:sns:us-east-2:123456789012:MyTopic:1234a567-bc89-012d-3e45-6fg7h80123i</SubscriptionArn>
   </ConfirmSubscriptionResult>
   <ResponseMetadata>
     <RequestId>abcd1efg-23hi-jkl4-m5no-p67q8rstuvw9</RequestId>
   </ResponseMetadata>
   </ConfirmSubscriptionResponse>
   ```

   The subscribed queue is ready to receive messages from the topic.
11. (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.

How do I force a subscription to require authentication on unsubscribe requests?

The subscription owner must set the AuthenticateOnUnsubscribe flag to true on subscription-confirmation.

- AuthenticateOnUnsubscribe is automatically set to true when the queue owner creates the subscription.
- AuthenticateOnUnsubscribe cannot be set to true when the subscription confirmation link is navigated to without authentication.

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. 134). 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)
- Asia Pacific (Jakarta)
- 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 Amazon 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),
Asia Pacific (Jakarta), Middle East (Bahrain), Europe (Milan), and Africa (Cape Town). Regions launched prior to March 20, 2019 are enabled by default.

**Cross-region delivery support to Amazon SQS**

<table>
<thead>
<tr>
<th>Cross-region delivery type</th>
<th>Supported/Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default-enabled Region to opt-in Region</td>
<td>Supported using sns.&lt;opt-in-region&gt;.amazonaws.com in the service principal for the queue</td>
</tr>
<tr>
<td>Opt-in Region to default-enabled Region</td>
<td>Supported using sns.&lt;opt-in-region&gt;.amazonaws.com in the service principal for the queue</td>
</tr>
<tr>
<td>Opt-in Region to opt-in Region</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

The following is an example of an access policy statement that allows an Amazon SNS topic in an opt-in Region (af-south-1) to deliver to an Amazon SQS queue in an enabled-by-default Region (us-east-1). It contains the necessary regionalized service principal configuration under the path Statement/Principal/Service.

```json
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.af-south-1.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:af-south-1:111111111111:sourse_topic_name"
        }
      }
    }
  ]
}
```

• To subscribe an AWS Lambda function in US East (N. Virginia) to an Amazon 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), Asia Pacific (Jakarta), Middle East (Bahrain), Europe (Milan), and Africa (Cape Town). Regions launched prior to March 20, 2019 are enabled by default.
Cross-region delivery support to AWS Lambda

<table>
<thead>
<tr>
<th>Cross-region delivery type</th>
<th>Supported/Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default-enabled Region to opt-in Region</td>
<td>Not supported</td>
</tr>
<tr>
<td>Opt-in Region to default-enabled Region</td>
<td>Supported using sns.&lt;opt-in-region&gt;.amazonaws.com in the service principal for the Lambda function</td>
</tr>
<tr>
<td>Opt-in Region to opt-in Region</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

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. 136)
- Configuring message delivery status attributes for topics subscribed to Amazon SNS endpoints using the AWS SDKs (p. 137)

**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](https://docs.aws.amazon.com/cloudwatch/latest/dg/cloudwatch-console.html).

### 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**
Amazon Simple Notification Service Developer Guide

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

- 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**

**Note**
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```
• For API details, see SetTopicAttributes in AWS SDK for Java 2.x API Reference.

JavaScript

**SDK for JavaScript V3**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see SetTopicAttributes in AWS SDK for JavaScript API Reference.

Kotlin

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.
Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun setTopAttr(attribute: String?, topicArnVal: String?, value: String?) {
    val request = SetTopicAttributesRequest {
        attributeName = attribute
        attributeValue = value
        topicArn = topicArnVal
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.setTopicAttributes(request)
        println("Topic \${request.topicArn} was updated.")
    }
}
```

- For API details, see `SetTopicAttributes` in AWS SDK for Kotlin API reference.

**PHP**

**SDK for PHP**

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 * 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());
}

- For API details, see SetTopicAttributes in AWS SDK for PHP API Reference.

### Ruby SDK for Ruby

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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
})
```

- 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. 146).
Topics

- Delivery protocols and policies (p. 142)
- Delivery policy stages (p. 142)
- Creating an HTTP/S delivery policy (p. 143)

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. 143).
- Amazon SNS applies jittering to delivery retries. For more information, see the Exponential Backoff and Jitter post on the AWS Architecture Blog.
- The total policy retry time for an HTTP/S endpoint cannot be greater than 3,600 seconds. This is a hard limit and cannot be increased.

<table>
<thead>
<tr>
<th>Endpoint type</th>
<th>Delivery protocols</th>
<th>Immediate retry (no delay) phase</th>
<th>Pre-backoff phase</th>
<th>Backoff phase</th>
<th>Post-backoff phase</th>
<th>Total attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS managed endpoints</td>
<td>Amazon Kinesis Data Firehose¹</td>
<td>3 times, without delay</td>
<td>2 times, 1 second apart</td>
<td>10 times, with exponential backoff, from 1 second to 20 seconds</td>
<td>100,000 times, 20 seconds apart</td>
<td>100,015 times, over 23 days</td>
</tr>
<tr>
<td></td>
<td>AWS Lambda</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Amazon SQS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer managed endpoints</td>
<td>SMTP</td>
<td>0 times, without delay</td>
<td>2 times, 10 seconds apart</td>
<td>10 times, with exponential backoff, from 10 seconds to 600 seconds (10 minutes)</td>
<td>38 times, 600 seconds (10 minutes) apart</td>
<td>50 attempts, over 6 hours</td>
</tr>
<tr>
<td></td>
<td>SMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mobile push</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ 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.
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 on 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. 146).

**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.

```json
do

```json

```

```

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

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minDelayTarget</code></td>
<td>The minimum delay for a retry.</td>
<td>1 to maximum delay</td>
</tr>
<tr>
<td></td>
<td><strong>Unit:</strong> Seconds</td>
<td><strong>Default:</strong> 20</td>
</tr>
<tr>
<td><code>maxDelayTarget</code></td>
<td>The maximum delay for a retry.</td>
<td>Minimum delay to 3,600</td>
</tr>
<tr>
<td></td>
<td><strong>Unit:</strong> Seconds</td>
<td><strong>Default:</strong> 20</td>
</tr>
<tr>
<td><code>numRetries</code></td>
<td>The total number of retries, including immediate, pre-backoff, backoff, and post-backoff retries.</td>
<td>0 to 100</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 3</td>
<td></td>
</tr>
<tr>
<td><code>numMaxDelayRetries</code></td>
<td>The number of retries in the post-backoff phase, with the maximum delay between them.</td>
<td>0 or greater</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 0</td>
<td></td>
</tr>
<tr>
<td><code>numNoDelayRetries</code></td>
<td>The number of retries to be done immediately, with no delay between them.</td>
<td>0 or greater</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> 0</td>
<td></td>
</tr>
</tbody>
</table>
Amazon SNS uses the following formula to calculate the number of retries in the backoff phase:

\[
\text{numRetries} - \text{numNoDelayRetries} - \text{numMinDelayRetries} - \text{numMaxDelayRetries}
\]

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

- \text{minDelayTarget} – Defines the delay associated with the first retry attempt in the backoff phase.
- \text{maxDelayTarget} – Defines the delay associated with the final retry attempt in the backoff phase.
- \text{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. 148) and Amazon SNS message delivery retries (p. 141).

**Note**

- The Amazon SNS subscription and Amazon SQS queue must be under the same AWS account and Region.
- For a FIFO topic (p. 43), 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 KMS 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. 438) 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. 146)
- How do dead-letter queues work? (p. 147)
- How are messages moved into a dead-letter queue? (p. 147)
- How can I move messages out of a dead-letter queue? (p. 148)
- How can I monitor and log dead-letter queues? (p. 148)
- Configuring an Amazon SNS dead-letter queue for a subscription (p. 148)

**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. 141).

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. 148).

Note
For a FIFO topic (p. 43), 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. 486).

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. 136).

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. 146) and Amazon SNS message delivery retries (p. 141).

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. 174).
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",
         "Condition": {
            "ArnEquals": {
            }
         }
      }
   ]
}
```

**Topics**

- To configure a dead-letter queue for an Amazon SNS subscription using the AWS Management Console (p. 149)
- To configure a dead-letter queue for an Amazon SNS subscription using an AWS SDK (p. 150)
- To configure a dead-letter queue for an Amazon SNS subscription using the AWS CLI (p. 150)
- To configure a dead-letter queue for an Amazon SNS subscription using AWS CloudFormation (p. 151)

**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. 148).

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. 43), 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 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. 148).

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**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
// 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.

// 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);
```

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

Before your begin this tutorial, make sure you complete the prerequisites (p. 148).

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

```bash
aws sns set-subscription-attributes
   --subscription-arn arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-bc89-012d-3e45-6fg7h890123i
   --attribute-name RedrivePolicy
```
To configure a dead-letter queue for an Amazon SNS subscription using AWS CloudFormation

Before your begin this tutorial, make sure you complete the prerequisites (p. 148).

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

```json
{
  "Resources": {
    "mySubscription": {
      "Type": "AWS::SNS::Subscription",
      "Properties": {
        "Protocol": "sqs",
        "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.


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. 153)
- Example use case for message archiving and analytics (p. 164)
- Working with delivery stream destinations (p. 155)
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. 153)
- Fanout to Lambda functions (p. 173)
- Fanout to Amazon SQS queues (p. 174)
- Fanout to HTTP/S endpoints (p. 184)
- Fanout to AWS Event Fork Pipelines (p. 200)

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. 136).

Topics

- Prerequisites for subscribing Kinesis Data Firehose delivery streams to Amazon SNS topics (p. 153)
- Subscribing a Kinesis Data Firehose delivery stream to an Amazon SNS topic (p. 155)
- Working with delivery stream destinations (p. 155)
- Example use case for message archiving and analytics (p. 164)

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:
Prerequisites

- 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:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "firehose:DescribeDeliveryStream",
            "firehose:ListDeliveryStreams",
            "firehose:ListTagsForDeliveryStream",
            "firehose:PutRecord",
            "firehose:PutRecordBatch"
         ],
         "Resource": [
         ],
         "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:

```json
{
   "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. 155).
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. 153), first make sure that you've addressed all the prerequisites (p. 153).

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. 153).
   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. 129).
5. (Optional) To configure a filter policy, expand the Subscription filter policy section. For more information, see Amazon SNS subscription filter policies (p. 79).
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. 146).
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. 153), you can send messages to additional endpoints. This section describes how to work with supported destinations.

Topics

- Amazon S3 destinations (p. 155)
- OpenSearch Service destinations (p. 158)
- Amazon Redshift destinations (p. 160)
- HTTP destinations (p. 163)

Amazon S3 destinations

This section provides information about Amazon Kinesis Data Firehose delivery streams that publish data to Amazon Simple Storage Service (Amazon S3).
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. 129).
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-1:333333333333:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5","MessageAttributes":{"myKey1":{"Type":"String","Value":"myValue1"}},"myKey2":{"Type":"String","Value":"myValue2"}}
```

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. 158)
- Analyzing messages for OpenSearch Service destinations (p. 159)

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. 129).

```json
{
    "_index": "my-index",
    "_type": "_doc",
    "_id": "496131009631113232032504054021932879477586550985932802.0",
    "_version": 1,
    "_score": null,
    "_source": {
        "Type": "Notification",
        "MessageId": "bf32e294-46e3-5dd5-a6b3-bad65162e136",
        "Subject": "Sample subject",
        "Message": "Sample message",
        "Timestamp": "2020-12-02T22:29:21.189Z",
        "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:

```json
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.
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. 129).
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.

<table>
<thead>
<tr>
<th>type</th>
<th>messageid</th>
<th>topicarn</th>
<th>subject</th>
<th>message</th>
<th>timestamp</th>
<th>unsubscribe</th>
<th>messageattributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notification</td>
<td>ea544832-a0d8-581d-a0d8</td>
<td>arn:aws:sns:us-east-1:111111111111/my-topic</td>
<td>Sample subject 1</td>
<td>Sample message 1</td>
<td>2020-12-02T00:33:32.272Z</td>
<td><a href="https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b">https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b</a></td>
<td>&quot;my_attribute&quot;: {&quot;Type&quot;: &quot;String&quot;, &quot;Value&quot;: &quot;my_value&quot;}</td>
</tr>
<tr>
<td>Notification</td>
<td>ce644832-a0d8-581d-a0d8</td>
<td>arn:aws:sns:us-east-1:111111111111/my-topic</td>
<td>Sample subject 3</td>
<td>Sample message 3</td>
<td>2020-12-09T00:08:44.405Z</td>
<td><a href="https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b">https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b</a></td>
<td>&quot;my_attribute3&quot;: {&quot;Type&quot;: &quot;String&quot;, &quot;Value&quot;: &quot;my_value&quot;}</td>
</tr>
</tbody>
</table>

For more information about fanning out notifications to Amazon Redshift endpoints, see Amazon Redshift destinations (p. 160).

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 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. 164)
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. 129).

```
"body": {
  "requestId": "ebc9e8b2-fce3-4aef-a8f1-71698bf8175f",
  "timestamp": 1606255960435,
  "records": [
    { "data": "eyJUeXBlIjoiTm90aWZpY2F0aW9uIiwiTWVzc2FnZUlkIjoiMjFkMmUzOGQtMmNhYi01ZjYxLTliYTItYmJiYWFhYzg0MGY2IiwiVG9waWNBcm4iOiJhcm46Y ... iZSZTdWJzY3JpcHRpb25Bcm49YXJuOmF3czpzbnM6MTExMTExMTExMTExOm15LXRvcGljOjAxYjY5MTJjLTAwNzAtNGQ4Yi04YjEzLTU1NWJmYjc2ZTdkNCJ9"
  ]
}
```

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.
This page describes how to create the following resources for the message archiving and analytics example use case:

- 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:

```json
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "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 \(\text{us-east-1}\) 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. 32). 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. 167).

**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. 164).

**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.
5. On the Choose a destination page, do the following:
Example use case

a. For Destination, choose Amazon S3.
b. Under S3 destination, for S3 bucket, choose the S3 bucket that you created initially (p. 166).
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. 168).

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. 164):

• 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. 169).

**Testing and querying the configuration**

This page describes how to test the message archiving and analytics example use case (p. 164) 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.

```json
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15 04:15:05","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. 63).

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. 166).

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:

```sql
SELECT destination
FROM default.notifications
GROUP BY destination
ORDER BY count(*) desc
LIMIT 1;
```

To find the top destination, run the following query:

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. 164), you can use the following YAML template:

```yaml
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:
```
Example use case

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
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. 136).

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-fc5cf758c76",
"Subject" : "Testing publish to subscribed queues",
"Message" : "Hello world!",
"Timestamp" : "2012-03-29T05:12:16.901Z",
"SignatureVersion" : "1",
"Signature" : "EXAMPLEnTrFPa3...",
"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. 175)
  2. Give sqs:SendMessage permission to the Amazon SNS topic so that it can send messages to the queue. (p. 175)
  3. Subscribe the queue to the Amazon SNS topic. (p. 176)
  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. 177)
  5. Test it out by publishing a message to the topic and reading the message from the queue. (p. 178)
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. 130).

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. 179).

**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. If you are subscribing a queue to multiple topics, your policy must contain one `Statement` element for each topic.

**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.
   - Use the **aws:SourceArn** or **aws:SourceAccount** global condition keys to protect against the **confused deputy** scenario. To use these condition keys, set the value to the ARN of your topic. If your queue is subscribed to multiple topics, you can use **aws:SourceAccount** instead.

   For example, the following policy allows MyTopic to send messages to MyQueue.

   ```json
   { "Statement": [ {
         "Effect": "Allow",
         "Principal": {
           "Service": "sns.amazonaws.com"
         },
         "Action": "sqs:SendMessage",
         "Condition": {
           "ArnEquals": {
           }
         }
       }
   ]}
   ```

**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
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. 475), 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. 177).** 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. 178).** 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.

```json
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sns:Publish",
    }
  ]
}
```

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.

```json
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage"
      ],
      "Resource": [
        "arn:aws:sqs:us-east-2:123456789012:MyQueue2"
      ]
    }
  ]
}
```
Subscribing a queue to a topic

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.

```json
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": "sns:Publish",
        }
    ]
}
```

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.

```json
{
    "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.

```json
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEnTrFPa3...",
}
```
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.

MyRDMessageQueueGroup 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 MyRDMessageQueueGroup 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.

```json
{
  "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."
    }
  }
}
```


"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": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage"
              ],
              "Resource": [
                {
                  "Fn::GetAtt": ["MyQueue1", "Arn"]
                },
                {
                  "Fn::GetAtt": ["MyQueue2", "Arn"]
                }
              ]
            }]
          }
        }
      }
    ],
    "Users": {
      "Ref": "MyQueueUser"
    }
  }
},
"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": ["sqs:SendMessage"],
          "Resource": "*",
          "Condition": {
            "ArnEquals": {
              "aws:SourceArn": {
                "Ref": "MySNSTopic"
              }
            }
          }
        }
      ]
    },
    "Queues": [{
      "Ref": "MyQueue1"
    }, {
      "Ref": "MyQueue2"
    }]
  }
},
"Outputs": {
  "MySNSTopicARN": {
    "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"
          }
        ]
      ]
    }
  }
}
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](#).

**Important**
HTTPS URLs are only retrievable from the Amazon SNS GetSubscriptionAttributes API action, for principals to which you have granted API access.

**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](#) and [HTTP/HTTPS notification JSON format](#).

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: da41e39f-ea4d-435a-b922-c6aae3915ebe
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",
  "Subject" : "test",
  "Message" : "test message",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLElDMXvB8r9R83tGoNn0ecwd5UjlllzzsVsbItzfaMpn2nk5HVSw7Xn0n/491kxDKz8Yrz1H2qjXj2izB02o2071c4qK1fMUDfJf3uef2h2Qvycj2233fe7bb5f59f96de52f.pem",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfbd224c720ce7bb5f59f96de52f.pem",
}
```

**Topics**

- Subscribing an HTTP/S endpoint to a topic (p. 185)
- Verifying the signatures of Amazon SNS messages (p. 191)
- Parsing message formats (p. 193)

**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

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. 193). The following HTTP POST request is an example of a subscription confirmation message.

```plaintext
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
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...",
  "Message": "You have chosen to subscribe to the topic arn:aws:sns:us-west-2:123456789012:MyTopic.
To confirm the subscription, visit the SubscribeURL included in this message."
  "Timestamp": "2012-04-26T20:45:04.751Z",
  "SignatureVersion": "1",
  "Signature": "EXAMPLEpH+...",
```
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. 191).

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. 194). 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.

```xml
<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. 193). 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
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",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEw6JRN...",
}
```

5. Make sure that your endpoint responds to the HTTP POST message from Amazon SNS with the appropriate status code. The connection will timeout 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.
3. Choose the Create subscription.
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. The confirmation message is displayed. 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. 186) 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. 186), 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. 475).
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.

```json
{
    "Statement": [
        {
            "Sid": "AllowPublishToMyTopic",
            "Effect": "Allow",
            "Action": "sns:Publish",
        }
    ]
}
```

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.

```json
{
    "Statement": [
        {
            "Sid": "Allow-publish-to-topic",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": "sns:Publish",
        }
    ]
}
```

**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. 186), the code that you deployed at your endpoint should process the notification.
Verifying the signatures of Amazon SNS messages

To verify the authenticity of a message sent to your HTTP endpoint by Amazon SNS, you can verify the message signature. There are two cases where we recommend verifying the authenticity of the message. First, when Amazon SNS sends a message to your HTTP endpoint that you subscribed to a topic. Second, when Amazon SNS sends you a confirmation message to your HTTP endpoint upon the execution of the Subscribe or the Unsubscribe API actions.

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.

Amazon SNS supports two message signature versions:

- SignatureVersion1: Amazon SNS creates the signature based on the SHA1 hash of the message.
- SignatureVersion2: Amazon SNS creates the signature based on the SHA256 hash of the message.

To configure the message signature version on Amazon SNS topics

By default, Amazon SNS topics use SignatureVersion 1. To choose the hashing algorithm on your Amazon SNS topic, either SignatureVersion 1 (SHA1) or SignatureVersion 2 (SHA256), you can use the SetTopicAttributes API action.

The following code example shows how to set the topic attribute SignatureVersion using the AWS CLI:

```bash
aws sns set-topic-attributes
    --attribute-name SignatureVersion
    --attribute-value 2
```

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. 186).

The SignatureVersion tells you the signature version used by Amazon SNS to generate the signature of the message. From the signature version, you can determine the requirements for how to generate the signature. For notifications, Amazon SNS currently supports signature version 1 and 2. This section provides the steps for verifying a signature using these signature versions.

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-1G1WEFCOXTC0P
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.

```plaintext
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-1G1WEFCOXTC0P&Token=233...
Timestamp
Token
233...
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXTC0P
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 algorithm used to generate the signature.
   a. If the SignatureVersion is 1, use SHA1 as the hash algorithm.
   b. If the SignatureVersion is 2, use SHA256 as the hash algorithm.

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. 193)
- HTTP/HTTPS subscription confirmation JSON format (p. 194)
- HTTP/HTTPS notification JSON format (p. 196)
- HTTP/HTTPS unsubscribe confirmation JSON format (p. 197)
- SetSubscriptionAttributes delivery policy JSON format (p. 198)
- SetTopicAttributes delivery policy JSON format (p. 199)

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
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. 
To 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 or SHA256withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

**SignatureVersion**

Version of the Amazon SNS signature used.

- If the SignatureVersion is 1, Signature is a Base64-encoded SHA1withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.
- If the SignatureVersion is 2, Signature is a Base64-encoded SHA256withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

**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
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...",
   "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.",
   "Timestamp" : "2012-04-26T20:45:04.751Z",
   "SignatureVersion" : "1",
   "Signature" : "signature-value"
}
```
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 or SHA256withRSA signature of the Message, MessageId, Subject (if present), Type, Timestamp, and TopicArn values.

**SignatureVersion**

Version of the Amazon SNS signature used.

- If the SignatureVersion is 1, Signature is a Base64-encoded SHA1withRSA signature of the Message, MessageId, Subject (if present), Type, Timestamp, and TopicArn values.
- If the SignatureVersion is 2, Signature is a Base64-encoded SHA256withRSA signature of the Message, MessageId, Subject (if present), Type, Timestamp, and TopicArn values.

**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.

```
"Signature" : "EXAMPLEpH+DcEwjAPg809mY8dReBSksfg2S7WKQcikcNKWLQjwu6A4VbeS0QHVCKhR57fUQvi2egU3N858fiTDN6bkkoXyYDvry0Ad8L10Hs3zJH6Dr+kLkG44ucG+iU6FAl+08Oy6Ku37zw6T3q7aZsN+jV5I7I708I+hP9ACL+gqQ8/K90k9c56cfF820HDeY85JylmDdm+GdoU8h5zTlovJn4KCI1CXGu43obcgFxeL3khZl8IKvO61GWB6jI9b5+gLPoBc1Q=",
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
```
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. To 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 or SHA256withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

**SignatureVersion**

Version of the Amazon SNS signature used.

- If the SignatureVersion is 1, Signature is a Base64-encoded SHA1withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.
- If the SignatureVersion is 2, Signature is a Base64-encoded SHA256withRSA signature of the Message, MessageId, Type, Timestamp, and TopicArn values.
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
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...",
  "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55. To cancel this operation and restore the subscription, visit the SubscribeURL included in this message."
}
```

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.

```
{
  "Type" : "UnsubscribeConfirmation",
  "MessageId" : "47138184-6831-46b8-8f7c-afc488602d7d",
  "Token" : "2336412f37...",
  "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55. To cancel this operation and restore the subscription, visit the SubscribeURL included in this message."
}
```
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](http://sns.us-east-2.amazonaws.com/?Action=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](http://sns.us-east-2.amazonaws.com/?Action=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. 153).

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. 204).

Topics
- How AWS Event Fork Pipelines works (p. 200)
- Deploying AWS Event Fork Pipelines (p. 203)
- Deploying and testing the AWS Event Fork Pipelines sample application (p. 204)
- Subscribing an AWS Event Fork Pipelines to an Amazon SNS topic (p. 210)

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. 201)
- The event search and analytics pipeline (p. 202)
- The event replay pipeline (p. 202)

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. 79). 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. 210).

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. 200).

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. 204)
- Step 1: To deploy the sample application (p. 206)
- Step 2: To execute the sample application (p. 207)
- Step 3: To verify the execution of the sample application and its pipelines (p. 208)
- Step 4: To simulate an issue and replay events for recovery (p. 209)

### 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. 201) 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. 202) 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. 202) 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. 79).

```
{
   "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://abcdefghij.execute-api.us-east-2.amazonaws.com/Prod/checkout
   ```
4. Copy the following JSON to a file named **test_event.json**.

   ```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 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:

```bash
curl -d "$(cat test_event.json)" https://abcdefghij.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 serverl-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 serverl-analyt-abcdefgh1ijk 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.
4. In the View/Delete Messages in fork-example-ecommerce-my-app...ReplayP-ReplayQueue-123ABC4D5EF6 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.

```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:

```bash
curl -d "$\{(cat test_event_2.json)\}" https://abcdefghij.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](https://console.aws.amazon.com/dynamodb/")
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](https://console.aws.amazon.com/lambda/")
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)](https://aws.amazon.com/sam/), which you can deploy directly from the [AWS Event Fork Pipelines suite](https://github.com/awslabs/aws-event-fork-pipelines) (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. 200).

This section show 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 on 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. 211)
- To deploy and subscribe the event search and analytics pipeline (p. 213)
- To deploy and subscribe the event replay pipeline (p. 215)

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. 153).

This page shows how to deploy the Event Storage and Backup Pipeline (p. 201) 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. 79) 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. 153).

This page shows how to deploy the Event Search and Analytics Pipeline (p. 202) 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. 79) 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
Subscribing an event pipeline to a topic

- 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)

   (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. 202) 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. 79) 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. 217)
- Mobile push notifications (p. 312)
- Email notifications (p. 339)

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. 260), or you can send a message to multiple phone numbers (p. 256) at once by subscribing those phone numbers to a topic and sending your message to the topic.

You can set SMS preferences (p. 252) 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. 273), 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. 288).

Topics
- SMS sandbox (p. 218)
- Origination identities for SMS messages (p. 220)
- Requesting support for SMS messaging with Amazon SNS (p. 244)
- Setting SMS messaging preferences (p. 252)
- Sending SMS messages (p. 256)
- Monitoring SMS activity (p. 267)
- Managing phone numbers and SMS subscriptions (p. 273)
- Supported Regions and countries (p. 288)
- SMS best practices (p. 299)
- SMS requirements for Singapore (p. 307)
- Special requirements for sending SMS messages to US destinations (p. 308)
- Special requirements for sending SMS messages to recipients in India (p. 309)
**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. 218)
- Deleting phone numbers from the SMS sandbox (p. 219)
- Moving out of the SMS sandbox (p. 219)

**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. 218), first add destination phone numbers, and then verify them.

**Note**

As with accounts that aren't in the SMS sandbox, an origination identity (p. 220) is required before you can send SMS messages to recipients in some countries or regions. For more information, see Supported Regions and countries (p. 288). Origination IDs include sender IDs (p. 220) and different types of origination numbers (p. 221). 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. 288).
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. 218).

To delete destination phone numbers from the SMS sandbox

1. Wait 24 hours after verifying the phone number (p. 218), 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. 288) 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. 218) 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. 218) 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. 260).

3. On the Amazon SNS console's Mobile text messaging (SMS) page, under Account information, choose Exit SMS sandbox.

   The Amazon Support Center opens and creates a case, with the Service limit increase option already selected.

4. Under Case details, do the following:
   a. For Limit type, choose SNS Text Messaging.
   b. (Optional) Provide a link to the site or the name of the app that you plan to send SMS messages from.
   c. (Optional) Choose the type of messages that you plan to send: One Time Password, Promotional, or Transactional.
   d. (Optional) Choose the AWS Region that you plan to send SMS messages from.
Origination identities

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. 220)**
- **Origination numbers (p. 221)**

**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, use the AWS Service Quotas console to create a service limit increase case while requesting the 10DLC number for that Region. To learn more about how to request origination identities, see Requesting support for SMS messaging with Amazon SNS (p. 244).

**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.

Sender IDs are supported in several countries and regions around the world. In some places, if you're a business that sends SMS messages to individual customers, you must use a sender ID that's pre-
registered with a regulatory agency or industry group. For a complete list of countries and regions that support or require sender IDs, see Supported Regions and countries (p. 288).

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.

Important
AWS prohibits 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.

Advantages
Sender IDs provide the recipient with more information about the message sender. It's easier to establish your brand identity by using a sender ID than by using a short or long code. There's no additional charge for using a sender ID.

Disadvantages
Support and requirements for sender ID authentication aren't consistent across all countries or regions. Several major markets (including Canada, China, and the United States) don't support sender ID. In some areas, you must have your sender IDs pre-approved by a regulatory agency before you can use them.

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. 220) instead of origination numbers.

Topics
- 10DLC (p. 221)
- Toll-free numbers (p. 234)
- Toll-free number registration requirements and process (p. 237)
- Short codes (p. 242)
- Person-to-person (P2P) long codes (p. 243)
- U.S. product number comparison (p. 243)

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. 232).

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.

10DLC capabilities

The capabilities of 10DLC phone numbers depend on which mobile carriers your recipients use. AT&T provides a limit on the number of message parts that can be sent each minute for each campaign. T-Mobile provides a daily limit of messages that can be sent for each company, with no limit on the number of message parts that can be sent per minute. Verizon hasn't published throughput limits, but uses a filtering system for 10DLC that is designed to remove spam, unsolicited messages, and abusive content, with less emphasis on the actual message throughput.

New 10DLC campaigns that are associated with unvetted companies can send 75 message parts per minute to recipients who use AT&T, and 2,000 messages per day to recipients who use T-Mobile. The company limit is shared across all of your 10DLC campaigns. For example, if you have registered one company and two campaigns, the daily allotment of 2,000 messages to T-Mobile customers is shared across those campaigns. Similarly, if you register the same company in more than one AWS account, the daily allotment is shared across those accounts.

If your throughput needs exceed these limits, you can request that your company registration be vetted. When you vet your company registration, a third-party verification provider analyzes your company details. The verification provider then provides a vetting score, which determines the capabilities of your 10DLC campaigns. There is a one-time charge for the vetting service. For more information, see Vetting your Amazon SNS 10DLC registration (p. 226).

Your actual throughput rate will vary depending on various factors, such as whether or not your company has been vetted, your campaign types, and your vetting score. The following flowchart shows the throughput rates for various situations.
Throughput rates for 10DLC are determined by the US mobile carriers in cooperation with the Campaign Registry. Neither Amazon SNS nor any other SMS sending service can increase 10DLC throughput beyond these rates. If you need high throughput rates and high deliverability rates across all US carriers, we recommend that you use a short code. For more information about obtaining a short code, see Requesting dedicated short codes for SMS messaging with Amazon SNS (p. 245).

Getting started with 10DLC

Use the Amazon Pinpoint console (not Amazon SNS) 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. 224). 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.
Amazon Simple Notification Service Developer Guide
Origination identities

Note
Amazon SNS SMS messaging is available in Regions where Amazon Pinpoint is not currently supported. Follow the instructions for your account type:

a. If you are using a commercial cloud account, you need to open the Amazon Pinpoint console in the US East (N. Virginia) Region to register your 10DLC company and campaign. Don't request a 10DLC number.

b. If you are using an AWS GovCloud (US) account, open the Amazon Pinpoint console in the US West Region to register your 10DLC company and campaign. Don't request a 10DLC number.

c. Use the AWS Service Quotas console to create a service limit increase case while requesting the 10DLC number for that Region. For information on Regions where Amazon Pinpoint is available, see Amazon Pinpoint endpoints and quotas in the AWS General Reference.

2. (Optional, but recommended) Apply for vetting

If your company registration is successful, you can begin creating low-volume, mixed-use 10DLC campaigns. These campaigns can send 75 messages per minute to recipients who use AT&T, and your registered company can send 2,000 messages per day to recipients who use T-Mobile. If your use case requires a throughput rate that exceeds these values, you can apply for vetting of your company registration. Vetting your company registration can increase the throughput rates for your companies and campaigns, but it isn't guaranteed to do so. For more information about vetting, see Vetting your Amazon SNS 10DLC registration (p. 226).

3. 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. 228).

4. 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. 247).

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, use the AWS Service Quotas console to create a service limit increase case while requesting the 10DLC number for that Region. For information on Regions where Amazon Pinpoint is available, see Amazon Pinpoint endpoints and quotas in the AWS General Reference.

10DLC company registration statuses

When you register your company or brand, one of two statuses is returned: either Unverified or Verified. If the status for your company registration is Unverified, it means that there was an issue with your
registration. For example, the Registered Company Name that you provided might not exactly match the registered name of the company associated with the Tax ID that you provided. If you find a problem with your company registration details, you can correct them. For more information about modifying your company registration details, see Editing or deleting a registered company (p. 227).

If the status for your company registration is Verified, then the registration details you provided were accurate, and you can begin creating 10DLC campaigns.

Registering your company or brand

You need to register your company only once. After it’s registered, you can edit your company and contact information. To delete a registered company, create a case with AWS Support. For more information about editing or deleting company details, see Editing or deleting a registered company (p. 227).

To register a company

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. In the navigation pane, under SMS and voice, choose Phone numbers.
3. On the 10DLC campaigns tab, choose Register company.
   
   **Note**
   The Register your company page displays the Registration fee. This is a one-time fee associated with registering your company. This cost is separate from any other monthly costs or fees. It is charged to you when you register your company, or when you modify the details of an existing company registration.

4. In the Company info section, do the following:
   
   • For Legal company name, enter the name that the company is registered under. The name that you enter must be an exact match for the company name that’s associated with the tax ID that you provide.
   
   **Important**
   Make sure to use your company’s exact legal name. Once submitted you can’t change this information. Incorrect or incomplete information might result in your registration being delayed or denied.

   • For What type of legal form is this organization, choose the option that best describes your company.

   **Note**
   The US government and Not-for-profit options can only be used to register United States-based organizations. If your organization is based in a country other than the US, you must register as Private for-profit, regardless of the actual legal form of your organization.

   • If you chose Public for profit in the previous step, enter the company’s stock symbol and the stock exchange that it’s listed on.
   • For Country of registration, choose the country where the company is registered.
   • For Doing Business As (DBA) or brand name, enter any other names that your company does business as.
   • For Tax ID, enter your company’s tax ID. The ID that you enter depends on the country that your company is registered in.
   • If you’re registering a US or non-US entity that has an IRS Employer Identification Number (EIN), enter your nine-digit EIN. The legal company name, EIN, and physical address that you enter must all match the company information that is registered with the IRS.
• If you're registering a Canadian entity, enter your federal or provincial Corporation number. Don't enter the Business Number (BN) provided by the CRA. The legal company name, Corporation number, and physical address that you enter must all match the company information that is registered with Corporations Canada.

• If you're registering an entity that is based in another country, enter the primary tax ID for your country. In many countries, this is the numeric portion of your VAT ID number.

• For **Vertical**, choose the category that best describes the company you're registering.

5. In the **Contact info** section, do the following:

   • For **Address/Street**, enter the physical street address associated with your company.
   • For **City**, enter the city where the physical address is located.
   • For **State or region**, enter the state or region where the address is located.
   • For **Zip Code/Postal Code**, enter the ZIP or postal code for the address.
   • For **Company website**, enter the full URL of your company's website. Include "http://" or "https://" at the beginning of the address.
   • For **Support email**, enter an email address.
   • For **Support phone number**, enter a phone number with a country code.

   **Note**
   The Campaign Registry requires a contact email address and phone number in case they need to verify the registration information with a representative of your company.

6. When you finish, choose **Create**. Your company registration is submitted to the Campaign Registry.

   In most cases, your registration is accepted immediately, and a status is provided.

If the status for your company registration is **Verified**, then you can begin creating low-volume, mixed-use 10DLC campaigns. You can use this type of campaign to send up to 75 messages per minute to recipients who use AT&T, and your registered company can send 2,000 messages per day to recipients who use T-Mobile. You can also send messages to recipients who use other US carriers, such as Verizon and US Cellular. These carriers don't strictly enforce throughput limits, but they do heavily monitor 10DLC messages for signs of spam and abuse.

If your use case requires a throughput rate that exceeds these values, you can apply for additional vetting of your company registration. For more information about vetting your brand registration, see [Vetting your Amazon SNS 10DLC registration](p. 226).

If the status for your company registration is **Unverified**, there were issues with the information that you provided. Check the information that you provided and confirm that all of the fields contain the correct information. You can make changes to some parts of your company registration in the Amazon Pinpoint console. For more information about modifying your company registration details, see [Editing a 10DLC company registration](p. 227).

**Vetting your Amazon SNS 10DLC registration**

If your company registration is successful and you want to register a campaign with higher throughput capabilities, then you must vet your company registration.

When you vet your registration, a third-party organization analyzes the company details that you provided and returns a vetting score. A high vetting score can lead to higher throughput rates for your 10DLC company and the campaigns associated with it. However, vetting isn't guaranteed to increase your throughput.

Vetting scores aren't applied retroactively. In other words, if you've already created a 10DLC campaign, and you later vet your company registration, your vetting score isn't automatically applied to your existing campaign. For this reason, you should vet your company or brand before you create any of your 10DLC campaigns.
**Note**
There is a $40 non-refundable fee for vetting your company or brand.

**To vet your company registration**

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. In the navigation pane, under **SMS and voice**, choose **Phone numbers**.
3. On the **10DLC campaigns** tab, choose the **10DLC company** that you want to vet.
4. On the company details page, toward the bottom of the page, choose **Apply for vetting**.
5. On the **Apply for additional vetting** window, choose **Submit**.

For US-based companies, the vetting process typically takes around a minute to complete. For non-US-based companies, the vetting process might take significantly more time, depending on how readily available data is for that country.

After you submit a vetting request, you return to the company details page. The **Company vetting results** section displays the status and results of your vetting request. When the vetting process completes, this table shows a vetting score in the **Score** column. Your vetting score determines your 10DLC throughput capabilities. Your throughput varies based on the type of campaign that you create. If you create mixed-use or marketing-related 10DLC campaigns, you need to have a higher vetting score than you'd need for other campaign types in order to achieve high throughput rates. For more information about the capabilities of 10DLC phone numbers, see **10DLC capabilities** (p. 222).

If you change the details of your company registration after completing the vetting process, you can request to have your registration vetted again. If you only change the Vertical for your company registration, your vetting score won't change. If you change any details other than the Vertical, your vetting result could change. In either case, you're charged the one-time vetting fee again.

**Editing or deleting a registered company**

You can edit some of the 10DLC registration information for your company directly in the Amazon Pinpoint console. You can also delete a 10DLC company registration by creating a case in the AWS Support Center.

**Editing a 10DLC company registration**

After you complete the 10DLC registration process for a company, you can edit the details of your registration.

If you see an error message after editing your company registration details, there might be other issues with your registration. You can open a ticket with AWS Support to request more information.

**To edit a company**

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. In the navigation pane, under **SMS and voice**, choose **Phone numbers**.
3. On the **10DLC campaigns** tab, in the list of campaigns, choose the name of the company that you want to edit the details for.
4. Choose **Edit**.
5. Modify the registration information as needed. When you finish, choose **Re-submit**.

**Note**
When you re-submit your registration details, you're charged a one-time registration fee. Currently, this registration fee is $4 (USD).
Deleting a 10DLC company registration

If you want to delete a registered company, you have to open a ticket with AWS Support. You can’t delete a company through the Amazon Pinpoint console.

To 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. In the navigation pane, under SMS and voice, choose Phone numbers.
3. On the 10DLC campaigns tab, choose Request to delete company.
4. Confirm that you want to continue by choosing Request to delete company again to open the AWS Support Center.
5. For the case type, choose Service limit increase.
6. For the Limit type, choose Pinpoint.
7. In the Requests section, choose the Region, and then for the Limit, choose 10DLC - request to update/delete 10DLC company or campaign.
8. Leave the New limit value field empty.
9. For Use case description, enter the company name you want to delete, 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

When you register a 10DLC campaign, you provide a description of your use case, as well as the message templates that you plan to use. Before you can create and register a 10DLC campaign, you must first register your company. For information on registering your company, see Registering a company (p. 224).

Note
After you register your company, Amazon Pinpoint shows one of two statuses for the registration: Verified or Unverified. You can only complete the 10DLC campaign registration process if the status of your company registration is Verified. You will be able to create low-volume mixed use campaigns.
If the status is Unverified, it usually means that some of the data that you provided when you registered your company was incorrect. You won’t be able to create any 10DLC campaigns while your company has this status. You can modify your company registration to attempt to fix the issues with your company registration. For more information about modifying 10DLC company registrations, see Editing or deleting a registered company (p. 227).

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.

In this section, you’ll choose the company you’re creating the 10DLC campaign for and provide additional details.

To register 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 SMS and voice, choose Phone numbers.
3. On the **10DLC campaigns** tab, choose **Create a 10DLC campaign**.

4. On the **Create a 10DLC campaign** page, in the **Campaign info** section, do the following:
   
   a. 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. For more information about registering a company, see **Registering a company** (p. 224).
   
   b. For **10DLC campaign name**, enter a name for the campaign.
   
   c. For **Vertical**, choose option that best represents your company.
   
   d. For **Help message**, enter the message that your customers receive if they send the keyword “HELP” to your 10DLC phone number.
   
   e. For **Stop message**, enter the message that your customers receive if they send the keyword “STOP” to your 10DLC phone number.

   **Tip**
   
   Your customers can reply to your messages with the word “HELP” to learn more about the messages that they're receiving from you. They can also reply “STOP” to opt-out of receiving messages from you. The US mobile carriers require you to provide responses to both of these keywords.
   
   The following is an example of a HELP response that complies with the requirements of the US mobile carriers:
   
   **ExampleCorp Account Alerts: For help call 1-888-555-0142 or go to example.com. Msg&data rates may apply. Text STOP to cancel.**
   
   The following is an example of a compliant STOP response:
   
   **You are unsubscribed from ExampleCorp Account Alerts. No more messages will be sent. Reply HELP for help or call 1-888-555-0142.**
   
   Your responses to these keywords must contain 160 characters or fewer.

5. In the **Campaign use case** section, do the following:

   a. For **Use case type**, if you have a charity-related use case, choose **Special**. Otherwise, choose **Standard**.
   
   b. 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.

   **Note**
   
   The monthly charge for registering the 10DLC campaign is shown next to each use case type. Most 10DLC campaign types have the same monthly charge. The charge for registering Low-Volume Mixed use cases is lower than for other use case types. However, Low-Volume Mixed campaigns support lower throughput rates than other campaign types.

   c. Enter at least one **Sample SMS message**. This is the sample message you plan to send to your customers. If you plan to use multiple message templates for this 10DLC campaign, include them as well.

   **Important**
   
   Don't use placeholder text for your sample messages. The example messages that you provide should reflect the actual messages that you plan to send as accurately as possible.

6. The **Campaign and content attributes** section contains a series of **Yes** or **No** questions related to the particular features of the campaign. Some attributes are mandatory, so you can't change the default value.

   Make sure that the attributes you choose are accurate for your campaign.

   Indicate whether each of the following applies to the campaign that you're registering:

   - **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. Links from common URL shorteners, such as TinyUrl or Bit.ly, are not allowed. However, you can use URL shorteners that offer custom domains.

• **Embedded phone number** – The campaign includes an embedded phone number that isn't 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.

7. Choose Create.

After you submit the registration details for your campaign, 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 phone number and associate that number with your campaign. For more information, see *Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS* (p. 247).

• **Pending** – Your 10DLC campaign hasn’t been approved yet. In some cases, approval might take one week or more. If the status changes, the Amazon Pinpoint console reflects that change. We don’t notify you of status changes.

• **Rejected** – Your 10DLC campaign was rejected. To get more information, submit a support request that includes the campaign ID of the rejected campaign.

• **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.

8. 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. 247).

**Using 10DLC campaigns in multiple AWS Regions**

When you register a company, that company is available in your AWS account in all AWS Regions. However, the same isn't true for 10DLC campaigns. A 10DLC campaign can only be used in the AWS Region that it was registered in.

If you plan to use 10DLC in more than one AWS Region, you must register separate 10DLC campaigns in each of those Regions. This step is necessary in order to comply with carrier requirements. You’re charged for each campaign that you register, even if the use case is exactly the same.

Registering multiple campaigns has the added benefit of increasing your throughput rates for messages that you send to recipients who use AT&T as their mobile carrier, because AT&T provides 10DLC throughput rates for each campaign. Compare this to the way that T-Mobile handles 10DLC throughput, which is based on a daily message allocation for each company (regardless of the number of campaigns).

**Editing or deleting a 10DLC campaign**

You can edit the HELP response, STOP response, and the sample messages for a 10DLC campaign using the Amazon Pinpoint console. You can also delete 10DLC campaigns using the console.
Editing a 10DLC campaign

After your campaign is approved, you can modify the HELP, STOP, and sample messages. You can also add additional sample messages. Changes to these fields don't require re-approval from the Campaign Registry or from carriers. You can't modify any other fields after the 10DLC campaign is approved.

You can have a maximum of five sample messages. You can't reduce the number of sample messages that you originally registered. For example, if you registered your campaign with three sample SMS messages, you can't reduce the number of sample SMS messages to less than three.

Note
If you want to modify any fields other than the HELP, STOP, and sample messages, you must first delete the 10DLC campaign and then recreate the campaign to include the updated information.

To edit 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. In the navigation pane, under SMS and voice, choose Phone numbers.
3. On the 10DLC campaigns tab, choose the 10DLC campaign that you want to edit.
4. In the Campaign messages section of the campaign details page, choose Edit.
5. Update any of the following fields:
   - Help message
   - Stop message
   - Sample SMS message

   You can't delete a previously added sample message, or delete the contents of a sample message so that the field is empty. If you delete the contents of a message without replacing that content, the original message will be used on updating.
6. Choose Update. A confirmation banner appears letting you know the campaign messages were updated.

Deleting a 10DLC campaign

You can delete a 10DLC campaign using the Amazon Pinpoint console. Before you delete a 10DLC campaign, you must first remove all of the phone numbers associated with that campaign.

Important
When you remove a 10DLC number from a campaign, you no longer have access to that number. Additionally, deleted 10DLC campaigns can't be restored.

To 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. In the navigation pane, under SMS and voice, choose Phone numbers.
3. On the 10DLC campaigns tab, choose the 10DLC campaign that you want to edit.
4. In the Phone numbers section, note the phone numbers associated with the campaign.
5. On the Phone numbers tab, choose the 10DLC number you want to remove, and then choose Remove phone number.

   Note
   This step is only required if you have multiple 10DLC phone numbers associated with the campaign. If you have only a single phone number associated with the 10DLC campaign
that number will appear on the **10DLC campaigns** tab. Note the number displayed on the tab.

6. Enter `delete` into the confirmation box, and then choose Confirm. A success message appears at the top of the SMS and voice page.

7. Repeat the previous two steps for each 10DLC number associated with the campaign.

8. After removing any numbers associated with the 10DLC campaign, choose the **10DLC campaigns** tab.

9. Choose the 10DLC campaign you want to delete.

10. In the upper right-hand corner of the **10DLC campaign details** page, choose Delete.

11. Enter `delete` into the confirmation box, and then choose Confirm. A success message appears at the top of the SMS and voice page.

**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. 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. 228).

**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/](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 **10 DLC - Associate existing US long code to 10DLC campaign**.
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

Each 10DLC phone number is associated with a single account in a single AWS Region. If you want to use the same 10DLC phone number to send messages in more than one account or Region, you have two options:

1. You can register the same company and campaign in each of your AWS accounts. These registrations are managed and charged separately. If you register the same company in multiple AWS accounts, the number of messages that you can send to T-Mobile customers per day is shared across each of those accounts.
2. You can complete the 10DLC registration process in one AWS account, and use AWS Identity and Access Management (IAM) to grant other accounts permission to send through your 10DLC number.

**Note**

This option allows for true cross-account access to your 10DLC phone numbers. However, note that messages sent from your secondary accounts are treated as if they were sent from your primary account. Quotas and billing are counted against the primary account and not against any secondary accounts.

Setting up cross-account access using IAM policies

You can use IAM roles to associate other accounts with your main account. Then, you can delegate access permissions from your primary account to your secondary accounts by granting them access to the 10DLC numbers in the primary account.

**To grant access to a 10DLC number in your primary account**

1. If you haven’t already done so, complete the 10DLC registration process in the primary account. This process involves three steps:
   - Register your company. For more information, see Registering your company or brand (p. 225) for use with 10DLC.
   - Register your 10DLC campaign (use case). For more information, see Registering a 10DLC campaign (p. 228).
   - Associate a phone number with your 10DLC campaign. For more information, see Associating a long code with a 10DLC campaign (p. 232).
2. Create an IAM role in your primary account that allows another account to call the Publish API operation for your 10DLC phone number. For more information on creating roles, see Creating IAM roles in the IAM User Guide.
3. Delegate and test access permission from your primary account using IAM roles with any of your other accounts that need to use your 10DLC numbers. For example, you might delegate access permission from your Production account to your Development account. For more information about delegating and testing permissions, see Delegate access across AWS account using IAM roles in the IAM User Guide.
4. Using the new role, send a message using a 10DLC number from the main account. For more information about using a role, see Using IAM roles in the IAM User Guide.

Getting information about 10DLC registration issues

In some situations, you could receive an error message when you attempt to register your company or your 10DLC campaign.

**Company registration issues**

When you register your company, you see one of two registration statuses: **Verified** or **Unverified**. If the company registration status is **Verified**, then your company registration was successful. You can begin to create 10DLC campaigns.
If the status for your company registration is **Unverified**, there were issues with the information that you provided. The Amazon Pinpoint console provides information about the reasons that your company registration received this status.

**To view registration issues for your 10DLC company registration**

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at [https://console.aws.amazon.com/pinpoint/](https://console.aws.amazon.com/pinpoint/).
2. In the navigation pane, under **SMS and voice**, choose **Phone numbers**.
3. On the **10DLC campaigns** tab, in the list of campaigns, choose the name of the company that you want to find more information about.
4. The company detail page contains information about the issues that were identified in your registration. If a field in the **Company info** section contains a warning symbol, the registration issue is related to the information in that field.

   Check the information that you provided and confirm that all of the fields contain the correct information. You can edit your company registration in the Amazon Pinpoint console. For more information about modifying your company registration details, see [Editing or deleting a registered company](p. 227).

**Campaign registration issues**

When you register your 10DLC campaign, you might see an error message in certain situations.

If you're unable to identify the issue with your registration, you can create a case with the AWS Support Center to request additional information. Use the following procedures to create an AWS Support case. The AWS Support team will provide information about the reasons why your 10DLC campaign registration was rejected.

**To submit a request for information about a rejected 10DLC campaign**

1. Sign in to the AWS Management Console and open the Amazon Pinpoint console at [https://console.aws.amazon.com/pinpoint/](https://console.aws.amazon.com/pinpoint/).
2. Choose **Support**, and then **Support Center**.
3. On the Support page, choose **Create case**.
4. For **Case type**, choose **Service limit increase**.
5. For **Limit type**, choose **Pinpoint SMS**.
6. In the **Requests** section, do the following:
   - For **Region**, choose the AWS Region that you attempted to register the campaign in.
   - For **Resource Type**, choose **10DLC Registration**.
   - For the **Limit**, choose **Company or 10DLC Campaign Registration Rejection**.
7. For **Use case description**, enter the rejected 10DLC campaign ID.
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**

**Important**

US mobile carriers have recently changed their regulations and require all toll-free numbers (TFNs) to complete a registration process with a regulatory body before September 30, 2022.
Check the status of your TFN by going to the section called “Toll-free number registration status” (p. 240). For more information about registering your company see the section called “Registering your toll-free number” (p. 237).

Allow up to 15 business days for processing after submitting your registration.

**Update September 19, 2022:** Effective October 1, 2022, mobile carriers will apply the following industry-wide thresholds for messaging sent over any unregistered toll-free number:

- Daily limit: 2,000 messages
- Weekly limit: 12,000 messages
- Monthly limit: 25,000 messages

**We strongly encourage you to complete your registration as soon as possible. These limits are subject to decreases over time as carriers continue to restrict unregistered traffic.**

A toll-free number (TFN) 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 TFNs to send transactional messages only.

### Topics

- Guidelines for using toll-free numbers (p. 235)
- Purchase a toll-free number (p. 236)
- Register a toll-free number (p. 236)
- Advantages and disadvantages of toll-free numbers (p. 237)

### Guidelines for using toll-free numbers

TFNs are typically used only within the US for transactional messaging, such as registration confirmation or for sending one-time passwords. They can be used for both voice messaging and SMS. Average throughput is three message parts per second (MPS). However, this throughput is affected by character encoding. For more information about how character encoding affects message parts, see **SMS character limits in Amazon Pinpoint**. For more information about registering a TFN, see **Toll-free number registration requirements and process** (p. 237).

Each customer account can have up to five TFNs. If you're sending over 15 text messages per second but less than 100, we recommend that you register one or more **10DLC origination IDs** (p. 225). If your use cases require sending more than 100 text messages per second, we recommend that you purchase and register one or more **short codes** (p. 242).

When using a TFN 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. 221) or **short code** (p. 242).

  Using short codes and 10DLCs require that you register your message template, where you can specify a shortened URL.

- Be aware that keywords 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. You also can't modify messages that are sent when users reply with STOP and UNSTOP.

- Don't send the same or similar message contents using multiple TFNs. Carriers call this practice **snowshoeing** or **number pooling** and target these messages for filtering.

- Any messages related to the following industries may be considered restricted and are subject heavy filtering or outright blocks. This can include one time passwords (OTP) and multi factor authentication (MFA) for services related to restricted categories.
If you had a registration denied for being a non-compliant use case and you feel this designation is incorrect, you can submit a request via support. For details on how to do this, see Registration issues (p. 240).

The following table describes the types of restricted content:

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gambling</td>
<td>• Casinos</td>
</tr>
<tr>
<td></td>
<td>• Sweepstakes</td>
</tr>
<tr>
<td></td>
<td>• Apps/websites</td>
</tr>
<tr>
<td>High-risk financial services</td>
<td>• Payday loans</td>
</tr>
<tr>
<td></td>
<td>• Short-term high-interest loans</td>
</tr>
<tr>
<td></td>
<td>• Auto loans</td>
</tr>
<tr>
<td></td>
<td>• Mortgage loans</td>
</tr>
<tr>
<td></td>
<td>• Student loans</td>
</tr>
<tr>
<td></td>
<td>• Debt collection</td>
</tr>
<tr>
<td></td>
<td>• Stock alerts</td>
</tr>
<tr>
<td></td>
<td>• Cryptocurrency</td>
</tr>
<tr>
<td>Debt forgiveness</td>
<td>• Debt consolidation</td>
</tr>
<tr>
<td></td>
<td>• Debt reduction</td>
</tr>
<tr>
<td></td>
<td>• Credit repair programs</td>
</tr>
<tr>
<td>Get-rich-quick schemes</td>
<td>• Work-from-home programs</td>
</tr>
<tr>
<td></td>
<td>• Risk-investment opportunities</td>
</tr>
<tr>
<td></td>
<td>• Pyramid or multi-level marketing schemes</td>
</tr>
<tr>
<td>Prohibited/controlled substances</td>
<td>• Cannabis/CBD</td>
</tr>
<tr>
<td>Phishing</td>
<td>• Attempts to get users to reveal personal information or website login information</td>
</tr>
<tr>
<td>S.H.A.F.T.</td>
<td>• Sex</td>
</tr>
<tr>
<td></td>
<td>• Hate</td>
</tr>
<tr>
<td></td>
<td>• Alcohol</td>
</tr>
<tr>
<td></td>
<td>• Firearms</td>
</tr>
<tr>
<td></td>
<td>• Tobacco/vape</td>
</tr>
</tbody>
</table>

**Purchase a toll-free number**

To purchase TFNs, use the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/. For more information, see Toll-free number registration requirements and process (p. 237).

Currently, Amazon Pinpoint supports toll-free numbers for both voice and SMS messages. Amazon SNS supports SMS messaging only.

**Register a toll-free number**

To register a TFN, see Toll-free number registration requirements and process (p. 237).
Advantages and disadvantages of toll-free numbers

Advantages

Toll-free originators have higher MPS over long codes as well as good deliverability.

Disadvantages

There's no control over opt-outs and opt-ins, as these are managed at the carrier level.

Do not include shortened URLs in your message, or use the number to send a promotional message. Instead use a 10DLC number or a short code. When you use a short code or 10DLC number, you need to register your message templates, which can contain shortened URLs and can be promotional messages. For more about short codes, see Short codes (p. 242). For more about 10DLC, see 10DLC (p. 221).

Toll-free number registration requirements and process

Important

US mobile carriers have recently changed their regulations and require all toll-free numbers (TFNs) to complete a registration process with a regulatory body before September 30, 2022. Check the status of your TFN by going to the section called “Toll-free number registration status” (p. 240). For more information about registering your company, see the section called “Registering your toll-free number” (p. 237).

Allow up to 15 business days for processing after submitting your registration.

Update September 19, 2022: Effective October 1, 2022, mobile carriers will apply the following industry-wide thresholds for messaging sent over any unregistered toll-free number:

- Daily limit: 2,000 messages
- Weekly limit: 12,000 messages
- Monthly limit: 25,000 messages

We strongly encourage you to complete your registration as soon as possible. These limits are subject to decreases over time as carriers continue to restrict unregistered traffic.

You can use toll-free phone numbers (TFNs) to deliver Amazon SNS messages to recipients. After you request a TFN, you can use it to register your company. Each TFN requires a single specific use case to use with it. For example, if you register a TFN to use for one-time passwords, the TFN can only be used for sending one-time passwords.

Important

A TFN can be revoked if it is used for any purpose other than its specified use case.

For guidelines on using TFNs, as well as their advantages and disadvantages, see Toll-free numbers (p. 234).

Toll-free number forbidden use cases

Amazon SNS has limited ability to send messages in cases where the messages are blocked (for example, use cases related to controlled substance, or phishing), or when high levels of filtering are expected (for example, high risk financial messages). You may be unable to register TFNs associated with restricted content use cases defined in Guidelines for using toll-free numbers (p. 235).

Registering your toll-free number

After purchasing a TFN, you need to register the number.

Topics

- Self-serve registration for toll-free numbers in Amazon Pinpoint regions (p. 238)
- Manual form-based registration process for toll-free numbers in regions other than Amazon Pinpoint regions (p. 239)
Self-serve registration for toll-free numbers in Amazon Pinpoint regions

If you've requested the TFN in the Amazon Pinpoint regions, complete the company registration process directly in the Amazon Pinpoint console. When registering your TFN, make sure the information is complete and accurate, or your registration can be rejected. The information you enter should be an exact match for your company's corporate head quarters.

To register a toll-free number

1. Before you can register, you have to purchase the TFN by following the directions in Toll-free numbers (p. 234).
2. Sign in to the AWS Management Console, and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
3. In the navigation pane under SMS and voice, choose Phone numbers.
4. On the Toll-free registrations tab, choose the Toll-free Phone number you want to register, then choose Create registration.
5. In the Company Information section, do the following:
   - **Company Name** – Enter the name of your company.
   - **Company Website** – Enter the URL for your company's website.
   - **Address 1** – Enter the street address of your corporate headquarters.
   - **Address 2 (Optional)** – Enter suite number of your corporate headquarters.
   - **City** – Enter the city of your corporate headquarters.
   - **State** – Enter the state of your corporate headquarters.
   - **ZIP Code** – Enter the ZIP code of your corporate headquarters.
   - **Country** – Enter the two digit ISO country code.
6. In the Contact Information section, do the following:
   - **First Name** – Enter the first name for the contact person of your business.
   - **Last Name** – Enter the last name for the contact person of your business.
   - **Support Email** – Enter the email address for the contact person of your business.
   - **Support Phone Number** – Enter the phone number for the contact person of your business.
7. In the Messaging Use Case section, do the following:
   - **Monthly SMS Volume** – Select the number of SMS messages for each month.
   - **Use Case Category** – Select one of the following use case types that the number will be used for:
     - **Two-factor authentication** – Use this for sending two-factor authentication codes.
     - **One-time passwords** – Use this for sending a user a one-time password.
     - **Notifications** – Use this if you only intend to send your users important notifications.
     - **Polling and surveys** – Use this to poll users on their preferences.
     - **Info on demand** – Use this for sending users messages after they have sent a request.
     - **Promotions and Marketing** – Use this if you only intend to send marketing messages to your users.
     - **Other** – Use this if your use case doesn't fall into any other category. Complete the Use Case Details field.
• **Use Case Details (Optional)** – Use this field to give more context to the selected Use Case Category.

• **Opt-in Workflow Description** – Enter a description of how users will consent to receive SMS messages. For example, did they give consent by filling out an online form on your website.

• **Opt-in Workflow File** – Upload an image of how users consent to receiving messages. Supported file types are PNG and ZIP. Upload multiple images using a single ZIP file. The maximum file size is 400 KB. Examples of opt-in screenshots are as follows:
  - **Website opt-in** – Provide screenshots of a webform where the client adds their number and agrees to receive messages.
  - **Website Posting (Support)** – Provide where the number is advertised, and where the customer finds the number to text.
  - **Keyword or QR Code Opt-in** – Provide where the customer finds the keyword or QR code to opt-in to these messages.
  - **2FA/OTP** – Provide a screenshot of the opt-in (if applicable). If it is verbal, provide a screenshot of the verbal opt-in script.
  - **Informational** – Provide a screenshot of a verbal consent workflow and the messaging content.

8. In the **Messaging Samples** section, do the following:

• **Message Sample 1** – Enter an example message of the SMS message body that will be sent to your end users.

• **Message Sample 2 (Optional) and Message Sample 3 (Optional)** – Enter more example messages of the SMS message body that will be sent to your end users.

9. Choose **Submit registration**.

**Manual form-based registration process for toll-free numbers in regions other than Amazon Pinpoint regions**

1. Download this [US_TFN_Registration.zip](#) and use the example registration form ([AWS US Toll-Free Registration Form-Business - Final.docx](#)) to complete the required information in the TFN registration CSV file ([bulkUStfn - Final.csv](#)).

   Each registration request or use case can only have up to five TFNs. If you believe you qualify for an exemption to this rule, provide a detailed explanation for consideration. List all phone numbers associated with the registration or use case.

2. Create a case with [AWS Support](#). Attach your completed CSV file to the case, and submit the TFN registration request.

**Key point to note**

1. Registrations can take up to two weeks to process once all the required information has been submitted. If information is missing or incomplete, the registration process will be delayed. If your registration is rejected, we'll help you find the reason why it was denied and suggest methods to improve your campaign so it can be registered.

2. TFNs work well for transactional use cases such as multi-factor authentication (MFA) where limited throughput is required. Each TFN can send up to three text messages parts per second, and each customer account can have up to five TFNs. If you're sending over 15 text messages parts per second but less than 100, we recommend you register one or more 10DLC (p. 221) origination IDs. If your use cases require sending more than 100 text messages per second, we recommend you purchase and register one or more short codes (p. 242). For more details, see **Guidelines for using toll-free numbers** (p. 235).
Toll-free number registration status

After you've registered a TFN, there are five different statuses that the registration can be in:

- **Created** – Your registration has been created, but hasn't been submitted yet.
- **Reviewing** – Your registration has been accepted and is being reviewed. It may take up to 15 business days for the review to be completed.
- **Complete** – Your registration has been approved and you can start using the TFN to start sending messages.
- **Requires Updates** – You need to fix your registration and resubmit it by following the directions in the section called “Editing a toll-free number registration” (p. 240). Fields that require updates will have a warning icon and a brief description of the issue.
- **Closed** – You deleted the TFN and need to delete the registration for the number.

Check your registration status

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. In the navigation pane, under **SMS and voice**, choose **Phone numbers**.
3. On the **Toll-free registrations** tab, choose the **Toll-free Phone number**.
4. You can view the registration status of each TFN.

Editing a toll-free number registration

If there is an issue with the registration after you've submitted it, the registration status will be set to **Requires Updates**. You can make changes to the registration form in this state. Fields that require updates will have a warning icon and a brief description of the issue.

Deleting a toll-free number registration

After you've submitted the TFN registration, you can delete the number. It is recommend that you delete the TFN before the registration.

To delete a registration

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. In the navigation pane, under **SMS and voice**, choose **Phone numbers** tabs.
3. On the **Toll-free registrations** tab, choose the **Registration ID** you want to delete, then select the **Delete Registration** button.

Registration issues

If your toll-free number registration is not accepted, you will see a message explaining why it was rejected.

To submit a request for information about a rejected toll-free number

1. 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 **Case type**, choose **Service limit increase**.
5. For **Limit type**, choose **Pinpoint SMS**.
6. In the **Requests** section, do the following:
• Choose the place where you attempted to register the campaign.
  • For Resource Type, choose 10DLC or TFN Registration.
  • For the Limit, choose Company or Campaign Registration Rejection.

7. For Use case description, enter the rejected toll-free number.

8. Under Contact options, for Preferred contact language, choose the language that you prefer to use when communicating with AWS support.

9. For Contact method, choose your preferred method of communicating with AWS support.

10. Choose Submit.

Toll-free number frequently asked questions

Frequently asked questions about the TFN registration process.

Do I currently own a toll-free number?

To check if you own a toll-free number

• Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
• In the navigation pane, under SMS and voice, choose Phone numbers.
• The TFN type is listed as toll-free.

Do I have to register my toll-free number?

Yes. To continue using a TFN you currently own, you must register it before September 30, 2022. If you are purchasing a new TFN after September 30, 2022, you must register it before you can send messages.

How do I purchase a toll-free number?

Follow the directions at Requesting a phone number using the Amazon Pinpoint console to purchase a TFN.

How do I register my toll-free number?

Follow the directions at the section called “Registering your toll-free number” (p. 237) to register a TFN.

What is the registration status of my toll-free number and what does it mean?

Follow the directions at the section called “Toll-free number registration status” (p. 240) to check your registration and status.

What information do I need to provide?

You need to provide your company's address, a business contact, and a use case for the TFN. You can find the required information at the section called “Registering your toll-free number” (p. 237).

What if my registration is rejected?

If your registration is rejected, the status is changed to Requires Updates. To make updates, see the section called “Editing a toll-free number registration” (p. 240).

What permissions do I need?

The IAM user/role you use to visit the Amazon Pinpoint console must have the “sms-voice:*” permission, otherwise you'll get an access denied error.
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.

Advantages

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 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 originating 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.

Disadvantages

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. 245).
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. 308).

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. 247).

Advantages
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.

Disadvantages
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).

A 10DLC number is used only for sending messages within the US. Using a 10DLC number requires that you register your company brand and the campaign that you want to associate the number with. Once approved you can request a 10DLC phone number on the SMS and voice page of the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/. Once requested, the time to receive approval is 7-10 days. The number can't be used with any other campaigns.

U.S. product number comparison

This table shows the support comparison for U.S. phone number types.

<table>
<thead>
<tr>
<th>Product feature</th>
<th>Short code</th>
<th>Toll-free number</th>
<th>10DLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number format</td>
<td>5-6 digits</td>
<td>10-digit number</td>
<td>10-digit number</td>
</tr>
<tr>
<td>Channel support</td>
<td>SMS</td>
<td>SMS</td>
<td>SMS</td>
</tr>
<tr>
<td>SMS traffic type</td>
<td>Promotional and transactional</td>
<td>Transactional</td>
<td>Promotional and transactional</td>
</tr>
<tr>
<td>Product feature</td>
<td>Short code</td>
<td>Toll-free number</td>
<td>10DLC</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Requires vetting</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Estimated provisioning time</td>
<td>12 weeks¹</td>
<td>Immediately</td>
<td>1 week</td>
</tr>
<tr>
<td>SMS throughput (number of SMS</td>
<td>100 message</td>
<td>3 message parts</td>
<td>Varies based on your 10DLC registration. Supports up to 100 message parts per second.</td>
</tr>
<tr>
<td>messages per second)²</td>
<td>parts per second; higher throughput available for an additional fee.</td>
<td>per second</td>
<td></td>
</tr>
<tr>
<td>Keywords required</td>
<td>Opt-in, opt-out, and HELP</td>
<td>STOP, UNSTOP. These are network-managed. You cannot change the opt-out and opt back in messages.</td>
<td>Opt-in, opt-out, and HELP</td>
</tr>
</tbody>
</table>

¹ Provisioning estimate doesn't include approval time.

² For more information on the maximum size for SMS messages, see [Publishing to a mobile phone](p. 260).

### 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](p. 260) 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. 218) so that you can send SMS messages without restrictions. For more information, see [Moving out of the SMS sandbox](p. 219).

- A dedicated [origination number](p. 221)

- A dedicated [sender ID](p. 221)

  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. 288).

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. 245)
• Requesting 10DLC numbers, toll-free numbers, and P2P long codes for SMS messaging with Amazon SNS (p. 247)
• Requesting sender IDs for SMS messaging with Amazon SNS (p. 248)
• Requesting increases to your monthly SMS spending quota for Amazon SNS (p. 250)

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. 250).

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. 308).
To request 10DLC numbers (p. 221), toll-free numbers (p. 234), and P2P long codes (p. 243), use the Amazon Pinpoint console. For detailed instructions, see Requesting a number in the Amazon Pinpoint User Guide.

**Important**
US mobile carriers have recently changed their regulations, and will require that all toll-free numbers (TFNs) complete a registration process with a regulatory body before September 30, 2022. For more information about registering a toll-free number, see Toll-free number registration requirements and process (p. 237).

If you purchased your toll-free number on or before September 30, 2022 its status will be in the Active state until October 1, 2022, unless you've completed your registration and it has been returned with the state set to Completed. Otherwise, it will be placed in the Pending state and you will not be able to send messages with it until you've register the number, the registration has been returned or the registration is set to the Active state. Registration can take up to 15 business days.

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, use the AWS Service Quotas console to create a service limit increase case while requesting the 10DLC number for that Region. 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. 250).

### 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. 250).

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. 288).

**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. 288) 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 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 Support Center.
2. Under Create case, choose Service limit increase, and then under Case details, perform the following steps:
   a. For Limit 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. 288).

   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.

3. 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.

4. 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**

Amazon SNS provides spending quotas to help you manage the maximum per-month cost incurred by sending SMS using your account. The spending quota limits your risk in case of malicious attack, and prevents your upstream application from sending more messages than expected. You can configure Amazon SNS to stop publishing SMS messages when it determines that sending an SMS message will incur a cost that exceeds your spending quota for the current month.

To ensure your operations are not impacted, we recommend requesting a spending quota high enough to support your production workloads. For more information, see Step 1: Open an Amazon SNS SMS case (p. 251). Once you have received the quota, you can manage your risk by applying the full quota, or a smaller value, as described in Step 2: Update your SMS settings (p. 251). By applying a smaller value, you can control your monthly spending with the option to scale up if necessary.

**Important**

Because Amazon SNS is a distributed system, it stops sending SMS messages within minutes if the spending quota is 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. 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. Sign in to the AWS Support Center.
2. Under Create case, choose Service limit increase, and then under Case details, perform the following steps:
3. For Limit type, choose SNS Text Messaging.
4. For Provide a link to the site or app which will be sending SMS messages - optional, enter the URL of your website or application.
5. For What type of messages do you plan to send? - optional, choose One Time Password, Promotional, or Transactional, depending on what you plan to send.
6. For Targeted Countries - optional, choose General Limits.
7. Under Requests, for Request 1, do the following:
8. For Resource Type, choose General Limits.
9. For New limit, enter the needed spend limit that you calculated earlier.
10. Under Case description, for Use case description, enter the description that you wrote earlier.
11. Expand Contact options, and then choose your preferred contact language.
12. Choose Submit.
13. 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.
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. 260).

Topics
- Setting SMS messaging preferences using the AWS Management Console (p. 252)
- Setting preferences (AWS SDKs) (p. 253)

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. 288).
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](https://docs.aws.amazon.com/sns/latest/dg/API_SetSMSAttributes.html).

The following code examples show how to set the default settings for sending SMS messages using Amazon SNS.

**C++**

**SDK for C++**

   **Note**
   There’s more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://aws.amazon.com/code-examples/).

How to use Amazon SNS to set the `DefaultSMSType` attribute.

```cpp
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);
```
• For API details, see **SetSmsAttributes** in *AWS SDK for C++ API Reference*.

### Java

**SDK for Java 2.x**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the **AWS Code Examples Repository**.

```java
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    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);
        }
    }
}
```

• For API details, see **SetSmsAttributes** in *AWS SDK for Java 2.x API Reference*.

### JavaScript

**SDK for JavaScript V3**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the **AWS Code Examples Repository**.

Create the client in a separate module and export it.

```javascript
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: {
    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();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](https://aws.amazon.com/documentation/js/).
- For API details, see [SetSMSAttributes](https://aws.amazon.com/documentation/js/api-ref/) in [AWS SDK for JavaScript API Reference](https://aws.amazon.com/documentation/js/api-ref/).

**PHP**

**SDK for PHP**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-code-examples).

```
$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());
}
```
Sending SMS messages

This section describes how to send SMS messages.

Topics

- Publishing to a topic (p. 256)
- Publishing to a mobile phone (p. 260)

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. 256)
- Sending a message to a topic (AWS SDKs) (p. 258)

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. 288).
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. 218).

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. 260).
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 2.x**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a topic and return its ARN.

```java
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 "";
}
```

Subscribe an endpoint to a topic.

```java
public static void subTextSNS(SnsClient snsClient, String topicArn, String phoneNumber) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("sms")
            .endpoint(phoneNumber)
            .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);
    }
```
Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```java
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    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);
        }
    }
}
```

Publish a message to a topic. The message is sent to every subscriber.

```java
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("Message sent. Status was " + result.sdkHttpResponse().statusCode());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```
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. 256).

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. 220). 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. 260)
- Sending a message (AWS SDKs) (p. 261)

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. 288).
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. 288).

   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. 220).

   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. 309).

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. 288).

If you don't specify a sender ID, a long code (p. 243) 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. 221).

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.

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

```csharp
using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;
using System;
using System.Threading.Tasks;

namespace SNSMessageExample
{
    class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
        /// Constructs a new SNSMessage object initializing the Amazon Simple
        /// Notification Service (Amazon SNS) client using the supplied
        /// Region endpoint.
        /// </summary>
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
        /// sending test messages with this object.</param>
        public SNSMessage(RegionEndpoint regionEndpoint)
        {
            snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);
        }

        /// <summary>
        /// Sends the SMS message passed in the text parameter to the phone number
        /// in phoneNum.
        /// </summary>
        /// <param name="phoneNum">The ten-digit phone number to which the text
        /// message will be sent.</param>
        /// <param name="text">The text of the message to send.</param>
        /// <returns></returns>
        public async Task SendTextMessageAsync(string phoneNum, string text)
        {
            if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
            {
                return;
            }

            // Now actually send the message.
            var request = new PublishRequest
            {
                Message = text,
                PhoneNumber = phoneNum
            };
```
try {
    var response = await snsClient.PublishAsync(request);
} catch (Exception ex) {
    Console.WriteLine($"Error sending message: {ex}");
}

• For API details, see Publish in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

/**
 * 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
 * use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
 * 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);

        try {
            var response = await snsClient.PublishAsync(request);
        } catch (Exception ex) {
            Console.WriteLine($"Error sending message: {ex}");
        }
    }
}
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;

• For API details, see Publish in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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);
    }
}

• For API details, see Publish in AWS SDK for Java 2.x API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.
Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun pubTextSMS(messageVal: String?, phoneNumberVal: String?) {
    val request = PublishRequest {
        message = messageVal
        phoneNumber = phoneNumberVal
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}

• For API details, see Publish in AWS SDK for Kotlin API reference.

PHP

SDK for PHP

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require '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:
 */

$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());
}
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. 268)
- Viewing Amazon CloudWatch metrics and logs for SMS deliveries (p. 268)
- Viewing daily SMS usage reports (p. 270)
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. 288).
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. 270).

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. 268)
- Viewing CloudWatch Logs (p. 268)
- Example log for successful SMS delivery (p. 269)
- Example log for failed SMS delivery (p. 269)
- SMS delivery failure reasons (p. 270)

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. 486).

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. 288).
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.

   The service role you specify must allow write access to your account's resources. For more information on creating IAM roles, see **Creating a role for an AWS service** in the **IAM User Guide**.
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:

```json
{
  "notification": {
    "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
    "timestamp": "2016-06-28 00:40:34.558"
  },
  "delivery": {
    "phoneCarrier": "My Phone Carrier",
    "mnc": 270,
    "numberOfMessageParts": 1,
    "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:

```json
{
  "notification": {
    "messageId": "1077257a-92f3-5ca3-bc97-6a915b310625",
```

269
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. 270)
- Subscribing to daily usage reports (p. 271)

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
Monitoring SMS activity

- 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. 272).

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.

4. On the Edit text messaging preferences page, in the Details section, specify the Amazon S3 bucket name for usage reports.
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.

AWS provides tools for all services with service principals that have been given access to resources in your account. When the principal in an Amazon S3 bucket policy statement is an **AWS service principal**, you can use the `aws:SourceArn` or `aws:SourceAccount` global condition keys to protect against the **confused deputy problem**. To limit which region and account from which the bucket can receive daily usage reports, use `aws:SourceArn` as shown in the example below. If you do not wish to limit which regions can generate these reports, use `aws:SourceAccount` to limit based on which account is generating the reports. If you don’t know the ARN of the resource, use `aws:SourceAccount`.

Use the following example that includes confused deputy protection when you create an Amazon S3 bucket to receive daily SMS usage reports from Amazon SNS.

```json
{
  "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"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:sns:region: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"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:sns:region:account_id:*"
      }
    }
  },
  { "Sid": "AllowListBucket",
    "Effect": "Allow",
    "Principal": { "Service": "sns.amazonaws.com" },
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::my-s3-bucket",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "account_id"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:sns:region:account_id:*"
      }
    }
  }
}
```
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. 273)
- Managing phone numbers and subscriptions (console) (p. 274)
- Managing phone numbers and subscriptions (AWS SDKs) (p. 275)

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:
To opt out, the recipient must reply to the same origination number (p. 221) 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. 288).
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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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" + result.phoneNumbers());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

• For API details, see ListPhoneNumbersOptedOut in AWS SDK for Java 2.x API Reference.

PHP

SDK for PHP

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
### 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**

**Note**

There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use the Amazon Simple Notification Service (Amazon SNS) to check whether a phone number has been opted out. The example was created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
```
public class IsPhoneNumOptedOut
{
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";
        IAmazonSimpleNotificationService client = new
        AmazonSimpleNotificationServiceClient();

        await CheckIfOptedOutAsync(client, phoneNumber);
    }
}

/// <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
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
{
    var request = new CheckIfPhoneNumberIsOptedOutRequest
    {
        PhoneNumber = phoneNumber,
    };

    try
    {
        var response = await
        client.CheckIfPhoneNumberIsOptedOutAsync(request);

        {
            string optOutStatus = response.IsOptedOut ? "opted out" : "not
            opted out.";
            Console.WriteLine($"The phone number: {phoneNumber} is
            {optOutStatus}";
        }
    }
    catch (AuthorizationErrorException ex)
    {
        Console.WriteLine($"{ex.Message}";)
    }
}

• For API details, see CheckIfPhoneNumberIsOptedOut in AWS SDK for .NET API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

```java
public static void checkPhone(SnsClient snsClient, String phoneNumber) {
    try {
        CheckIfPhoneNumberIsOptedOutRequest request =
        CheckIfPhoneNumberIsOptedOutRequest.builder()
```
For API details, see CheckIfPhoneNumberIsOptedOut in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
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());
}
```

• 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.
Managing SMS subscriptions

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
/// <summary>
/// Given the ARN for an Amazon SNS subscription, this method deletes
/// the subscription.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
/// to delete an Amazon SNS subscription.</param>
/// <param name="subscriptionArn">The ARN of the subscription to delete.</param>
public static async Task TopicUnsubscribeAsync(
    IAmazonSimpleNotificationService client,
    string subscriptionArn)
{
    var response = await client.UnsubscribeAsync(subscriptionArn);
}
```

- For API details, see Unsubscribe in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);
```
• For API details, see Unsubscribe in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```

• For API details, see Unsubscribe in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see Unsubscribe in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun unSub(subscriptionArnVal: String) {
  val request = UnsubscribeRequest {
    subscriptionArn = subscriptionArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    snsClient.unsubscribe(request)
    println("Subscription was removed for ${request.subscriptionArn}"")
  }
}

• For API details, see Unsubscribe in AWS SDK for Kotlin API reference.

PHP

SDK for PHP

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require '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:
$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());
}

• 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)**

**Note**
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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

• 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**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;

/// <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()
    {
        IAmazonSimpleNotificationService client = new AmazonSimpleNotificationServiceClient();

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

*For API details, see [DeleteTopic](#delete-topic-in-aws-sdk-for-net-api-reference) in AWS SDK for .NET API Reference.*

**C++**

**SDK for C++**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);

• For API details, see DeleteTopic in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}

• For API details, see DeleteTopic in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
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:
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());
}

• For API details, see DeleteTopic in AWS SDK for PHP API Reference.

Python

SDK for Python (Boto3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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

• 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:

<table>
<thead>
<tr>
<th>Region name</th>
<th>Region</th>
<th>Endpoint</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
<td>sns.us-east-2.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>sns.us-east-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td>sns.us-west-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>sns.us-west-2.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Africa (Cape Town)</td>
<td>af-south-1</td>
<td>sns.af-south-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
<td>sns.ap-south-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Osaka)</td>
<td>ap-northeast-3</td>
<td>sns.ap-northeast-3.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td>sns.ap-southeast-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td>sns.ap-southeast-2.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td>sns.ap-northeast-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
<td>sns.ca-central-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td>eu-central-1</td>
<td>sns.eu-central-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td>eu-west-1</td>
<td>sns.eu-west-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (London)</td>
<td>eu-west-2</td>
<td>sns.eu-west-2.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (Milan)</td>
<td>eu-south-1</td>
<td>sns.eu-south-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (Paris)</td>
<td>eu-west-3</td>
<td>sns.eu-west-3.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Europe (Stockholm)</td>
<td>eu-north-1</td>
<td>sns.eu-north-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Middle East (Bahrain)</td>
<td>me-south-1</td>
<td>sns.me-south-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
</tbody>
</table>
## Supported Regions and countries

<table>
<thead>
<tr>
<th>Region name</th>
<th>Region</th>
<th>Endpoint</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>South America (São Paulo)</td>
<td>sa-east-1</td>
<td>sns.sa-east-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>AWS GovCloud (US-East)</td>
<td>us-gov-east-1</td>
<td>sns.us-gov-east-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>AWS GovCloud (US-West)</td>
<td>us-gov-west-1</td>
<td>sns.us-gov-west-1.amazonaws.com</td>
<td>HTTP and HTTPS</td>
</tr>
</tbody>
</table>

You can use Amazon SNS to send SMS messages to the following countries and regions:

<table>
<thead>
<tr>
<th>Country or region</th>
<th>ISO code</th>
<th>Dialing code</th>
<th>Supports short codes</th>
<th>Supports long codes</th>
<th>Supports Sender IDs</th>
<th>Supports two-way SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>AF</td>
<td>93</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Albania</td>
<td>AL</td>
<td>355</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Algeria</td>
<td>DZ</td>
<td>213</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Andorra</td>
<td>AD</td>
<td>376</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Anguilla</td>
<td>AI</td>
<td>1-264</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Antigua and Barbuda</td>
<td>AG</td>
<td>1-268</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Argentina</td>
<td>AR</td>
<td>54</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Armenia</td>
<td>AM</td>
<td>374</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Aruba</td>
<td>AW</td>
<td>297</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
<td>61</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Austria</td>
<td>AT</td>
<td>43</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Azerbaijan</td>
<td>AZ</td>
<td>994</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bahamas</td>
<td>BS</td>
<td>1-242</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Bahrain</td>
<td>BH</td>
<td>973</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bangladesh</td>
<td>BD</td>
<td>880</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Barbados</td>
<td>BB</td>
<td>1-246</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Belarus</td>
<td>BY</td>
<td>375</td>
<td>No</td>
<td>No</td>
<td>Registration required</td>
<td>No</td>
</tr>
<tr>
<td>Belgium</td>
<td>BE</td>
<td>32</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Belize</td>
<td>BZ</td>
<td>501</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Bermuda</td>
<td>BM</td>
<td>1-441</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bhutan</td>
<td>BT</td>
<td>975</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bolivia</td>
<td>BO</td>
<td>591</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bosnia and Herzegovina</td>
<td>BA</td>
<td>387</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Botswana</td>
<td>BW</td>
<td>267</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>55</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Brunei</td>
<td>BN</td>
<td>673</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bulgaria</td>
<td>BG</td>
<td>359</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>BF</td>
<td>226</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Burundi</td>
<td>BL</td>
<td>257</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cambodia</td>
<td>KH</td>
<td>855</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cameroon</td>
<td>CM</td>
<td>237</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cape Verde</td>
<td>CV</td>
<td>238</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cayman Islands</td>
<td>KY</td>
<td>1-345</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Central African Republic</td>
<td>CF</td>
<td>236</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Chad</td>
<td>TD</td>
<td>235</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Chile</td>
<td>CL</td>
<td>56</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>China</td>
<td>CN</td>
<td>86</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes 2 (p. 298)</td>
</tr>
<tr>
<td>Colombia</td>
<td>CO</td>
<td>57</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Comoros</td>
<td>KM</td>
<td>269</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cook Islands</td>
<td>CK</td>
<td>682</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Costa Rica</td>
<td>CR</td>
<td>506</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Croatia</td>
<td>HR</td>
<td>385</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cyprus</td>
<td>CY</td>
<td>357</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Czechia (Czech Republic)</td>
<td>CZ</td>
<td>420</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Democratic Republic of the Congo</td>
<td>CD</td>
<td>243</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Denmark</td>
<td>DK</td>
<td>45</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Djibouti</td>
<td>DJ</td>
<td>253</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Dominica</td>
<td>DN</td>
<td>1-767</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Dominican Republic</td>
<td>DO</td>
<td>1-809, 1-829, 1-849</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ecuador</td>
<td>EC</td>
<td>593</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Egypt</td>
<td>EG</td>
<td>20</td>
<td>Yes</td>
<td>No</td>
<td>Registration required(^1) (p. 298)</td>
<td>Yes</td>
</tr>
<tr>
<td>El Salvador</td>
<td>SV</td>
<td>503</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Equatorial Guinea</td>
<td>GQ</td>
<td>240</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Eritrea</td>
<td>ER</td>
<td>291</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Estonia</td>
<td>EE</td>
<td>372</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ethiopia</td>
<td>ET</td>
<td>251</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Faroe Islands</td>
<td>FO</td>
<td>298</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Fiji</td>
<td>FJ</td>
<td>679</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Finland</td>
<td>FI</td>
<td>358</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>France</td>
<td>FR</td>
<td>33</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>French Guiana</td>
<td>GF</td>
<td>594</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>French Polynesia</td>
<td>PF</td>
<td>689</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

\(^1\) Registration required.
<table>
<thead>
<tr>
<th>Country or region</th>
<th>ISO code</th>
<th>Dialing code</th>
<th>Supports short codes</th>
<th>Supports long codes</th>
<th>Supports Sender IDs</th>
<th>Supports two-way SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gabon</td>
<td>GA</td>
<td>241</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Gambia</td>
<td>GM</td>
<td>220</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Georgia</td>
<td>GE</td>
<td>995</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Germany</td>
<td>DE</td>
<td>49</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ghana</td>
<td>GH</td>
<td>233</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Gibraltar</td>
<td>GL</td>
<td>350</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Greece</td>
<td>GR</td>
<td>30</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Greenland</td>
<td>GL</td>
<td>299</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Grenada</td>
<td>GD</td>
<td>1-473</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Guadeloupe</td>
<td>GP</td>
<td>590</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Guam</td>
<td>GU</td>
<td>1-671</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Guatemala</td>
<td>GT</td>
<td>502</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Guernsey</td>
<td>GG</td>
<td>44-1481</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Guinea</td>
<td>GN</td>
<td>224</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Guinea-Bissau</td>
<td>GW</td>
<td>245</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Guyana</td>
<td>GY</td>
<td>592</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Haiti</td>
<td>H</td>
<td>509</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Honduras</td>
<td>HN</td>
<td>504</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>HK</td>
<td>852</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Hungary</td>
<td>HU</td>
<td>36</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Iceland</td>
<td>IS</td>
<td>354</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
<td>91</td>
<td>Yes</td>
<td>Yes (^4) (p. 298)</td>
<td>Registration required (^3) (p. 298)</td>
<td>Yes</td>
</tr>
<tr>
<td>Indonesia</td>
<td>ID</td>
<td>62</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Iraq</td>
<td>IQ</td>
<td>964</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ireland</td>
<td>IE</td>
<td>353</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Isle of Man</td>
<td>IM</td>
<td>44-1624</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Israel</td>
<td>IL</td>
<td>972</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>--------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
<td>39</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ivory Coast</td>
<td>CL</td>
<td>225</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Jamaica</td>
<td>JM</td>
<td>1-876</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
<td>81</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Jersey</td>
<td>JE</td>
<td>44-1434</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Jordan</td>
<td>JO</td>
<td>962</td>
<td>No</td>
<td>No</td>
<td>Registration required</td>
<td>No</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>KZ</td>
<td>7</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Kenya</td>
<td>KE</td>
<td>254</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Kosovo</td>
<td>XK</td>
<td>383</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Kuwait</td>
<td>KW</td>
<td>965</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Kyrgyzstan</td>
<td>KG</td>
<td>996</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Laos</td>
<td>LA</td>
<td>856</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Latvia</td>
<td>LV</td>
<td>371</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Lebanon</td>
<td>LB</td>
<td>961</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Lesotho</td>
<td>LS</td>
<td>266</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Liberia</td>
<td>LR</td>
<td>231</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Libya</td>
<td>LY</td>
<td>218</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Liechtenstein</td>
<td>LI</td>
<td>423</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Lithuania</td>
<td>LT</td>
<td>370</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>LU</td>
<td>352</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Macau</td>
<td>MO</td>
<td>853</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Macedonia</td>
<td>MK</td>
<td>389</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Madagascar</td>
<td>MG</td>
<td>261</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Malawi</td>
<td>MW</td>
<td>265</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Malaysia</td>
<td>MY</td>
<td>60</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Maldives</td>
<td>MV</td>
<td>960</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mali</td>
<td>ML</td>
<td>223</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Malta</td>
<td>MT</td>
<td>356</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Marshall Islands, The</td>
<td>MH</td>
<td>692</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Martinique</td>
<td>MQ</td>
<td>596</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Mauritania</td>
<td>MR</td>
<td>222</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mauritius</td>
<td>MU</td>
<td>230</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mayotte</td>
<td>YT</td>
<td>262</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
<td>52</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Micronesia (Federated States of)</td>
<td>FM</td>
<td>691</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Moldova</td>
<td>MD</td>
<td>373</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Monaco</td>
<td>MC</td>
<td>377</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Mongolia</td>
<td>MN</td>
<td>976</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Montenegro</td>
<td>ME</td>
<td>382</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Montserrat</td>
<td>MS</td>
<td>1-664</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
<td>212</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mozambique</td>
<td>MS</td>
<td>258</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Myanmar</td>
<td>MM</td>
<td>95</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Namibia</td>
<td>NA</td>
<td>264</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Nepal</td>
<td>NP</td>
<td>977</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
<td>31</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Netherlands Antilles</td>
<td>AN</td>
<td>599</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>New Caledonia</td>
<td>NC</td>
<td>687</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
<td>64</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Nicaragua</td>
<td>NI</td>
<td>505</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Niger</td>
<td>NE</td>
<td>227</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Nigeria</td>
<td>NG</td>
<td>234</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Niue</td>
<td>NU</td>
<td>683</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>47</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Oman</td>
<td>OM</td>
<td>968</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Pakistan</td>
<td>PK</td>
<td>92</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Palestine</td>
<td>PS</td>
<td>970</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Panama</td>
<td>PA</td>
<td>507</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Papua New Guinea</td>
<td>PG</td>
<td>675</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Paraguay</td>
<td>PY</td>
<td>595</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Peru</td>
<td>PE</td>
<td>51</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Philippines</td>
<td>PH</td>
<td>63</td>
<td>Yes</td>
<td>Registration required</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Poland</td>
<td>PL</td>
<td>48</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Portugal</td>
<td>PT</td>
<td>351</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>PR</td>
<td>1-797, 1-939</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Qatar</td>
<td>QA</td>
<td>974</td>
<td>No</td>
<td>Registration required</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Republic of the Congo</td>
<td>CG</td>
<td>242</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Réunion (France)</td>
<td>RE</td>
<td>262</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Romania</td>
<td>RO</td>
<td>40</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Russia</td>
<td>RU</td>
<td>7</td>
<td>Yes</td>
<td>No</td>
<td>Registration required</td>
<td>Yes</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Rwanda</td>
<td>RW</td>
<td>250</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Saint Kitts and Nevis</td>
<td>KN</td>
<td>1-869</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Saint Lucia</td>
<td>LC</td>
<td>1-758</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Samoa</td>
<td>WS</td>
<td>685</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>San Marino</td>
<td>SM</td>
<td>378</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>São Tomé and Príncipe</td>
<td>ST</td>
<td>239</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Saudi Arabia</td>
<td>SA</td>
<td>966</td>
<td>No</td>
<td>Yes(^4) (p. 298)</td>
<td>Registration required(^1) (p. 298)</td>
<td>No</td>
</tr>
<tr>
<td>Senegal</td>
<td>SN</td>
<td>221</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Serbia</td>
<td>RS</td>
<td>381</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Seychelles</td>
<td>SC</td>
<td>248</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sierra Leone</td>
<td>SL</td>
<td>232</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Singapore</td>
<td>SG</td>
<td>65</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes(^5) (p. 298)</td>
<td>Yes</td>
</tr>
<tr>
<td>Slovakia</td>
<td>SK</td>
<td>421</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Slovenia</td>
<td>SI</td>
<td>386</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Solomon Islands</td>
<td>SB</td>
<td>677</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Somalia</td>
<td>SO</td>
<td>252</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>South Africa</td>
<td>ZA</td>
<td>27</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>South Korea</td>
<td>KR</td>
<td>82</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>South Sudan</td>
<td>SS</td>
<td>211</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Spain</td>
<td>ES</td>
<td>34</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sri Lanka</td>
<td>LK</td>
<td>94</td>
<td>No</td>
<td>No</td>
<td>Registration required(^1) (p. 298)</td>
<td>No</td>
</tr>
<tr>
<td>Suriname</td>
<td>SR</td>
<td>597</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Swaziland</td>
<td>SZ</td>
<td>268</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sweden</td>
<td>SE</td>
<td>46</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
<td>41</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\(T\)
<table>
<thead>
<tr>
<th>Country or region</th>
<th>ISO code</th>
<th>Dialing code</th>
<th>Supports short codes</th>
<th>Supports long codes</th>
<th>Supports Sender IDs</th>
<th>Supports two-way SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taiwan</td>
<td>TW</td>
<td>886</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tajikistan</td>
<td>TJ</td>
<td>992</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tanzania</td>
<td>TX</td>
<td>255</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Thailand</td>
<td>TH</td>
<td>66</td>
<td>No</td>
<td>Yes</td>
<td>Registration required</td>
<td>Yes</td>
</tr>
<tr>
<td>Timor-Leste</td>
<td>TL</td>
<td>670</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Togo</td>
<td>TG</td>
<td>228</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tonga</td>
<td>TO</td>
<td>676</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Trinidad and Tobago</td>
<td>TT</td>
<td>1-868</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tunisia</td>
<td>TN</td>
<td>216</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Turkey</td>
<td>TR</td>
<td>90</td>
<td>No</td>
<td>No</td>
<td>Registration required</td>
<td>No</td>
</tr>
<tr>
<td>Turkmenistan</td>
<td>TM</td>
<td>993</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Turks and Caicos Islands</td>
<td>TC</td>
<td>1-649</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tuvalu</td>
<td>TC</td>
<td>688</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uganda</td>
<td>UG</td>
<td>256</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ukraine</td>
<td>UA</td>
<td>380</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>United Arab Emirates (UAE)</td>
<td>AE</td>
<td>971</td>
<td>Yes</td>
<td>Yes</td>
<td>Registration required</td>
<td>Yes</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>GB</td>
<td>44</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>United States</td>
<td>US</td>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Uruguay</td>
<td>UY</td>
<td>598</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Uzbekistan</td>
<td>UZ</td>
<td>998</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vanuatu</td>
<td>VU</td>
<td>678</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
<td>58</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Country or region</td>
<td>ISO code</td>
<td>Dialing code</td>
<td>Supports short codes</td>
<td>Supports long codes</td>
<td>Supports Sender IDs</td>
<td>Supports two-way SMS</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Vietnam</td>
<td>VN</td>
<td>84</td>
<td>No</td>
<td>No</td>
<td>Registration required</td>
<td>No</td>
</tr>
<tr>
<td>Virgin Islands, British</td>
<td>VG</td>
<td>1-284</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Virgin Islands, US</td>
<td>VI</td>
<td>1-340</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yemen</td>
<td>YE</td>
<td>967</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Zambia</td>
<td>ZM</td>
<td>260</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>ZW</td>
<td>263</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes

1. Senders are required to use a pre-registered alphabetic Sender ID. To request a Sender ID from AWS Support, see Requesting sender IDs for SMS messaging with Amazon SNS (p. 248). 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, open an Amazon SNS SMS case with AWS Support. When you create the case, provide the same information that you would provide to request a sender ID. For more information, see Requesting sender IDs for SMS messaging with Amazon SNS (p. 248). 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.

   **Note**
   In order to send messages to China, you must first register your templates through AWS Support for approval.

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. 309).

4. Long codes in these countries only support inbound messaging. In other words, you cannot use these long codes to send messages to your recipients, but you can use them to receive messages from your recipients. These long codes are useful way to allow your recipients to opt-out if you send messages using an alphabetic Sender ID, because Sender IDs only support outbound messages.

5. 298
Amazon SNS can send SMS traffic to Singapore using a Sender ID that has been registered on the Singapore SMS Sender ID Registry (SSIR), a registry created by the Info-communications Media Development Authority (IMDA) of Singapore. For more information on requirements to use a Singapore Sender ID, see SMS requirements for Singapore (p. 307).

You can also send SMS traffic in Singapore using unregistered Sender IDs or alternative origination identity types such as Short Codes or Long Codes.

**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, regulations, and carrier requirements (p. 300)
- Obtain permission (p. 300)
- Don't send to old lists (p. 303)
- Audit your customer lists (p. 303)
- Keep records (p. 303)
- Make your messages clear, honest, and concise (p. 303)
- Respond appropriately (p. 305)
- Adjust your sending based on engagement (p. 306)
- Send at appropriate times (p. 306)
- Avoid cross-channel fatigue (p. 306)
- Use dedicated short codes (p. 306)
- Verify your destination phone numbers (p. 306)
- Design with redundancy in mind (p. 307)
- SMS limits and restrictions (p. 307)
- Managing opt out keywords (p. 307)
- CreatePool (p. 307)
- PutKeyword (p. 307)
- Managing number settings (p. 307)
Comply with laws, regulations, and carrier requirements

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](https://www.fcc.gov/) 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?](https://www.gov.uk/guidance/privacy-and-electronic-communications-regulations-2003) 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](https://eur-lex.europa.eu) 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](https://www.parlimentofcanada.ca) 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 [Japan's countermeasures against spam](https://www.mlit.go.jp) at the website of the Japanese Ministry of Internal Affairs and Communications.

As a sender, these laws may apply to you even if your company or organization isn't based 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.

In many countries, the local carriers ultimately have the authority to determine what kind of traffic flows over their networks. This means that the carriers might impose restrictions on SMS content that exceed the minimum requirements of local laws.

Obtain permission

Never send messages to recipients who haven't explicitly asked to receive the specific types of messages that you plan to send. Don't share opt-in lists, even among organizations within the same company.

If recipients can sign up to receive your messages by using an online form, add systems that prevent automated scripts from subscribing people without their knowledge. You should also limit the number of times a user can submit a phone number in a single session.

When you receive an SMS opt-in request, send the recipient a message that asks them to confirm that they want to receive messages from you. Don't send that recipient any additional messages until they confirm their subscription. A subscription confirmation message might resemble the following example:

```
Text YES to join ExampleCorp 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.
Opt-in workflow

In some cases (like US Toll-Free or Short Code registration) mobile carriers require you to provide mockups or screen shot of your entire opt-in workflow. The mockups or screen shot must closely resemble the opt-in workflow that your recipients will complete.

Your mockups or screen shot should include all of the required disclosures listed below to maintain the highest level of compliance.

**Required disclosures**

- A description of the messaging use case that you will send through your program.
- The phrase “Message and data rates may apply.”
- An indication of how often recipients will get messages from you. For example, a recurring messaging program might say “one message per week.” A one-time password or multi-factor authentication use case might say “message frequency varies” or “one message per login attempt.”
- Links to your Terms and Conditions and Privacy Policy documents.

**Common rejection reasons for non compliant opt-ins**

- If the provided company name does not match what is provided in the mockup or screen shot. Any non obvious relations should be explained in the opt-in workflow description.
- If it appears that a message will be sent to the recipient, but no consent is explicitly gathered before doing so. Explicit consent is a requirement of all messaging.
- If it appears that receiving a text message is required to sign up for a service. This is not compliant if the workflow doesn't provide any alternative to receiving an opt-in message in another form like email or a voice call.
- If the opt-in language is presented entirely in the Terms of Service. The disclosures should always be presented to the recipient at time of opt-in rather than housed inside a linked policy document.
- If a customer provided consent to receive one type of message from you and you send them other types of text messages. For example they consent to receive one-time passwords but are also sent polling and survey messages.
- If the required disclosures (listed above) are not presented to the recipients.

The following example complies with the mobile carriers’ requirements for a multi-factor authentication use case.
Mockup of multi-factor authentication use case

It contains finalized text and images, and it shows the entire opt-in flow, complete with annotations. In the opt-in flow, the customer has to take distinct, intentional actions to provide their consent to receive text messages and contains all of the required disclosures.

Other opt-in workflow types

Mobile carriers will also accept opt-in workflows outside of applications and websites like verbal or written opt-in if it complies with what is outlined above. A compliant opt-in workflow and verbal or
written script will gather explicit consent from the recipient to receive a specific message type. Examples of this include the verbal script a support agent uses to gather consent before recording into a service database or a phone number listed on a promotional flyer. To provide a mockup of these opt-in workflow types you can provide a screenshot of your opt-in script, marketing material or database where numbers are collected. Mobile carriers may have additional questions around these use cases if an opt-in is not clear or the use case exceed certain volumes.

Don't send to old lists

People change phone numbers often. A phone number that you gathered consent to contact two years ago might belong to somebody else today. Don't use an old list of phone numbers for a new messaging program; if you do, you're likely to have some messages fail because the number is no longer in service, and some people who opt out because they don't remember giving you their consent in the first place.

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 ExampleCorp 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.

Make your messages clear, honest, and concise

SMS is a unique medium. The 160-character-per-message limit means that your messages have to be concise. Techniques that you might use in other communication channels, such as email, might not apply to the SMS channel, and might even seem dishonest or deceptive when used with SMS messages. If the content in your messages doesn't align with best practices, recipients might ignore your messages; in the worst case, the mobile carriers might identify your messages as spam and block future messages from your phone number.

This section provides some tips and ideas for creating an effective SMS message body.

Identify yourself as the sender

Your recipients should be able to immediately tell that a message is from you. Senders who follow this best practice include an identifying name ('program name') at the beginning of each message.
Don't do this:

Your account has been accessed from a new device. Reply Y to confirm.

Try this instead:

ExampleCorp Financial Alerts: You have logged in to your account from a new device. Reply Y to confirm, or STOP to opt-out.

Don't try to make your message look like a person-to-person message

Some marketers are tempted to add a personal touch to their SMS messages by making their messages appear to come from an individual. However, this technique might make your message seem like a phishing attempt.

Don't do this:

Hi, this is Jane. Did you know that you can save up to 50% at Example.com? Click here for more info: https://www.example.com.

Try this instead:

ExampleCorp Offers: Save 25-50% on sale items at Example.com. Click here to browse the sale: https://www.example.com. Text STOP to opt-out.

Be careful when talking about money

Scammers often prey upon people’s desire to save and receive money. Don’t make offers seem too good to be true. Don’t use the lure of money to deceive people. Don’t use currency symbols to indicate money.

Don't do this:

Save big $$$ on your next car repair by going to https://www.example.com.

Try this instead:


Use only the necessary characters

Brands are often inclined to protect their trademarks by including trademark symbols such as ™ or ® in their messages. However, these symbols are not part of the standard set of characters (known as the GSM alphabet) that can be included in a 160-character SMS message. When you send a message that contains one of these characters, your message is automatically sent using a different character encoding system, which only supports 70 characters per message part. As a result, your message could be broken into several parts. Because you’re billed for each message part that you send, it could cost you more than you expect to spend to send the entire message. Additionally, your recipients might receive several sequential messages from you, rather than one single message. For more information about SMS character encoding, see SMS character limits in Amazon Pinpoint in the Amazon Pinpoint User Guide.

Don't do this:

ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget® at example.com and use the promo code WIDGET.
Try this instead:

ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget (R) at example.com and use the promo code WIDGET.

Note
The two preceding examples are almost identical, but the first example contains a Registered Trademark symbol (®), which is not part of the GSM alphabet. As a result, the first example is sent as two message parts, while the second example is sent as one message part.

Use valid, safe links

If your message includes links, double-check the links to make sure that they work. Test your links on a device outside your corporate network to ensure that links resolve properly. Because of the 160-character limit of SMS messages, very long URLs could be split across multiple messages. You should use redirect domains to provide shortened URLs. However, you shouldn't use free link-shortening services such as tinyurl.com or bitly.com, because carriers tend to filter messages that include links on these domains. However, you can use paid link-shortening services as long as your links point to a domain that is dedicated to the exclusive use of your company or organization.

Don't do this:

Go to https://tinyurl.com/4585y8mr today for a special offer!

Try this instead:

ExampleCorp Offers: Today only, get an exclusive deal on an ExampleCorp Widget. See https://a.co/cFKmaRG for more info. Text STOP to opt-out.

Limit the number of abbreviations that you use

The 160-character limitation of the SMS channel leads some senders to believe that they need to use abbreviations extensively in their messages. However, the overuse of abbreviations can seem unprofessional to many readers, and could cause some users to report your message as spam. It's completely possible to write a coherent message without using an excessive number of abbreviations.

Don't do this:

Get a gr8 deal on ExampleCorp widgets when u buy a 4-pack 2day.

Try this instead:

ExampleCorp Alerts: Today only—an exclusive deal on ExampleCorp Widgets at example.com. Text STOP to opt-out.

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: ExampleCorp alerts: email help@example.com or call 425-555-0199. 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:

You're unsubscribed from ExampleCorp alerts. No more messages will be sent. Reply HELP, email help@example.com, or call 425-555-0199 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.

If you send campaigns or journeys to very large audiences, double-check the throughput rates for your origination numbers. Divide the number of recipients by your throughput rate to determine how long it will take to send messages to all of your recipients.

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.

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. 245).

Verify your destination phone numbers

When you send SMS messages through Amazon SNS, you're billed for each message part you send. The price you pay per message part varies on the recipient's country or region. For more information about SMS pricing, see Amazon SNS Pricing.

When Amazon SNS accepts a request to send an SMS message (as the result of a call to the SendMessages API, or as the result of a campaign or journey being launched), you're charged for sending that message. This statement is true even if the intended recipient doesn't actually receive the message. For example, if the recipient's phone number is no longer in service, or if the number that you sent the message to wasn't a valid mobile phone number, you're still billed for sending the message.
Amazon SNS accepts valid requests to send SMS messages and attempts to deliver them. For this reason, you should validate that the phone numbers that you send messages to are valid mobile numbers. You can use the Amazon SNS phone number validation service to determine if a phone number is valid and what type of number it is (such as mobile, landline, or VoIP). For more information, see Validating phone numbers in Amazon Pinpoint in the Amazon Pinpoint Developer Guide.

**Design with redundancy in mind**

For mission-critical messaging programs, we recommend that you configure Amazon SNS in more than one AWS Region. Amazon SNS is available in several AWS Regions. For a complete list of Regions where Amazon SNS is available, see the AWS General Reference.

The phone numbers that you use for SMS messages—including short codes, long codes, toll-free numbers, and 10DLC numbers—can't be replicated across AWS Regions. As a result, in order to use Amazon SNS in multiple Regions, you must request separate phone numbers in each Region where you want to use Amazon SNS. For example, if you use a short code to send text messages to recipients in the United States, you need to request separate short codes in each AWS Region that you plan to use.

In some countries, you can also use multiple types of phone numbers for added redundancy. For example, in the United States, you can request short codes, 10DLC numbers, and toll-free numbers. Each of these phone number types takes a different route to the recipient. Having multiple phone number types available—either in the same AWS Region or spread across multiple AWS Regions—provides an additional layer of redundancy, which can help improve resiliency.

**SMS limits and restrictions**

For SMS limits and restrictions, see SMS limits and restrictions in Amazon Pinpoint in the Amazon Pinpoint User Guide.

**Managing opt out keywords**

SMS recipients can use their devices to opt out of messages by replying with a keyword. For more information, see Opting out of receiving SMS messages (p. 273).

**CreatePool**

Use the CreatePool API action to create a new pool and associate a specified origination identity to the pool. For more information, see CreatePool in Amazon Pinpoint SMS and Voice API.

**PutKeyword**

Use the PutKeyword API action to create or update a keyword configuration on an origination phone number or pool. For more information, see PutKeyword in Amazon Pinpoint SMS and Voice API.

**Managing number settings**

You can use the options in the Number settings section of the SMS and voice settings page to manage settings for the dedicated short codes and long codes that you requested from AWS Support and assigned to your account. For more information, see Managing number settings in Amazon Pinpoint User Guide.

**SMS requirements for Singapore**

Amazon SNS customers are able to send SMS traffic in Singapore using a Sender ID that has been registered through the Singapore SMS Sender ID Registry (SSIR). SSIR was launched in March 2022.
Amazon Simple Notification Service Developer Guide
SMS requirements for US destinations

through the Singapore Network Information Centre (SGNIC) which is owned by Info-communications Media Development Authority (IMDA) of Singapore, and enables organizations to register their Sender ID when sending SMS to mobile phones in Singapore.

Before you can begin using a Singapore Sender ID registered through SSIR, you must complete an onboarding process through Amazon to allow-list your account for usage of your registered Sender ID prior to initiating the registration through SGNIC. Instructions for onboarding your registered Sender ID to Amazon are listed below. If you do not want to use a Singapore registered Sender ID there is no action for you to take, and you can continue sending your messages through Amazon SNS.

**Important**
To register a Sender ID with Singapore Network Information Centre (SGNIC), there are two steps that must be completed in the following order.

1. You must first work with Amazon to register your Singapore (SG) Sender ID for your account. Once this step is complete you can proceed to the next step.
2. Work with SGNIC to register your Sender ID.

**Warning**
Doing these steps out of order may result in your Sender ID being blocked by the service, or prevent your Sender ID from being preserved on the mobile device.

**Note**
You are required to submit a Sender ID registration from each individual AWS account you require to use the Sender ID.

**To register your Singapore Sender ID with Amazon SNS**

1. Complete the steps at Requesting sender IDs for SMS messaging with Amazon SNS (p. 248). In your request, provide the following required information.
   - The AWS Region that you use with Amazon SNS.
   - The company name.
   - An estimate of the number of messages that you plan to send each month.
   - A description of your use case.
   - Information about the steps that your recipients must complete to opt-in to receiving your messages.
   - Confirmation that you collect and manage opt-ins and opt-outs.
2. Complete the form provided to you by the Amazon SNS team after completing the previous step.
3. Reply back to the case and attach the completed form for processing.
4. Amazon will inform you in the support case when your Sender ID is successfully registered to your account, and provide you with the necessary routing information to initiate the registration through SGNIC.

**Special requirements for sending SMS messages to US destinations**

**Important**
US mobile carriers have recently changed their regulations, and will require that all toll-free numbers (TFNs) complete a registration process with a regulatory body before September 30, 2022. If you currently have a toll-free number you must register your toll-free number by September 30, 2022 or you will no longer be able to use the toll-free number. You can check the status of your number by going to the section called “Toll-free number registration
status” (p. 240). For more information about registering your company see the section called “Registering your toll-free number” (p. 237).

It can take up to 15 business days for your registration to be processed after it is submitted.

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. 221) for messaging US destinations:

- Short codes (p. 242)
- 10-digit long codes (10DLC) (p. 221)
- Toll-free numbers (p. 234)

### 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.

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. 309)
- Step 1: Registering with the TRAI (p. 310)
- Step 2: Requesting a sender ID (p. 310)
- Step 3: Sending SMS messages (p. 311)
- Troubleshooting SMS messages sent to recipients in India (p. 312)

### Sending SMS messages to India: task overview

To send SMS messages to India, complete the following tasks:

1. Register with the TRAI (p. 310).
2. Request a sender ID (p. 310).
3. Send SMS messages (p. 311).
Step 1: Registering with the TRAI

Before you can send SMS messages to recipients in India, you must register your organization with the Telecom Regulatory Authority of India (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.

The following is a sample list of a few Distributed Ledger Technology (DLT) registration sites you can use to register your organization with the TRAI (fees may apply). The registration process varies by site. Contact their respective support teams for assistance.

- **BSNL DLT** - Free registration.
- **Jio Trueconnect** - Charges a fee for completing the registration process.
- **Smart Enterprise Solutions** - Charges a fee for completing the registration process.
- **Vilpower** - Includes a template that you can download and modify to fit your needs. Vilpower charges a fee for completing the registration process.

To register your organization with the TRAI

The following details how to register your organization with the TRAI using Vilpower.

1. In a web browser, go to the Vilpower website at [https://www.vilpower.in](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**

   DLT registration provider websites are not maintained by Amazon Web Services. Steps on their websites are subject to change.

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. 248). 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. 310), 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. 288).
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. 260).

**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. 313)
- User notification process overview (p. 313)
- Setting up a mobile app (p. 313)
- Sending mobile push notifications (p. 321)
- Mobile app attributes (p. 324)
- Mobile app events (p. 327)
- Mobile push API actions (p. 329)
- Mobile push API errors (p. 330)
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. 313).

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. 314) 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 application (p. 314).
3. Use the returned credentials to request a device token for your mobile app and device from the push notification service. 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. 315).
5. Use the EndpointArn to publish a message to an app on a mobile device (p. 313). For more information, see Publishing to a mobile device (p. 322) 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. 314) to set up mobile applications.

Topics
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. 313).

Creating a platform application

For Amazon SNS to send notification messages to mobile endpoints, whether directly or via subscriptions to a topic, you must first create a platform application. After registering the app with AWS, the next step is to create an endpoint for the app and mobile device. Amazon SNS then uses the endpoint for sending notification messages to the app and device.

To create a platform application

1. Sign in to the Amazon SNS console.
2. In the navigation pane, choose **Mobile**, and then choose **Push notifications**.

3. In the **Platform applications** section, choose **Create platform application**.

   For a list of AWS Regions where you can create mobile applications, see Supported Regions for mobile applications (p. 338).

4. For **Application name**, 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. Names must also be 1–256 characters long.

5. For **Push notification platform**, choose the platform that the app is registered with, and then enter the appropriate credentials.

   **Note**
   
   If you’re using one of the Apple Push Notification Service (APNs) platforms, you can choose between **token or certificate-based authentication** (p. 320), then choose **Choose file** to upload the .p8 or .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. 314) and User notification process overview (p. 313).

This section describes the recommended approach for creating a platform endpoint.

**Topics**

- Create a platform endpoint (p. 315)
- Pseudo code (p. 316)
- AWS SDK example (p. 316)
- Troubleshooting (p. 318)

### 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
```

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 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = "\n" + "Usage: " + "<token>\n\n" + "Where:\n\n" + "token - The name of the FIFO topic. \n\n" + "platformApplicationArn - The ARN value of platform application. You can get this value from the AWS Management Console. \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String token = args[0];
        String platformApplicationArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        createEndpoint(snsClient, token, platformApplicationArn);
    }

    public static void createEndpoint(SnsClient snsClient, String token, String platformApplicationArn) {
        System.out.println("Creating platform endpoint with token "+ token);

        try {
            CreatePlatformEndpointRequest endpointRequest = CreatePlatformEndpointRequest.builder()
                .token(token)
                .platformApplicationArn(platformApplicationArn)
                .build();

            CreatePlatformEndpointResponse response = snsClient.createPlatformEndpoint(endpointRequest);
            System.out.println("The ARN of the endpoint is "+ response.endpointArn());
        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

For more information, see Mobile push API actions (p. 329).
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
• 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 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](https://aws.amazon.com/documentation/java/). For more information about credentials, see [How Do I Get Security Credentials?](https://aws.amazon.com/documentation/general/latest/r气味.md/credentials/) in the [AWS General Reference](https://aws.amazon.com/documentation/general/latest/r气味.md/).  

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-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](https://opencsv.org/) 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:

```
csvfilename:C:\mytokendirectory\mytokens.csv
goodfilename:C:\\mylogfiles\\goodtokens.csv
badfilename:C:\\mylogfiles\\badtokens.csv
delimiterchar:'
quotechar:
umofthreads: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 STDOUT 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-b586-13803d420071
   <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.

Apple authentication methods

You can authorize Amazon SNS to send push notifications to your iOS or macOS app by providing information that identifies you as the developer of the app. To authenticate, provide either a **key** or a **certificate** when creating a platform application, both of which you can get from your Apple Developer account.

**Token signing key**

A private signing key that Amazon SNS uses to sign Apple Push Notification Service (APNs) authentication tokens.

If you provide a signing key, Amazon SNS uses a token to authenticate with APNs for every push notification that you send. With your signing key, you can send push notifications to APNs production and sandbox environments.

Your signing key doesn't expire, and you can use the same signing key for multiple apps. For more information, see Communicate with APNs using authentication tokens in the Developer Account Help section of the Apple website.

**Certificate**

A TLS certificate that Amazon SNS uses to authenticate with APNs when you send push notifications. You obtain the certificate from your Apple Developer account.
Certificates expire after one year. When this happens, you must create a new certificate and provide it to Amazon SNS. For more information, see Establishing a Certificate-Based Connection to APNs on the Apple Developer website.

To manage APNs settings using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. Under Mobile, choose Push notifications.
3. Select the Application for which you would like to edit the APNs settings, and then choose Edit.
4. On the Edit page, for Authentication type, choose either Token or Certificate.
5. Depending on the authentication type that you choose, do one of the following:
   - If you choose Token, provide the following information from your Apple Developer account. Amazon SNS requires this information to construct authentication tokens.
     - Signing key – The authentication token signing key from your Apple Developer account, which you download as a .p8 file. Apple lets you download your signing key only once.
     - Signing key ID – The ID that's assigned to your signing key. Amazon SNS requires this information to construct authentication tokens. To find this value in your Apple Developer account, choose Certificates, IDs & Profiles, and then choose your key in the Keys section.
     - Team identifier – The ID that's assigned to your Apple Developer account team. You can find this value on the Membership page.
     - Bundle identifier – The ID that's assigned to your app. To find this value, choose Certificates, IDs & Profiles, choose App IDs in the Identifiers section, and then choose your app.
   - If you choose Certificate, provide the following information:
     - SSL certificate – The .p12 file for your TLS certificate. You can export this file from Keychain Access after you download and install your certificate from your Apple Developer account.
     - Certificate password – If you assigned a password to your certificate, specify it here.
6. When you finish, choose Save changes.

Sending mobile push notifications

This section describes how to send mobile push notifications.

Topics

- Publishing to a topic (p. 321)
- Publishing to a mobile device (p. 322)
- Publishing with platform-specific payload (p. 322)

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. 329) 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.

```json
{"GCM":"{ "notification": { "body": "Sample message for Android or iOS endpoints", "title":"TitleTest" } }"}
```

Sending silent notification messages

If you want to send the message as a silent notification, you need to set the content_available attribute to true in the JSON formatted message.

The following examples show a silent notification message for the FCM platform.

```json
{"GCM": "{ "notification": {"content_available": true }, "data": { "body": "Sample message for Android or iOS endpoints", "title":"Hello world"}"
```

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.

```json
{}
```
Sending mobile push notifications

"GCM": "{\"notification\": {\"body\": \"Sample message for Android or iOS 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. 75) 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.TOPI":
      {"DataType": "String", "StringValue": "com.amazon.mobile.messaging.myapp"},
```

323
Inferring the APNs push type header from the payload

If you don't set the `apns-push-type` APNs header, Amazon SNS sets the 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.

```json
{
  "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.
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

```json
{
"status": "SUCCESS",
"notification": {
  "timestamp": "2015-01-26 23:07:39.54",
  "messageId": "9655abe4-6ed6-5734-89f7-e6a6a42de02a"
},
"delivery": {
  "statusCode": 200,
  "dwellTimeMs": 65,
  "token": "Examplei7fFachk31xjLqT64RaBkcGhochmf1VQAz9k-IBJ3kJp7fedYPzEwT_Pq3TuO1rOqro1cwWJUvgkccPPYcaXCpPwmg3Bqy-wiqIEzp5z7Yy_jmWPXKhddCzux6paEysay9Zn3DwNU3b8mHXxBf9dqaEw",
  "attempts": 1,
  "providerResponse": "\"multicast_id\":5138139752481671853,\"success\":1,\"failure\":0,\"canonical_ids\":0,\"results\":\"0:1422313659698010\d6ba8edf9fd7ecd\"
},
```
FAILURE

{
    "status": "FAILURE",
    "notification": {
        "timestamp": "2015-01-26 23:29:35.678",
        "messageId": "c3ad79b0-8996-550a-8bfa-24f0598989f"
    },
    "delivery": {
        "statusCode": 8,
        "dwellTimeMs": 1451,
        "token": "examp1e29z6j5c4df46f80189c4c83fjcgf7f6257e98542d2jt3395kj73",
        "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. 326).

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.

```java
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);
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:
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. 327)
- Sending mobile push notifications (p. 328)

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.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Notification trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventEndpointCreated</td>
<td>A new platform endpoint is added to your application.</td>
</tr>
<tr>
<td>EventEndpointDeleted</td>
<td>Any platform endpoint associated with your application is deleted.</td>
</tr>
<tr>
<td>EventEndpointUpdated</td>
<td>Any of the attributes of the platform endpoints associated with your application are changed.</td>
</tr>
<tr>
<td>EventDeliveryFailure</td>
<td>A delivery to any of the platform endpoints associated with your application encounters a permanent failure.</td>
</tr>
</tbody>
</table>

Note

To track delivery failures on the platform application side, subscribe to message delivery status events for the application. For more information, see Using Amazon SNS Application Attributes for Message Delivery Status.
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 **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:

```bash
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",
   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. 314).

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. 313).

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. 329).

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
<th>HTTPS status code</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Name is null string</td>
<td>The required application name is set to null.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>Platform Name is null string</td>
<td>The required platform name is set to null.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>Platform Name is invalid</td>
<td>An invalid or out-of-range value was supplied for the platform name.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Principal is not a valid certificate</td>
<td>An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Principal is a valid cert but not in a .pem format</td>
<td>A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>APNs — Principal is an expired certificate</td>
<td>An expired certificate was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Principal is not an Apple issued certificate</td>
<td>A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Principal is not provided</td>
<td>The APNs principal, which is the SSL certificate, was not provided.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Credential is not provided</td>
<td>The APNs credential, which is the private key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>APNs — Credential are not in a valid .pem format</td>
<td>The APNs credential, which is the private key, is not in a valid .pem format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>FCM — serverAPIKey is not provided</td>
<td>The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>FCM — serverAPIKey is empty</td>
<td>The FCM credential, which is the API key, is empty.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>FCM — serverAPIKey is a null string</td>
<td>The FCM credential, which is the API key, is null.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>FCM — serverAPIKey is invalid</td>
<td>The FCM credential, which is the API key, is invalid.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — clientsecret is not provided</td>
<td>The required client secret is not provided.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — clientsecret is a null string</td>
<td>The required string for the client secret is null.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>ADM — client_secret is empty string</td>
<td>The required string for the client secret is empty.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — client_secret is not valid</td>
<td>The required string for the client secret is not valid.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — client_id is empty string</td>
<td>The required string for the client ID is empty.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — clientId is not provided</td>
<td>The required string for the client ID is not provided.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — clientid is a null string</td>
<td>The required string for the client ID is null.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>ADM — client_id is not valid</td>
<td>The required string for the client ID is not valid.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointCreated has invalid ARN format</td>
<td>EventEndpointCreated has invalid ARN format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointDeleted has invalid ARN format</td>
<td>EventEndpointDeleted has invalid ARN format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointUpdated has invalid ARN format</td>
<td>EventEndpointUpdated has invalid ARN format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventDeliveryAttemptFailure has invalid ARN format</td>
<td>EventDeliveryAttemptFailure has invalid ARN format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventDeliveryFailure has invalid ARN format</td>
<td>EventDeliveryFailure has invalid ARN format.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointCreated is not an existing Topic</td>
<td>EventEndpointCreated is not an existing topic.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointDeleted is not an existing Topic</td>
<td>EventEndpointDeleted is not an existing topic.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventEndpointUpdated is not an existing Topic</td>
<td>EventEndpointUpdated is not an existing topic.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventDeliveryAttemptFailure is not an existing Topic</td>
<td>EventDeliveryAttemptFailure is not an existing topic.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>EventDeliveryFailure is not an existing Topic</td>
<td>EventDeliveryFailure is not an existing topic.</td>
<td>400</td>
<td>CreatePlatformApplication</td>
</tr>
<tr>
<td>Platform ARN is invalid</td>
<td>Platform ARN is invalid.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>Platform ARN is valid but does not belong to the user</td>
<td>Platform ARN is valid but does not belong to the user.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>APNs — Principal is not a valid certificate</td>
<td>An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Principal is a valid cert but not in a .pem format</td>
<td>A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Principal is an expired certificate</td>
<td>An expired certificate was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Principal is not an Apple issued certificate</td>
<td>A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Principal is not provided</td>
<td>The APNs principal, which is the SSL certificate, was not provided.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Credential is not provided</td>
<td>The APNs credential, which is the private key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>APNs — Credential are not in a valid .pem format</td>
<td>The APNs credential, which is the private key, is not in a valid .pem format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>FCM — serverAPIKey is not provided</td>
<td>The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>FCM — serverAPIKey is a null string</td>
<td>The FCM credential, which is the API key, is null.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>ADM — clientid is not provided</td>
<td>The required string for the client ID is not provided.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>ADM — clientid is a null string</td>
<td>The required string for the client ID is null.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>ADM — clientsecret is not provided</td>
<td>The required client secret is not provided.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>ADM — clientsecret is a null string</td>
<td>The required string for the client secret is null.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointUpdated has invalid ARN format</td>
<td>EventEndpointUpdated has invalid ARN format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointDeleted has invalid ARN format</td>
<td>EventEndpointDeleted has invalid ARN format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointUpdated has invalid ARN format</td>
<td>EventEndpointUpdated has invalid ARN format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventDeliveryAttemptFailure has invalid ARN format</td>
<td>EventDeliveryAttemptFailure has invalid ARN format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventDeliveryFailure has invalid ARN format</td>
<td>EventDeliveryFailure has invalid ARN format.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointCreated is not an existing Topic</td>
<td>EventEndpointCreated is not an existing topic.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointDeleted is not an existing Topic</td>
<td>EventEndpointDeleted is not an existing topic.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventEndpointUpdated is not an existing Topic</td>
<td>EventEndpointUpdated is not an existing topic.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventDeliveryAttemptFailure is not an existing Topic</td>
<td>EventDeliveryAttemptFailure is not an existing topic.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>EventDeliveryFailure is not an existing Topic</td>
<td>EventDeliveryFailure is not an existing topic.</td>
<td>400</td>
<td>SetPlatformAttributes</td>
</tr>
<tr>
<td>Platform ARN is invalid</td>
<td>The platform ARN is invalid.</td>
<td>400</td>
<td>GetPlatformApplicationAttributes</td>
</tr>
<tr>
<td>Platform ARN is valid but does not belong to the user</td>
<td>The platform ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>GetPlatformApplicationAttributes</td>
</tr>
<tr>
<td>Token specified is invalid</td>
<td>The specified token is invalid.</td>
<td>400</td>
<td>ListPlatformApplications</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Platform ARN is invalid</td>
<td>The platform ARN is invalid.</td>
<td>400</td>
<td>ListEndpointsByPlatformApplication</td>
</tr>
<tr>
<td>Platform ARN is valid but does not belong to the user</td>
<td>The platform ARN is valid, but does not belong to the user.</td>
<td>404</td>
<td>ListEndpointsByPlatformApplication</td>
</tr>
<tr>
<td>Token specified is invalid</td>
<td>The specified token is invalid.</td>
<td>400</td>
<td>ListEndpointsByPlatformApplication</td>
</tr>
<tr>
<td>Platform ARN is invalid</td>
<td>The platform ARN is invalid.</td>
<td>400</td>
<td>DeletePlatformApplication</td>
</tr>
<tr>
<td>Platform ARN is valid but does not belong to the user</td>
<td>The platform ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>DeletePlatformApplication</td>
</tr>
<tr>
<td>Platform ARN is invalid</td>
<td>The platform ARN is invalid.</td>
<td>400</td>
<td>CreatePlatformEndpoint</td>
</tr>
<tr>
<td>Platform ARN is valid but does not belong to the user</td>
<td>The platform ARN is valid, but does not belong to the user.</td>
<td>404</td>
<td>CreatePlatformEndpoint</td>
</tr>
<tr>
<td>Token is not specified</td>
<td>The token is not specified.</td>
<td>400</td>
<td>CreatePlatformEndpoint</td>
</tr>
<tr>
<td>Token is not of correct length</td>
<td>The token is not the correct length.</td>
<td>400</td>
<td>CreatePlatformEndpoint</td>
</tr>
<tr>
<td>Customer User data is too large</td>
<td>The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.</td>
<td>400</td>
<td>CreatePlatformEndpoint</td>
</tr>
<tr>
<td>Endpoint ARN is invalid</td>
<td>The endpoint ARN is invalid.</td>
<td>400</td>
<td>DeleteEndpoint</td>
</tr>
<tr>
<td>Endpoint ARN is valid but does not belong to the user</td>
<td>The endpoint ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>DeleteEndpoint</td>
</tr>
<tr>
<td>Endpoint ARN is invalid</td>
<td>The endpoint ARN is invalid.</td>
<td>400</td>
<td>SetEndpointAttributes</td>
</tr>
<tr>
<td>Endpoint ARN is valid but does not belong to the user</td>
<td>The endpoint ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>SetEndpointAttributes</td>
</tr>
<tr>
<td>Token is not specified</td>
<td>The token is not specified.</td>
<td>400</td>
<td>SetEndpointAttributes</td>
</tr>
<tr>
<td>Token is not of correct length</td>
<td>The token is not the correct length.</td>
<td>400</td>
<td>SetEndpointAttributes</td>
</tr>
<tr>
<td>Error</td>
<td>Description</td>
<td>HTTPS status code</td>
<td>API Action</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Customer User data is too large</td>
<td>The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.</td>
<td>400</td>
<td>SetEndpointAttributes</td>
</tr>
<tr>
<td>Endpoint ARN is invalid</td>
<td>The endpoint ARN is invalid.</td>
<td>400</td>
<td>GetEndpointAttributes</td>
</tr>
<tr>
<td>Endpoint ARN is valid but does not belong to the user</td>
<td>The endpoint ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>GetEndpointAttributes</td>
</tr>
<tr>
<td>Target ARN is invalid</td>
<td>The target ARN is invalid.</td>
<td>400</td>
<td>Publish</td>
</tr>
<tr>
<td>Target ARN is valid but does not belong to the user</td>
<td>The target ARN is valid, but does not belong to the user.</td>
<td>403</td>
<td>Publish</td>
</tr>
<tr>
<td>Message format is invalid</td>
<td>The message format is invalid.</td>
<td>400</td>
<td>Publish</td>
</tr>
<tr>
<td>Message size is larger than supported by protocol/end-service</td>
<td>The message size is larger than supported by the protocol/end-service.</td>
<td>400</td>
<td>Publish</td>
</tr>
</tbody>
</table>

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. 336)
- Precedence order for determining TTL (p. 337)
- Specifying TTL using the AWS Management Console (p. 337)

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:
Mobile push TTL

<table>
<thead>
<tr>
<th>Push notification service</th>
<th>TTL message attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Device Messaging (ADM)</td>
<td>AWS.SNS.MOBILE.ADM.TTL</td>
</tr>
<tr>
<td>Apple Push Notification Service (APNs)</td>
<td>AWS.SNS.MOBILE.APNS.TTL</td>
</tr>
<tr>
<td>Apple Push Notification Service Sandbox (APNs_SANDBOX)</td>
<td>AWS.SNS.MOBILE.APNS_SANDBOX.TTL</td>
</tr>
<tr>
<td>Baidu Cloud Push (Baidu)</td>
<td>AWS.SNS.MOBILE.BAIDU.TTL</td>
</tr>
<tr>
<td>Firebase Cloud Messaging (FCM)</td>
<td>AWS.SNS.MOBILE.FCM.TTL</td>
</tr>
<tr>
<td>Windows Push Notification Services (WNS)</td>
<td>AWS.SNS.MOBILE.WNS.TTL</td>
</tr>
</tbody>
</table>

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)
- AWS GovCloud (US-West)

### Mobile push notifications best practices

This section describes best practices that might help you improve your customer engagement.

#### Endpoint management

Delivery issues might occur in situations where device tokens change due to a user's action on the device (e.g. an app is re-installed on the device), or certificate updates affecting devices running on a particular iOS version. It's a recommended best practice by Apple to register with APNs each time your app launches.

Since the device token won't change each time an app is opened by a user, the idempotent `CreatePlatformEndpoint` API can be used. However, this can introduce duplicates for the same device in cases where the token itself is invalid, or if the endpoint is valid but disabled (e.g. a mismatch of production and sandbox environments).

A device token management mechanism such as the one in the **pseudo code** (p. 316) can be used.

#### Delivery status logging

To monitor push notification delivery status, we recommended you enable delivery status logging for your Amazon SNS platform application. This helps you troubleshoot delivery failures because the logs contain provider **response codes** (p. 326) returned from the push platform service. For details on enabling delivery status logging, see How do I access Amazon SNS topic delivery logs for push notifications?.

#### Event notifications

For managing endpoints in an event-driven fashion, you can make use of the **event notifications** (p. 329) functionality. This allows the configured Amazon SNS topic to fanout events...
to the subscribers such as a Lambda function, for platform application events of endpoint creation, deletion, updates, and delivery failures.

Email notifications

This page describes how to subscribe an email address (p. 339) 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.

Important

To prevent mailing list recipients from unsubscribing all recipients from Amazon SNS topic emails, see Set up an email subscription that requires authentication to unsubscribe from AWS Support.

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. 79).
   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. 146).
   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.

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
/// <summary>
/// Creates a new subscription to a topic.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
to create an Amazon SNS subscription.</param>
/// <param name="topicArn">The ARN of the topic to subscribe to.</param>
/// <returns>A SubscribeResponse object which includes the subscription
ARN for the new subscription.</returns>
public static async Task<SubscribeResponse> TopicSubscribeAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
{
    SubscribeRequest request = new SubscribeRequest()
    {
        TopicArn = topicArn,
        ReturnSubscriptionArn = true,
        Protocol = "email",
        Endpoint = "recipient@example.com",
    };

    var response = await client.SubscribeAsync(request);
    return response;
}
```

For API details, see Subscribe in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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.
*/

```c
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;
}
```

• For API details, see Subscribe in AWS SDK for C++ API Reference.

Go

SDK for Go V2

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see Subscribe in AWS SDK for Go API Reference.
Java

SDK for Java 2.x

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```

• For API details, see **Subscribe** in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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 */,
```
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();

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see Subscribe in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun subEmail(topicArnVal: String, email: String): String {
  val request = SubscribeRequest {
    protocol = "email"
    endpoint = email
    returnSubscriptionArn = true
    topicArn = topicArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    val result = snsClient.subscribe(request)
    return result.subscriptionArn.toString()
  }
}

• For API details, see Subscribe in AWS SDK for Kotlin API reference.

PHP

SDK for PHP

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require '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:
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());
}

• For API details, see Subscribe in AWS SDK for PHP API Reference.

Python

SDK for Python (Boto3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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.
        
        def __init__(self, sns_resource):
            """
: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.

```python
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
```

- For API details, see Subscribe in AWS SDK for Python (Boto3) API Reference.

Ruby

**SDK for Ruby**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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 subscriptions was created."
    else
        puts "The subscription was not created. Stopping program."
        exit 1
    end
end
```
Amazon Simple Notiﬁcation Service Developer Guide
AWS SDKs

run_me if $PROGRAM_NAME == __FILE__

• 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.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

async fn 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(())

• For API details, see Subscribe in AWS SDK for Rust API reference.

346


Code examples for Amazon SNS using AWS SDKs

The following code examples show how to use Amazon SNS with an AWS software development kit (SDK).

*Actions* are code excerpts that show you how to call individual service functions.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

*Cross-service examples* are sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see *Using Amazon SNS with an AWS SDK* (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**Code examples**

- **Actions for Amazon SNS using AWS SDKs** (p. 348)
  - Add tags to an Amazon SNS topic using an AWS SDK (p. 348)
  - Check whether a phone number is opted out of Amazon SNS using an AWS SDK (p. 350)
  - Confirm an endpoint owner wants to receive Amazon SNS messages using an AWS SDK (p. 353)
  - Create an Amazon SNS topic using an AWS SDK (p. 355)
  - Delete an Amazon SNS subscription using an AWS SDK (p. 361)
  - Delete an Amazon SNS topic using an AWS SDK (p. 365)
  - Get the properties of an Amazon SNS topic using an AWS SDK (p. 370)
  - Get the settings for sending Amazon SNS SMS messages using an AWS SDK (p. 374)
  - List phone numbers opted out of Amazon SNS using an AWS SDK (p. 377)
  - List the subscribers of an Amazon SNS topic using an AWS SDK (p. 378)
  - List Amazon SNS topics using an AWS SDK (p. 384)
  - Publish an Amazon SNS SMS text message using an AWS SDK (p. 390)
  - Publish to an Amazon SNS topic using an AWS SDK (p. 395)
  - Set a dead-letter queue for an Amazon SNS subscription using an AWS SDK (p. 401)
  - Set an Amazon SNS filter policy using an AWS SDK (p. 402)
  - Set the default settings for sending Amazon SNS SMS messages using an AWS SDK (p. 404)
  - Set Amazon SNS topic attributes using an AWS SDK (p. 406)
  - Subscribe a Lambda function to receive notifications from an Amazon SNS topic using an AWS SDK (p. 410)
  - Subscribe a mobile application to an Amazon SNS topic using an AWS SDK (p. 413)
  - Subscribe an HTTP endpoint to an Amazon SNS topic using an AWS SDK (p. 415)
  - Subscribe an email address to an Amazon SNS topic using an AWS SDK (p. 416)
- **Scenarios for Amazon SNS using AWS SDKs** (p. 423)
  - Create a platform endpoint for Amazon SNS push notifications using an AWS SDK (p. 424)
  - Create and publish to a FIFO Amazon SNS topic using an AWS SDK (p. 425)
• Publish SMS messages to an Amazon SNS topic using an AWS SDK (p. 427)
• Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK (p. 429)
• Cross-service examples for Amazon SNS using AWS SDKs (p. 431)
  • Build an application to submit data to a DynamoDB table (p. 431)
  • Build a publish and subscription application that translates messages (p. 432)
  • Create an Amazon Textract explorer application (p. 433)
  • Detect people and objects in a video with Amazon Rekognition using an AWS SDK (p. 434)
• Use API Gateway to invoke a Lambda function (p. 435)
• Use scheduled events to invoke a Lambda function (p. 436)

Actions for Amazon SNS using AWS SDKs

The following code examples demonstrate how to perform individual Amazon SNS actions with AWS SDKs. These excerpts call the Amazon SNS API and are not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Simple Notification Service API Reference.

Examples
• Add tags to an Amazon SNS topic using an AWS SDK (p. 348)
• Check whether a phone number is opted out of Amazon SNS using an AWS SDK (p. 350)
• Confirm an endpoint owner wants to receive Amazon SNS messages using an AWS SDK (p. 353)
• Create an Amazon SNS topic using an AWS SDK (p. 355)
• Delete an Amazon SNS subscription using an AWS SDK (p. 361)
• Delete an Amazon SNS topic using an AWS SDK (p. 365)
• Get the properties of an Amazon SNS topic using an AWS SDK (p. 370)
• Get the settings for sending Amazon SNS SMS messages using an AWS SDK (p. 374)
• List phone numbers opted out of Amazon SNS using an AWS SDK (p. 377)
• List the subscribers of an Amazon SNS topic using an AWS SDK (p. 378)
• List Amazon SNS topics using an AWS SDK (p. 384)
• Publish an Amazon SNS SMS text message using an AWS SDK (p. 390)
• Publish to an Amazon SNS topic using an AWS SDK (p. 395)
• Set a dead-letter queue for an Amazon SNS subscription using an AWS SDK (p. 401)
• Set an Amazon SNS filter policy using an AWS SDK (p. 402)
• Set the default settings for sending Amazon SNS SMS messages using an AWS SDK (p. 404)
• Set Amazon SNS topic attributes using an AWS SDK (p. 406)
• Subscribe a Lambda function to receive notifications from an Amazon SNS topic using an AWS SDK (p. 410)
• Subscribe a mobile application to an Amazon SNS topic using an AWS SDK (p. 413)
• Subscribe an HTTP endpoint to an Amazon SNS topic using an AWS SDK (p. 415)
• Subscribe an email address to an Amazon SNS topic using an AWS SDK (p. 416)

Add tags to an Amazon SNS topic using an AWS SDK

The following code examples show how to add tags to an Amazon SNS topic.
Add tags to a topic

Java

**SDK for Java 2.x**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```

- For API details, see [TagResource](https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/services/sns.html#sns-examples) in *AWS SDK for Java 2.x API Reference*.

Kotlin

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun addTopicTags(topicArn: String) {
    val tag = Tag {
        key = "Team"
        value = "Development"
    }

    val tag2 = Tag {
```
Check whether a phone number is opted out

```kotlin
key = "Environment"
value = "Gamma"
}
val tagList = mutableListOf<Tag>()
tagList.add(tag)
tagList.add(tag2)
val request = TagResourceRequest {
    resourceArn = topicArn
tags = tagList
}
SnsClient { region = "us-east-1" }.use { snsClient ->
snsClient.tagResource(request)
println("Tags have been added to $topicArn")
}
```

- For API details, see TagResource in AWS SDK for Kotlin API reference.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

## 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

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;
/// <summary>
/// This example shows how to use the Amazon Simple Notification Service
/// (Amazon SNS) to check whether a phone number has been opted out. The
/// example was created using the AWS SDK for .NET version 3.7 and
/// .NET Core 5.0.
/// </summary>
public class IsPhoneNumOptedOut {
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";
        IAmazonSimpleNotificationService client = new
        AmazonSimpleNotificationServiceClient();
```
await CheckIfOptedOutAsync(client, phoneNumber);
}

/// <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 CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
{
    var request = new CheckIfPhoneNumberIsOptedOutRequest
    {
        PhoneNumber = phoneNumber,
    }
    try
    {
        var response = await
          client.CheckIfPhoneNumberIsOptedOutAsync(request);

        {
            string optOutStatus = response.IsOptedOut ? "opted out" : "not
            opted out.";
            Console.WriteLine($"The phone number: {phoneNumber} is
            {optOutStatus}"");
        }
    }
    catch (AuthorizationErrorException ex)
    {
        Console.WriteLine($"{ex.Message}"");
    }
}

• For API details, see CheckIfPhoneNumberIsOptedOut in AWS SDK for .NET API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the
AWS Code Examples Repository.

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());
    }
Check whether a phone number is opted out

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

- For API details, see `CheckIfPhoneNumberIsOptedOut` in *AWS SDK for Java 2.x API Reference*.

### JavaScript

**SDK for JavaScript V3**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

- 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**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```php
```

- For more information, see *AWS SDK for PHP Developer Guide*.
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());
}

• 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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**

!important Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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: 

" +
        result.subscriptionArn());
}

• For API details, see ConfirmSubscription in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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);
    }
};
```
Create a topic

run();

- 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**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 * 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->confirmSubscription([ 'Token' => $subscription_token,
        'TopicArn' => $topic,
    ]);  
    var_dump($result); 
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see ConfirmSubscription in AWS SDK for PHP API Reference.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**Create an Amazon SNS topic using an AWS SDK**

The following code examples show how to create an Amazon SNS topic.
**AWS SDK for .NET**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use Amazon Simple Notification Service
/// (Amazon SNS) to add a new Amazon SNS topic. The example was created
/// using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";

        IAmazonSimpleNotificationService client = new
        AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}");
    }

    /// <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;
    }
}
```

- For API details, see CreateTopic in AWS SDK for .NET API Reference.

**C++**

**SDK for C++**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
Create a topic

```java
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 null;
}
```

- For API details, see `CreateTopic` in *AWS SDK for C++ API Reference*.

**Go**

**SDK for Go V2**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

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

**Java**

**SDK for Java 2.x**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```java
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 null;
}
```
Create a topic

Java

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
suspend fun createSNSTopic(topicName: String): String {
    val request = CreateTopicRequest {
        name = topicName
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}

- For API details, see CreateTopic in AWS SDK for Kotlin API reference.

**SDK for PHP**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require '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: 
 */

$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());
}
```

- 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)**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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

- For API details, see CreateTopic in AWS SDK for Python (Boto3) API Reference.

Ruby

**SDK for Ruby**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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}'"
```

360
if topic_created?(sns_client, topic_name)
  puts "The topic was created."
else
  puts "The topic was not created. Stopping program."
  exit 1
end

run_me if $PROGRAM_NAME == __FILE__

• 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.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

async fn make_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().unwrap_or_default()
    );
    Ok((})
}

• For API details, see CreateTopic in AWS SDK for Rust API reference.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

Delete an Amazon SNS subscription using an AWS SDK

The following code examples show how to delete an Amazon SNS subscription.

.NET

AWS SDK for .NET

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
/// <summary>
/// Given the ARN for an Amazon SNS subscription, this method deletes
/// the subscription.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
/// to delete an Amazon SNS subscription.</param>
/// <param name="subscriptionArn">The ARN of the subscription to delete.</param>
public static async Task TopicUnsubscribeAsync(
    IAmazonSimpleNotificationService client,
    string subscriptionArn)
{
    var response = await client.UnsubscribeAsync(subscriptionArn);
}

- For API details, see Unsubscribe in AWS SDK for .NET API Reference.

C++

SDK for C++

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);

- For API details, see Unsubscribe in AWS SDK for C++ API Reference.
```
Delete a subscription

Java

**SDK for Java 2.x**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws[AWS Code Examples Repository](https://github.com/aws).

```java
public static void unSub(SnsClient snsClient, String subscriptionArn) {
    try {
        UnsubscribeRequest request = UnsubscribeRequest.builder()
            .subscriptionArn(subscriptionArn)
            .build();

        UnsubscribeResponse result = snsClient.unsubscribe(request);
        System.out.println("Status was " +
            result.sdkHttpResponse().statusCode() +
            "\Subscription was removed for " + request.subscriptionArn());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

* For API details, see [Unsubscribe](https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/services/sns/api/unsubscribe.html) in *AWS SDK for Java 2.x API Reference*.

JavaScript

**SDK for JavaScript V3**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws[AWS Code Examples Repository](https://github.com/aws).

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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);
    } catch (e) {
        console.error(e);
    }
}
```
For more information, see AWS SDK for JavaScript Developer Guide.

For API details, see Unsubscribe in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun unSub(subscriptionArnVal: String) {
    val request = UnsubscribeRequest {
        subscriptionArn = subscriptionArnVal
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for ${request.subscriptionArn}"")
    }
}

For API details, see Unsubscribe in AWS SDK for Kotlin API reference.

PHP

SDK for PHP

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require '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:
guide_credentials.html
 */

$SnsSclient = new SnsClient([}
Delete a topic

```php
'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());
}
```

- For more information, see [AWS SDK for PHP Developer Guide](http://aws.amazon.com/sdk-for-php/)
- For API details, see [Unsubscribe in AWS SDK for PHP API Reference](http://aws.amazon.com/get-Started With AWS SNS with an AWS SDK (p. 6)]. This topic also includes information about getting started and details about previous SDK versions.

### 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.
Delete a topic

**.NET**

**AWS SDK for .NET**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
class DeleteSNSTopic
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new AmazonSimpleNotificationServiceClient();
        var response = await client.DeleteTopicAsync(topicArn);
    }
}
```

- For API details, see `DeleteTopic` in *AWS SDK for .NET API Reference*.

**C++**

**SDK for C++**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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;
    }
}
```
Delete a topic

For API details, see `DeleteTopic` in `AWS SDK for C++ API Reference`.

Java

**SDK for Java 2.x**

*Note*
There’s more on GitHub. Find the complete example and learn how to set up and run in the `AWS Code Examples Repository`.

```java
public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
  try {
    DeleteTopicRequest request = DeleteTopicRequest.builder()
      .topicArn(topicArn)
      .build();

    DeleteTopicResponse result = snsClient.deleteTopic(request);
    System.out.println("Status was " +
      result.sdkHttpResponse().statusCode());
  } catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
  }
}
```

For API details, see `DeleteTopic` in `AWS SDK for Java 2.x API Reference`.

JavaScript

**SDK for JavaScript V3**

*Note*
There’s more on GitHub. Find the complete example and learn how to set up and run in the `AWS Code Examples Repository`.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteTopic in AWS SDK for JavaScript API Reference.

**Kotlin**

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun deleteSNSTopic(topicArnVal: String) {
  val request = DeleteTopicRequest {
    topicArn = topicArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    snsClient.deleteTopic(request)
    println("$topicArnVal was successfully deleted.")
  }
}

- For API details, see DeleteTopic in AWS SDK for Kotlin API reference.

**PHP**

**SDK for PHP**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
/**
```
Delete a topic

* Deletes a SNS topic and all its subscriptions.
* This code expects that you have AWS credentials set up per:
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());
}

• For API details, see DeleteTopic in AWS SDK for PHP API Reference.

Python

SDK for Python (Boto3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

class 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

• 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, see Using Amazon SNS with an
AWS SDK (p. 6). This topic also includes information about getting started and details about previous
SDK versions.
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**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <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()
    {
        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)
```
Get the properties of a topic

```csharp
foreach (KeyValuePair<string, string> entry in topicAttributes)
{
    Console.WriteLine($"{entry.Key}: {entry.Value}\n";
}
```

- For API details, see `GetTopicAttributes` in *AWS SDK for .NET API Reference*.

### C++

**SDK for C++**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```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);
```

- For API details, see `GetTopicAttributes` in *AWS SDK for C++ API Reference*.

### Java

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the *AWS Code Examples Repository*.

```java
```
public static void getSNSTopicAttributes(SnsClient snsClient, String topicArn )
{
    try {
        GetTopicAttributesRequest request = GetTopicAttributesRequest.builder()
            .topicArn(topicArn)
            .build();

        GetTopicAttributesResponse result = snsClient.getTopicAttributes(request);
        System.out.println("Status is "+result.sdkHttpResponse().statusCode()+"Attributes: "+result.attributes());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

• For API details, see GetTopicAttributes in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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 "./lib/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();
Get the properties of a topic

• 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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```javascript
// 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);
    });
```

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see GetTopicAttributes in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun getSNSTopicAttributes(topicArnVal: String) {
    val request = GetTopicAttributesRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.getTopicAttributes(request)
        println("${result.attributes}"))
    }
}
```
For API details, see `GetTopicAttributes` in AWS SDK for Kotlin API reference.

**PHP**

**SDK for PHP**

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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());
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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++**

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    Aws::SNS::SNSService 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_out);

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);

• For API details, see GetSMSAttributes in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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");

}
JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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);
  }
};
rune()
```

• 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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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());
}

• 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

    public static void 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) {
            
    }
List the subscribers of a topic

```java
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <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();
        }
    }
}
List the subscribers of a topic

• For API details, see ListSubscriptions in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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);
```

• For API details, see ListSubscriptions in AWS SDK for C++ API Reference.

Go

SDK for Go V2

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see ListSubscriptions in AWS SDK for Go API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
public static void 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);
    }
}

For API details, see ListSubscriptions in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see ListSubscriptions in AWS SDK for JavaScript API Reference.

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun listSNSSubscriptions() {
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.listSubscriptions(ListSubscriptionsRequest {})
        response.subscriptions?.forEach { sub ->
            println("Sub ARN is ${sub.subscriptionArn}")
            println("Sub protocol is ${sub.protocol}")
        }
    }
}
```

• For API details, see ListSubscriptions in AWS SDK for Kotlin API reference.

**SDK for PHP**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 */

$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());
}

- For API details, see ListSubscriptions in AWS SDK for PHP API Reference.

**Python**

**SDK for Python (Boto3)**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```python
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
```

- For API details, see ListSubscriptions in AWS SDK for Python (Boto3) API Reference.

**Ruby**

**SDK for Ruby**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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__

• 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

List Amazon SNS topics using an AWS SDK

The following code examples show how to list Amazon SNS topics.

.NET

AWS SDK for .NET

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
public class ListSNSTopics
{
    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 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<Topic> topicList)
{
    foreach (var topic in topicList)
    {
        Console.WriteLine($"{topic.TopicArn}");
    }
}

• For API details, see ListTopics in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Aws::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);

• For API details, see ListTopics in AWS SDK for C++ API Reference.

Go

SDK for Go V2

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see ListTopics in AWS SDK for Go API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

public static void listSNSTopics(SnsClient snsClient) {
    try {
        ListTopicsRequest request = ListTopicsRequest.builder().build();

        ListTopicsResponse result = snsClient.listTopics(request);
        System.out.println("Status was " + result.sdkHttpResponse().statusCode() + "\nTopics\n" + result.topics());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

• For API details, see ListTopics in AWS SDK for Java 2.x API Reference.
JavaScript

SDK for JavaScript V3

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListTopics in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun listSNSTopics() {
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.listTopics(ListTopicsRequest { })
        response.topics?.forEach { topic ->
            println("The topic ARN is \
```
• For API details, see ListTopics in AWS SDK for Kotlin API reference.

PHP

SDK for PHP

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 */

$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());
}
```

• For API details, see ListTopics in AWS SDK for PHP API Reference.

Python

SDK for Python (Boto3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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.
        """
```
List topics

:returns: An iterator that yields the topics.

```python
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
```

- For API details, see ListTopics in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK for Ruby

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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__
```

- 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.
Amazon Simple Notiﬁcation Service Developer Guide
Publish an SMS text message

Note

There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

async fn show_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().unwrap_or_default());
}
}

Ok(())

• For API details, see ListTopics in AWS SDK for Rust API reference.
For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an
AWS SDK (p. 6). This topic also includes information about getting started and details about previous
SDK versions.

Publish an Amazon SNS SMS text message using an
AWS SDK
The following code examples show how to publish SMS messages using Amazon SNS.
.NET
AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

using
using
using
using
using

Amazon;
Amazon.SimpleNotificationService;
Amazon.SimpleNotificationService.Model;
System;
System.Threading.Tasks;

namespace SNSMessageExample
{
class SNSMessage
{
private AmazonSimpleNotificationServiceClient snsClient;
/// <summary>
/// Constructs a new SNSMessage object initializing the Amazon Simple
/// Notification Service (Amazon SNS) client using the supplied
/// Region endpoint.
/// </summary>
/// <param name="regionEndpoint">The Amazon Region endpoint to use in
/// sending test messages with this object.</param>
public SNSMessage(RegionEndpoint regionEndpoint)
{

390


snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);

/// <summary>
/// Sends the SMS message passed in the text parameter to the phone number
/// in phoneNum.
/// </summary>
/// <param name="phoneNum">The ten-digit phone number to which the text
/// message will be sent.</param>
/// <param name="text">The text of the message to send.</param>
/// <returns></returns>
public async Task SendTextMessageAsync(string phoneNum, string text)
{
    if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
    {
        return;
    }

    // Now actually send the message.
    var request = new PublishRequest
    {
        Message = text,
        PhoneNumber = phoneNum
    };

    try
    {
        var response = await snsClient.PublishAsync(request);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error sending message: {0}\n", ex);
    }
}

• For API details, see Publish in AWS SDK for .NET API Reference.

C++

SDK for C++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

/**
 * 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/
Publish an SMS text message

* <phone_number_value> input parameter uses E.164 format.
* For example, in United States, this input value should be of the form: +12223334444
*/

```c
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;
}
```

• For API details, see Publish in AWS SDK for C++ API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public static void pubTextSMS(SnsClient snsClient, String message, String phoneNumber) {
  try {
    PublishRequest request = PublishRequest.builder()
      .message(message)
      .phoneNumber(phoneNumber)
```
Publish an SMS text message

```java
.build();

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

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

• For API details, see Publish in AWS SDK for Java 2.x API Reference.

Kotlin

**SDK for Kotlin**

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```kotlin
suspend fun pubTextSMS(messageVal: String?, phoneNumberVal: String?) {
    val request = PublishRequest {
        message = messageVal
        phoneNumber = phoneNumberVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

• For API details, see Publish in AWS SDK for Kotlin API reference.

PHP

**SDK for PHP**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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.
 */
```
Publish an SMS text message

* This code expects that you have AWS credentials set up per:
* /

```php
$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());  
}
```

- For more information, see [AWS SDK for PHP Developer Guide](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/).
- For API details, see [Publish](https://docs.aws.amazon.com/sdk-for-php/v3/api-reference/)

**Python**

**SDK for Python (Boto3)**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-sns-text-message).

```python
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, 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']
```
Publish to a topic

```java
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
```

- 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

### 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**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

- For API details, see Publish in AWS SDK for .NET API Reference.

#### C++

**SDK for C++**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```cpp
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;
    }
}```
Amazon Simple Notification Service Developer Guide
Publish to a topic

::ShutdownAPI(options);

• For API details, see Publish in AWS SDK for C++ API Reference.

Go

SDK for Go V2

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see Publish in AWS SDK for Go API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
    }
}
```

• For API details, see Publish in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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();

• For more information, see AWS SDK for JavaScript Developer Guide.
• For API details, see Publish in AWS SDK for JavaScript API Reference.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

suspend fun pubTopic(topicArnVal: String, messageVal: String) {
  val request = PublishRequest {
    message = messageVal
    topicArn = topicArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    val result = snsClient.publish(request)
    println("${result.messageId} message sent.")
  }
}

• For API details, see Publish in AWS SDK for Kotlin API reference.
The `publish` method of the `SnsClient` class can be used to send a message to an Amazon SNS topic.

```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:
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.&#39;;
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

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

- For more information, see [AWS SDK for PHP Developer Guide](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/).

### Python

SDK for Python (Boto3)

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-samples).

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

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

```python
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.

```python
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

• For API details, see Publish in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK for Ruby

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require "aws-sdk-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__

• For more information, see AWS SDK for Ruby Developer Guide.
• For API details, see Publish in AWS SDK for Ruby API Reference.
Set a dead-letter queue for a subscription

**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.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```rust
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(())
}
```

- For API details, see **Publish** in AWS SDK for Rust API reference.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

### 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.
Set a filter policy

Java

SDK for Java 1.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
"arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-bc89-012d-3e45-6fg7h8901231";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =

// 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);
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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);
}
}

• For API details, see SetSubscriptionAttributes in AWS SDK for Java 2.x API Reference.

Python

SDK for Python (Boto3)

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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("Could'n add filter to subscription %s. ", subscription.arn)
            raise

• For API details, see SetSubscriptionAttributes in AWS SDK for Python (Boto3) API Reference.
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++**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

How to use Amazon SNS to set the DefaultSMSType attribute.

```cpp
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);
```

- For API details, see [SetSmsAttributes](#) in [AWS SDK for C++ API Reference](#).

Java

**SDK for Java 2.x**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public class SetSMSAttributes {
  public static void main(String[] args) {
```
Set the default settings for sending SMS messages

```
HashMap<String, String> attributes = new HashMap<>(1);
attributes.put("DefaultSMSType", "Transactional");
attributes.put("UsageReportS3Bucket", "janbucket");

SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .credentialsProvider(ProfileCredentialsProvider.create())
    .build();
setSNSAttributes(snsClient, attributes);
snsClient.close();
```

```
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);
    }
```

- For API details, see SetSmsAttributes in AWS SDK for Java 2.x API Reference.

**JavaScript**

**SDK for JavaScript V3**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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 */,
```

405
Set topic attributes

```javascript
  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();
```

- 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**

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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());
}
```

- 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
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("Status was " + result.sdkHttpResponse().statusCode() + "\nTopic " + request.topicArn() + " updated " + request.attributeName() + " to " + request.attributeValue());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see SetTopicAttributes in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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 = {
```
Set topic attributes

```javascript
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();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see SetTopicAttributes in AWS SDK for JavaScript API Reference.

Kotlin

```kotlin
suspend fun setTopAttr(attribute: String?, topicArnVal: String?, value: String?) {
    val request = SetTopicAttributesRequest {
        attributeName = attribute
        attributeValue = value
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.setTopicAttributes(request)
        println("Topic \${request.topicArn} was updated."")
    }
}
```

- For API details, see SetTopicAttributes in AWS SDK for Kotlin API reference.

PHP

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
```

- For API details, see SetTopicAttributes in AWS SDK for PHP for API reference.
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:
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());
}

• For API details, see SetTopicAttributes in AWS SDK for PHP API Reference.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

require "aws-sdk-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.
Subscribe a Lambda function to a topic

```ruby
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
})
```

- For more information, see [AWS SDK for Ruby Developer Guide](https://docs.aws.amazon.com/sdk-for-ruby/latest/api/Aws/SNS/Resource.html#set_attributes-class-Aws-SNS-Resource.html).

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an [AWS SDK](https://docs.aws.amazon.com/sdk-for-ruby/latest/api/Aws/SNS/Resource.html) (p. 6). This topic also includes information about getting started and details about previous SDK versions.

### 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++**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-sdk-cpp).

```cpp
/**
 * 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;
```
Subscribe a Lambda function to a topic

```cpp
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;
```

- For API details, see [Subscribe](https://docs.aws.amazon.com/sdk-for-cpp/v1/api/latest/reference/class Aws::SNS::Model::SubscribeRequest.html) in [AWS SDK for C++ API Reference](https://docs.aws.amazon.com/sdk-for-cpp/v1/api/latest/reference/class Aws::SNS::Model::SubscribeRequest.html).

### Java

#### SDK for Java 2.x

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-java-examples).

```java
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 "";
}
```
Subscribe a Lambda function to a topic

- For API details, see Subscribe in AWS SDK for Java 2.x API Reference.

**JavaScript**

SDK for JavaScript V3

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see Subscribe in AWS SDK for JavaScript API Reference.

**Kotlin**

SDK for Kotlin

**Note**
This is prerelease documentation for a feature in preview release. It is subject to change.

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
Subscribe a 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++

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.
Subscribe a mobile application to a topic

```
" <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;
}
```

- For API details, see **Subscribe** in *AWS SDK for C++ API Reference*.

**JavaScript**

**SDK for JavaScript V3**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws/aws-sdk-js-v3).

Create the client in a separate module and export it.

```javascript
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.**

```javascript
// 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 = {
```
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

```java
public static void subHTTPS(SnsClient snsClient, String topicArn, String url )
{
    try {
        SubscribeRequest request = SubscribeRequest.builder()
                .protocol("https")
                .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);
    }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.
Subscribe an email address to a topic

- For API details, see Subscribe in AWS SDK for Java 2.x API Reference.

PHP

SDK for PHP

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```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:
 */

$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());
}
```

- For API details, see Subscribe in AWS SDK for PHP API Reference.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

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.
subscribe an email address to a topic

**.net**

**aws sdk for .net**

**note**

there's more on github. find the complete example and learn how to set up and run in the [aws code examples repository](https://github.com/aws-samples/aws-sdk-dotnet).

```csharp
/// <summary>
/// creates a new subscription to a topic.
/// </summary>
/// <param name="client">the initialized amazon sns client object, used
to create an amazon sns subscription.</param>
/// <param name="topicArn">the arn of the topic to subscribe to.</param>
/// <returns>a subscriberesponse object which includes the subscription
arn for the new subscription.</returns>
public static async task<subscriberesponse> topicSubscribeAsync(
    iamazon simplesubscription service client,
    string topicArn)
{
    subscribeRequest request = new subscribeRequest()
    {
        topicArn = topicArn,
        returnSubscriptionArn = true,
        protocol = "email",
        endpoint = "recipient@example.com",
    };

    var response = await client.subscribeAsync(request);
    return response;
}
```


**c++**

**sdk for c++**

**note**

there's more on github. find the complete example and learn how to set up and run in the [aws code examples repository](https://github.com/aws-samples/aws-sdk-cpp).

```cpp
/**
 * 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;
}

• For API details, see Subscribe in AWS SDK for C++ API Reference.

Go

SDK for Go V2

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

• For API details, see Subscribe in AWS SDK for Go API Reference.

Java

SDK for Java 2.x

Note
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

    public static void 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);
}

- For API details, see Subscribe in AWS SDK for Java 2.x API Reference.

JavaScript

SDK for JavaScript V3

Note
There’s more on GitHub. Find the complete example and learn how to set up and run in
the AWS Code Examples Repository.

Create the client in a separate module and export it.

```javascript
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.

```javascript
// 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);
    }
```
Subscribe an email address to a topic

```kotlin
suspend fun subEmail(topicArnVal: String, email: String): String {
    val request = SubscribeRequest {
        protocol = "email"
        endpoint = email
        returnSubscriptionArn = true
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        return result.subscriptionArn.toString()
    }
}
```

• For more information, see [AWS SDK for JavaScript Developer Guide](#).
• For API details, see Subscribe in [AWS SDK for JavaScript API Reference](#).

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```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:
 */
```
$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());
}

• For API details, see Subscribe in AWS SDK for PHP API Reference.

Python

**SDK for Python (Boto3)**

**Note**
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

class 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

- For API details, see Subscribe in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK for Ruby

**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require "aws-sdk-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 subscriptions was created."
    else
        puts "The subscription was not created. Stopping program."
        exit 1
    end
end
run_me if $PROGRAM_NAME == __FILE__
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see Subscribe in AWS SDK for Ruby API Reference.
Scenarios for Amazon SNS using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon SNS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon SNS. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

• Create a platform endpoint for Amazon SNS push notifications using an AWS SDK (p. 424)
• Create and publish to a FIFO Amazon SNS topic using an AWS SDK (p. 425)
• Publish SMS messages to an Amazon SNS topic using an AWS SDK (p. 427)
• Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK (p. 429)

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 2.x**

*Note*
There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```java
public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = "\n" + "Usage: " + "<token>\n\n" + "Where:\n" + "  token - The name of the FIFO topic. \n\n" + "  platformApplicationArn - The ARN value of platform application. You can get this value from the AWS Management Console. \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String token = args[0];
        String platformApplicationArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        createEndpoint(snsClient, token, platformApplicationArn);
    }

    public static void createEndpoint(SnsClient snsClient, String token, String platformApplicationArn) {
        System.out.println("Creating platform endpoint with token "+ token);
        try {
            CreatePlatformEndpointRequest endpointRequest =
                CreatePlatformEndpointRequest.builder()
                .token(token)
                .platformApplicationArn(platformApplicationArn)
                .build();

            CreatePlatformEndpointResponse response =
                snsClient.createPlatformEndpoint(endpointRequest);
            System.out.println("The ARN of the endpoint is "+ response.endpointArn());
        }
    }
}
```
For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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 2.x**

**Note**

There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

```java
public static void main(String[] args) {
    final String usage = "\n" +
    "Usage: " +
    "    <topicArn>\n\n" +
    "Where:\n" +
    "    fifoTopicName - The name of the FIFO topic. \n\n" +
    "    fifoQueueARN - The ARN value of a SQS FIFO queue. You can get this value from the AWS Management Console. \n\n";

    if (args.length != 2) {
        System.out.println(usage);
        System.exit(1);
    }

    String fifoTopicName = "PriceUpdatesTopic3.fifo";
    String fifoQueueARN = "arn:aws:sqs:us-east-1:814548047983:MyPriceSQS.fifo";
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

    createFIFO(snsClient, fifoTopicName, fifoQueueARN);
}

public static void createFIFO(SnsClient snsClient, String topicName, String queueARN) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = new HashMap<>();
        topicAttributes.put("FifoTopic", "true");
        topicAttributes.put("ContentBasedDeduplication", "false");
```
CreateTopicRequest topicRequest = CreateTopicRequest.builder()
    .name(topicName)
    .attributes(topicAttributes)
    .build();

CreateTopicResponse response = snsClient.createTopic(topicRequest);
String topicArn = response.topicArn();
System.out.println("The topic ARN is" + topicArn);

// Subscribe to the endpoint by using the SNS service client.
// Only Amazon SQS FIFO queues can receive notifications from an Amazon
// SNS FIFO topic.
SubscribeRequest subscribeRequest = SubscribeRequest.builder()
    .topicArn(topicArn)
    .endpoint(queueARN)
    .protocol("sqs")
    .build();
snsClient.subscribe(subscribeRequest);
System.out.println("The topic is subscribed to the queue.");

// Compose and publish a message that updates the wholesale price.
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";

MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
    .dataType("String")
    .stringValue(attributeValue)
    .build();

Map<String, MessageAttributeValue> attributes = new HashMap<>();
attributes.put(attributeName, msgAttValue);
PublishRequest pubRequest = PublishRequest.builder()
    .topicArn(topicArn)
    .subject(subject)
    .message(payload)
    .messageGroupId(groupId)
    .messageDeduplicationId(dedupId)
    .messageAttributes(attributes)
    .build();
snsClient.publish(pubRequest);
System.out.println("Message was published to " + topicArn);
}
} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an
AWS SDK (p. 6). This topic also includes information about getting started and details about previous
SDK versions.
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 2.x**

*Note*

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/aws-samples/aws-code-examples).

Create a topic and return its ARN.

```java
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 "";
}
```

Subscribe an endpoint to a topic.

```java
public static void subTextSNS(SnsClient snsClient, String topicArn, String phoneNumber) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("sms")
            .endpoint(phoneNumber)
            .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);
    }
```
Publish SMS messages to a topic

Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```java
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    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);
        }
    }

    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("Message sent. Status was " + result.sdkHttpResponse().statusCode());
        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Publish a message to a topic. The message is sent to every subscriber.
For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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**

*Note* There’s more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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.

```java
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.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();
```

---

429
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());
} 

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.
Cross-service examples for Amazon SNS using AWS SDKs

The following sample applications use AWS SDKs to combine Amazon SNS with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples
- Build an application to submit data to a DynamoDB table (p. 431)
- Build a publish and subscription application that translates messages (p. 432)
- Create an Amazon Textract explorer application (p. 433)
- Detect people and objects in a video with Amazon Rekognition using an AWS SDK (p. 434)
- Use API Gateway to invoke a Lambda function (p. 435)
- Use scheduled events to invoke a Lambda function (p. 436)

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

Kotlin

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

Shows how to create a native Android application that submits data using the Amazon DynamoDB Kotlin API and sends a text message using the Amazon SNS Kotlin 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

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**Build a publish and subscription application that translates messages**

The following code examples show how to create an application that has subscription and publish functionality and translates messages.

**.NET**

**AWS SDK for .NET**

Shows how to use the Amazon Simple Notification Service .NET 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.

**Services used in this example**

• Amazon SNS
• Amazon Translate

**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

Kotlin

**SDK for Kotlin**

*Note*
This is prerelease documentation for a feature in preview release. It is subject to change.

Shows how to use the Amazon SNS Kotlin API to create an application that has subscription and publish functionality. In addition, this example application also translates messages.

For complete source code and instructions on how to create a web app, see the full example on GitHub.

For complete source code and instructions on how to create a native Android app, 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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
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

---

Show 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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.
• Amazon SNS
• Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

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.

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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.

**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.

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](https://github.com).

**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](https://github.com).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](https://docs.aws.amazon.com/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, see Using Amazon SNS with an AWS SDK (p. 6). This topic also includes information about getting started and details about previous SDK versions.
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. 437)
- Identity and access management in Amazon SNS (p. 459)
- Logging and monitoring in Amazon SNS (p. 482)
- Compliance validation for Amazon SNS (p. 491)
- Resilience in Amazon SNS (p. 491)
- Infrastructure security in Amazon SNS (p. 491)
- Amazon SNS security best practices (p. 492)

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](https://aws.amazon.com/amazon-sns/data-protection/). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR blog post](https://aws.amazon.com/security/post/2018/06/18/aws-shared-responsibility-model-and-gdpr/) 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](https://aws.amazon.com/amazon-sns/data-protection/).
- **Message data protection**
  - Message data protection is a new major feature of Amazon SNS
  - Use MDP to scan message for confidential or sensitive information
  - Provide message auditing to all content flowing through the topic
  - Provide content access controls to messages published to the topic and messages delivered by the topic
Important
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 Amazon Web 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. 438)
• Internetwork traffic privacy (p. 447)
• Message Data Protection security (p. 459)

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. 438)
• Key management (p. 440)
• Enabling server-side encryption (SSE) for an Amazon SNS topic (p. 444)
• Enabling server-side encryption (SSE) for an Amazon SNS topic with an encrypted Amazon SQS queue subscribed (p. 445)

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. 444). 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. 439) 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 the AWS KMS key (p. 439) 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. 439)
- Key terms (p. 439)

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.

AWS KMS key ID

The alias, alias ARN, key ID, or key ARN of an AWS KMS key, or a custom AWS KMS—in your account or in another account. While the alias of the AWS managed AWS KMS for Amazon SNS is always alias/aws/sns, the alias of a custom AWS KMS can, for example, be alias/MyAlias. You can use these AWS KMS keys 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 KMS for Amazon SNS for a topic, AWS KMS creates the AWS managed KMS for Amazon SNS.
• Alternatively, the first time you use the Publish action on a topic with SSE enabled, AWS KMS creates the AWS managed KMS for Amazon SNS.

You can create AWS KMS keys, define the policies that control how AWS KMS keys can be used, and audit AWS KMS usage using the AWS KMS keys section of the AWS KMS console or the CreateKey AWS KMS action. For more information, see AWS KMS keys and Creating Keys in the AWS Key Management Service Developer Guide. For more examples of AWS KMS identifiers, see KeyId in the AWS Key Management Service API Reference. For information about finding AWS KMS 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. 440) 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. 440)
• Configuring AWS KMS permissions (p. 441)
• AWS KMS errors (p. 443)

Estimating AWS KMS costs

To predict costs and better understand your AWS bill, you might want to know how often Amazon SNS uses your AWS KMS key.

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:

\[
R = \frac{B}{D} \times (2 \times P)
\]

\(B\) 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.
Data encryption

- There is 1 publishing principal.

\[
\frac{2,678,400}{300} \times (2 \times 1) = 17,856
\]

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.

\[
\frac{2,419,200}{300} \times (2 \times 3) + \frac{2,419,200}{300} \times (2 \times 5) = 129,024
\]

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 KMSs in specific regions in the Resource section of an IAM policy.

You must also ensure that the key policies of the AWS KMS key allow the necessary permissions. To do this, name the principals that produce and consume encrypted messages in Amazon SNS as users in the KMS key policy.

Alternatively, you can specify the required AWS KMS actions and KMS 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 in the AWS Key Management Service Developer Guide.

Allow a user to send messages to a topic with SSE

The publisher must have the \texttt{kms:GenerateDataKey} and \texttt{kms:Decrypt} permissions for the AWS KMS key.

```json
[
  "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"
    ]
  }]
```
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 KMS. 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 KMS policy.

   ```json
   {
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "service.amazonaws.com"
         },
         "Action": [
           "kms:GenerateDataKey*",
           "kms:Decrypt"
         ],
         "Resource": "*
       }]
   }
   ```

### Event source | Service principal
--- | ---
Amazon CloudWatch | cloudwatch.amazonaws.com
Amazon CloudWatch Events | events.amazonaws.com
AWS CodeCommit | codecommit.amazonaws.com
AWS CodeStar | codestar-notifications.amazonaws.com
AWS Database Migration Service | dms.amazonaws.com
AWS Directory Service | ds.amazonaws.com
Amazon DynamoDB | dynamodb.amazonaws.com
Amazon Inspector | inspector.amazonaws.com
Amazon Redshift | redshift.amazonaws.com
Amazon RDS | events.rds.amazonaws.com
Amazon S3 Glacier | glacier.amazonaws.com
Amazon Simple Email Service | ses.amazonaws.com
Amazon Simple Storage Service | s3.amazonaws.com
AWS Snowball | importexport.amazonaws.com
AWS Systems Manager Incident Manager | AWS Systems Manager Incident Manager consists of two service principles:
<table>
<thead>
<tr>
<th>Event source</th>
<th>Service principal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ssm-incidents.amazonaws.com; ssm-contacts.amazonaws.com</td>
</tr>
</tbody>
</table>

**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
- EC2 Image Builder

3. Add the `aws:SourceAccount` and `aws:SourceArn` condition keys to the KMS resource policy to further protect the KMS key from confused deputy attacks. Refer to service specific documentation list (above) for exact details in each case.

```json
{
    "Effect": "Allow",
    "Principal": {
        "Service": "service.amazonaws.com"
    },
    "Action": [
        "kms:GenerateDataKey*",
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "customer-account-id"
        },
        "ArnLike": {
        }
    }
}
```

4. **Enable SSE for your topic** (p. 444) using your KMS.
5. 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 KMS 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 AWS KMS 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. 438).

**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 AWS KMS key. For more information, see Key terms (p. 439).

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

   **Important**

   If you aren't the owner of the KMS, 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 KMS on the Amazon SNS console.

   Ask the owner of the KMS 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 KMS for Amazon SNS (Default) alias/aws/sns is selected by default.
Data encryption

Note
Keep the following in mind:

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

Note
For instructions on creating custom KMSs, see Creating Keys in the AWS Key Management Service Developer Guide

- To use a custom KMS ARN from your AWS account or from another AWS account, enter it into the AWS KMS key 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. 438).

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.

```json
{
    "Sid": "Allow Amazon SNS to use this key",
    "Effect": "Allow",
}```
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. 439).
      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.
   c. For Customer master key (CMK), choose MyCustomCMK which you created earlier (p. 445) 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. 445), 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. 447) 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. 448)
• Creating an Amazon VPC endpoint policy for Amazon SNS (p. 449)
• Publishing an Amazon SNS message from Amazon VPC (p. 450)

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. 450).

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:

```bash
$ 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:

```json
{
  "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`.

```json
{
  "Statement": [
    {
      "Action": ["sns:Publish"],
      "Effect": "Allow",
      "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

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:
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.
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.

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.

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.

7. For Subnets, choose the subnet that has VPCE-Tutorial-Subnet in the subnet ID.
8. For **Enable Private DNS Name**, select **Enable for this endpoint**.

9. For **Security group**, choose **Select security group**, and choose **VPCE-Tutorial-SecurityGroup**.

10. 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**.
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-d169-5be6-9042-410dfe9e86de"
   }
   ```

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/.
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

2. In the navigation menu on the left, choose Logs.
3. Check the logs that were written by the Lambda functions:
   b. Choose the log stream.
   c. Check that the log includes the entry From SNS: Hello.
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

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

Message Data Protection security

- Message Data Protection (p. 91) is a feature in Amazon SNS used to define your own rules and policies to audit and control the content for data in motion, as opposed to data at rest.
- Message Data Protection provides governance, compliance, and auditing services for enterprise applications that are message-centric, so data ingress and egress can be controlled by the Amazon SNS topic owner, and content flows can be tracked and logged.
- You can write payload-based governance rules to stop unauthorized payload content from entering your message streams.
- You can grant different content-access permissions to individual subscribers, and audit the entire content-flow process.

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. 459)
- Access control (p. 461)
- Overview of managing access in Amazon SNS (p. 461)
- Using identity-based policies with Amazon SNS (p. 475)
- Using temporary security credentials with Amazon SNS (p. 481)
- Amazon SNS API permissions: Actions and resources reference (p. 481)

Authentication

You can access AWS as any of the following types of identities:
• **AWS account root user**

When you create an AWS account, you begin with one 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. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see *Tasks that require root user credentials* in the *AWS General Reference*.

• **IAM users and groups**

An **IAM user** is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see *Rotate access keys regularly for use cases that require long-term credentials* in the *IAM User Guide*.

An **IAM group** is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see *When to create an IAM user (instead of a role)* in the *IAM User Guide*.

• **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** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see *Creating a role for a third-party Identity Provider* in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see *Permission sets* in the *AWS IAM Identity Center (successor to AWS Single Sign-On) 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. 461)
- Key concepts (p. 461)
- Architectural overview (p. 464)
- Using the Access Policy Language (p. 465)
- Evaluation logic (p. 466)
- Example cases for Amazon SNS access control (p. 470)

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. 470).
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see Limit subscriptions to HTTPS (p. 470).
- 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. 471).

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. 462)
- Statement (p. 462)
- Policy (p. 462)
- Issuer (p. 462)
- Principal (p. 463)
- Action (p. 463)
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 Topic A. 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 Topic A.

![Policy Diagram]

Only ASCII characters are allowed in policy documents. You can utilize `aws:SourceAccount` and `aws:SourceOwner` to work around the scenario where you need to plug-in other AWS services' ARNs that contain non-ASCII characters. See the difference between `aws:SourceAccount` versus `aws:SourceOwner` (p. 473).

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
Overview

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. 481).

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. 466).

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 grantn 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, grantn 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. 466).

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. 466).

**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.
Overview

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. 466).

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 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.). |
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 Evaluation logic (p. 466).

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. In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request. 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 Explicit deny (p. 464)).

4. If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. 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).
If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a default deny (for more information, see Default deny (p. 464)).

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 because the policy restricts access from Antarctica. Policy B returns an allow because the policy allows requests that come in on June 1, 2010. The request is therefore denied.
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

This section describes a few examples of typical use cases for access control.

**Topics**
- Grant AWS account access to a topic (p. 470)
- Limit subscriptions to HTTPS (p. 470)
- Publish messages to an Amazon SQS queue (p. 471)
- Allow Amazon S3 event notifications to publish to a topic (p. 472)
- Allow Amazon SES to publish to a topic that is owned by another account (p. 473)
- aws:SourceAccount versus aws:SourceOwner (p. 473)
- Allow accounts in an organization in AWS Organizations to publish to a topic in a different account (p. 474)
- Allow any CloudWatch alarm to publish to a topic in a different account (p. 474)
- Restrict publication to an Amazon SNS topic only from a specific VPC endpoint (p. 475)

### 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:

```json
{
    "Statement": [
        {
            "Sid": "grant-1234-publish",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": ["sns:Publish"],
        }
    ]
}
```

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.

```json
{
    "Statement": [{
        "Sid": "Statement1",
        "Effect": "Allow",
        "Principal": {
            "AWS": "111122223333"
        },
        "Action": ["sns:Subscribe"],
        "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.

```json
{
    "Statement": [{
        "Sid": "Allow-SNS-SendMessage",
        "Effect": "Allow",
        "Principal": {
            "Service": "sns.amazonaws.com"
        },
        "Action": ["sqs:SendMessage"],
        "Condition": {
            "ArnEquals": {
            }
        }
    }
}]
```

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.

```json
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "s3.amazonaws.com"
    },
    "Action": "sns:Publish",
    "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 S3 Glacier
- Amazon SES
- Amazon Simple Storage Service
- 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",
      "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 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. 472) 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. 473) 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",
      "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",
      "Condition": {
        "ArnLike": {
        }
      }
    }
  ]
}
```
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",
        "Condition": {
            "StringNotEquals": {
                "aws:sourceVpce": "vpce-1ab2c34d"
            }
        }
    }]
}
```

Using identity-based policies with Amazon SNS

**Topics**
- IAM and Amazon SNS policies together (p. 475)
- Amazon SNS resource ARN format (p. 478)
- Amazon SNS API actions (p. 479)
- Amazon SNS policy keys (p. 479)
- Example policies for Amazon SNS (p. 479)

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. 479).

**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. 466)). The following diagram illustrates the concept.
For examples of policies that cover Amazon SNS actions and resources, see Example policies for Amazon SNS (p. 479). 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.

\[
\text{arn:aws:sns:region:account_ID:topic_name}
\]

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.

\[
\text{arn:aws:sns:us-east-2:123456789012:MyTopic}
\]

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.

\[
\text{arn:aws:sns:*:123456789012:MyTopic}
\]

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_.

\[
\text{arn:aws:sns:*:123456789012:bob_*}
\]
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. 479)) 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. 479)) 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.

```json
{
  "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.

```json
{
}
```
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 Subscribe action, with string matching conditions for the sns:Protocol and sns:Endpoint policy keys.

```json
{
    "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.

```json
{
    "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%2FNYYxh6S%2Br3yy5K70oX4%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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Maximum quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>30 kb</td>
</tr>
<tr>
<td>Statements</td>
<td>100</td>
</tr>
<tr>
<td>Principals</td>
<td>1 to 200 (0 is invalid.)</td>
</tr>
</tbody>
</table>
**Valid Amazon SNS policy actions**

Amazon SNS supports the actions shown in the following table.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sns:AddPermission</td>
<td>Grants permission to add permissions to the topic policy.</td>
</tr>
<tr>
<td>sns:DeleteTopic</td>
<td>Grants permission to delete a topic.</td>
</tr>
<tr>
<td>sns:GetDataProtectionPolicy</td>
<td>Grants permission to retrieve a topic's data protection policy.</td>
</tr>
<tr>
<td>sns:GetTopicAttributes</td>
<td>Grants permission to receive all of the topic attributes.</td>
</tr>
<tr>
<td>sns:ListSubscriptionsByTopic</td>
<td>Grants permission to retrieve all the subscriptions to a specific topic.</td>
</tr>
<tr>
<td>sns:Publish</td>
<td>Grants permission to both publish and publish batch to a topic or endpoint.</td>
</tr>
<tr>
<td>sns:PutDataProtectionPolicy</td>
<td>Grants permission to set a topic's data protection policy.</td>
</tr>
<tr>
<td>sns:RemovePermission</td>
<td>Grants permission to remove any permissions in the topic policy.</td>
</tr>
<tr>
<td>sns:SetTopicAttributes</td>
<td>Grants permission to set a topic's attributes.</td>
</tr>
<tr>
<td>sns:Subscribe</td>
<td>Grants permission to subscribe to a topic.</td>
</tr>
</tbody>
</table>

**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. 479)) 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. 479)) 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.
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
• GetSMAttributes
• GetSMSSandboxAccountStatus
• GetSubscriptionAttributes
• GetTopicAttributes
• ListEndpointsByPlatformApplication
• ListOriginationNumbers
• ListPhoneNumbersOptedOut
• ListPlatformApplications
• ListSMSSandboxPhoneNumbers
• ListSubscriptions
• ListSubscriptionsByTopic
• ListTagsForResource
• ListTopics
• OptInPhoneNumber
• RemovePermission
• SetEndpointAttributes
• SetPlatformApplicationAttributes
• SetSMAttributes
• 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.

```json
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "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",
        "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"
    }
  ]
}
```
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. 487). For more information about Amazon SNS push notifications, see Mobile push notifications (p. 312).

**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
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](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Alarms.html).

### Amazon SNS metrics

Amazon SNS sends the following metrics to CloudWatch.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NumberOfMessagesPublished</strong></td>
<td>The number of messages published to your Amazon SNS topics.</td>
</tr>
<tr>
<td></td>
<td><strong>Units</strong>: <strong>Count</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Valid Dimensions</strong>: Application, PhoneNumber, Platform, and TopicName</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Statistics</strong>: Sum</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NumberOfNotificationsDelivered</td>
<td>The number of messages successfully delivered from your Amazon SNS topics to subscribing endpoints. 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. Units: Count Valid Dimensions: Application, PhoneNumber, Platform, and TopicName Valid Statistics: Sum</td>
</tr>
<tr>
<td>NumberOfNotificationsFailed</td>
<td>The number of messages that Amazon SNS failed to deliver. 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). This metric does not include messages that were rejected by subscription filter policies. You can control the number of retries for HTTP endpoints. For more information, see Amazon SNS message delivery retries (p. 141). Units: Count Valid Dimensions: Application, PhoneNumber, Platform, and TopicName Valid Statistics: Sum, Average</td>
</tr>
<tr>
<td>NumberOfNotificationsFilteredOut</td>
<td>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. Units: Count Valid Dimensions: Application, PhoneNumber, Platform, and TopicName Valid Statistics: Sum, Average</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NumberOfNotificationsFilteredOut-InvalidAttributes</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NumberOfNotificationsFilteredOut-NoMessageAttributes</strong></td>
<td>The number of messages that were rejected by subscription filter policies because the messages have no attributes.</td>
</tr>
<tr>
<td><strong>NumberOfNotificationsRedrivenToDlq</strong></td>
<td>The number of messages that have been moved to a dead-letter queue.</td>
</tr>
<tr>
<td><strong>NumberOfNotificationsFailedToRedriveToDlq</strong></td>
<td>The number of messages that couldn't be moved to a dead-letter queue.</td>
</tr>
<tr>
<td><strong>PublishSize</strong></td>
<td>The size of messages published.</td>
</tr>
</tbody>
</table>
### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
</table>
| SMSMonthToDateSpentUSD                    | The charges you have accrued since the start of the current calendar month for sending SMS messages.  
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.

For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see Setting SMS messaging preferences (p. 252).  

Units: USD  
Valid Dimensions: PhoneNumber  
Valid Statistics: Maximum |

| SMSSuccessRate                            | The rate of successful SMS message deliveries.  
Units: Count  
Valid Dimensions: PhoneNumber  
Valid Statistics: Sum, Average, Data Samples |

### Dimensions for Amazon SNS metrics

Amazon Simple Notification Service sends the following dimensions to CloudWatch.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Filters on application objects, which represent an app and device registered with one of the supported push notification services, such as APNs and FCM.</td>
</tr>
<tr>
<td>Application,Platform</td>
<td>Filters on application and platform objects, where the platform objects are for the supported push notification services, such as APNs and FCM.</td>
</tr>
<tr>
<td>Country</td>
<td>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.</td>
</tr>
<tr>
<td>PhoneNumber</td>
<td>Filters on the phone number when you publish SMS directly to a phone number (without a topic).</td>
</tr>
<tr>
<td>Platform</td>
<td>Filters on platform objects for the push notification services, such as APNs and FCM.</td>
</tr>
<tr>
<td>TopicName</td>
<td>Filters on Amazon SNS topic names.</td>
</tr>
<tr>
<td>SMSType</td>
<td>Filters on the message type of SMS message. Can be promotional or transactional.</td>
</tr>
</tbody>
</table>
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. 475).

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. 492)
- Implement least-privilege access (p. 492)
- Use IAM roles for applications and AWS services which require Amazon SNS access (p. 493)
- Implement server-side encryption (p. 493)
- Enforce encryption of data in transit (p. 493)
- Consider using VPC endpoints to access Amazon SNS (p. 494)
- Ensure subscriptions are not configured to deliver to raw http endpoints (p. 494)

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. 459)
- Amazon SNS API permissions: Actions and resources reference (p. 481)

### 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. 438) and Key management (p. 440).

### 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:

```json
{
  "Id": "ExamplePolicy",
  "Version": "2012-10-17",
  "Statement": [
```
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. 448) and Creating an Amazon VPC endpoint policy for Amazon SNS (p. 449).

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*.

Service features are sometimes rolled out incrementally to the AWS Regions where a service is available. We update this documentation for the first release only. We don't provide information about Region availability or announce subsequent Region rollouts. For information about Region availability of service features and to subscribe to notifications about updates, see [What's New with AWS?](#).

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA256 hash algorithm added for Amazon SNS message signing</td>
<td>Support added for SHA256 hash algorithm when using Amazon SNS message signing.</td>
<td>September 15, 2022</td>
</tr>
<tr>
<td>Additional regions added to SMS messaging</td>
<td>Amazon SNS supports SMS messaging in the following regions: Africa (Cape Town), Asia Pacific (Osaka), Europe (Milan) and AWS GovCloud (US-East).</td>
<td>September 9, 2022</td>
</tr>
<tr>
<td>Message data protection support added</td>
<td>Message data protection safeguards the data that's published to your Amazon SNS topics by using data protection policies to audit and block the sensitive information that moves between applications or AWS services.</td>
<td>September 8, 2022</td>
</tr>
<tr>
<td>New registration process for toll-free numbers</td>
<td>Support added for sending Amazon SNS messages using toll-free phone numbers (TFN) to United States recipients.</td>
<td>August 1, 2022</td>
</tr>
<tr>
<td>Support for Attribute-based access controls (ABAC)</td>
<td>Added support for attribute-based access control (ABAC) for API actions including Publish and PublishBatch. ABAC is an authorization strategy that defines access permissions based on tags which can be attached to IAM resources, such as IAM users and roles, and to AWS resources, like Amazon SNS topics, to simplify permission management.</td>
<td>January 10, 2022</td>
</tr>
<tr>
<td>Support for Apple token-based authentication for push notifications</td>
<td>You can authorize Amazon SNS to send push notifications to your iOS or macOS app by providing information that identifies you as the developer of the app.</td>
<td>October 28, 2021</td>
</tr>
<tr>
<td><strong>New senders of SMS messages are placed in the SMS sandbox</strong></td>
<td>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.</td>
<td>June 1, 2021</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>New senders of SMS messages are placed in the SMS sandbox</strong></td>
<td>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.</td>
<td>June 1, 2021</td>
</tr>
<tr>
<td><strong>New attributes for sending SMS messages to recipients in India</strong></td>
<td>Two new attributes, <strong>Entity ID</strong> and <strong>Template ID</strong>, are now required for sending SMS messages to recipients in India.</td>
<td>April 22, 2021</td>
</tr>
<tr>
<td><strong>Updates to message filtering operators</strong></td>
<td>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-</code>but operator for attribute string matching.</td>
<td>April 7, 2021</td>
</tr>
<tr>
<td><strong>Ending support for P2P long codes for US destinations</strong></td>
<td>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 <strong>10DLC</strong>.</td>
<td>February 16, 2021</td>
</tr>
<tr>
<td><strong>Support for 1-minute Amazon CloudWatch metrics</strong></td>
<td>The 1-minute CloudWatch metric for Amazon SNS is now available in all AWS Regions.</td>
<td>January 28, 2021</td>
</tr>
<tr>
<td><strong>Support for Amazon Kinesis Data Firehose endpoints</strong></td>
<td>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.</td>
<td>January 12, 2021</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Origination numbers are available</td>
<td>You can use origination numbers when sending text messages (SMS).</td>
<td>October 23, 2020</td>
</tr>
<tr>
<td>Support for Amazon SNS FIFO topics</td>
<td>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.</td>
<td>October 22, 2020</td>
</tr>
<tr>
<td>The Amazon SNS Extended Client Library for Java is available</td>
<td>You can use this library to publish large Amazon SNS messages.</td>
<td>August 25, 2020</td>
</tr>
<tr>
<td>SSE is available in the China Regions</td>
<td>Server-side encryption (SSE) for Amazon SNS is available in the China Regions.</td>
<td>January 20, 2020</td>
</tr>
<tr>
<td>Support for using DLQs to capture undeliverable messages</td>
<td>To capture undeliverable messages, you can use an Amazon SQS dead-letter queue (DLQ) with an Amazon SNS subscription.</td>
<td>November 14, 2019</td>
</tr>
<tr>
<td>Support for specifying custom APNs header values</td>
<td>You can specify a custom APNs header value.</td>
<td>October 18, 2019</td>
</tr>
<tr>
<td>Support for the 'apns-push-type' header field for APNs</td>
<td>You can use the apns-push-type header field for mobile notifications sent through APNs.</td>
<td>September 10, 2019</td>
</tr>
<tr>
<td>Support for topic troubleshooting using AWS X-Ray</td>
<td>You can use X-Ray to troubleshoot messages passing through SNS topics.</td>
<td>July 24, 2019</td>
</tr>
<tr>
<td>Support for attribute key matching using the 'exists' operator</td>
<td>To check whether an incoming message has an attribute whose key is listed in the filter policy, you can use the exists operator.</td>
<td>July 5, 2019</td>
</tr>
<tr>
<td>Support for anything-but matching of multiple numeric values</td>
<td>In addition to multiple strings, Amazon SNS allows anything-but matching of multiple numeric values.</td>
<td>July 5, 2019</td>
</tr>
<tr>
<td>Amazon SNS release notes are available as an RSS feed</td>
<td>Following the title on this page (Documentation history), choose RSS.</td>
<td>June 22, 2019</td>
</tr>
</tbody>
</table>
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.