
Amazon Simple Notification Service

Developer Guide



Amazon Simple Notification Service: Developer Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

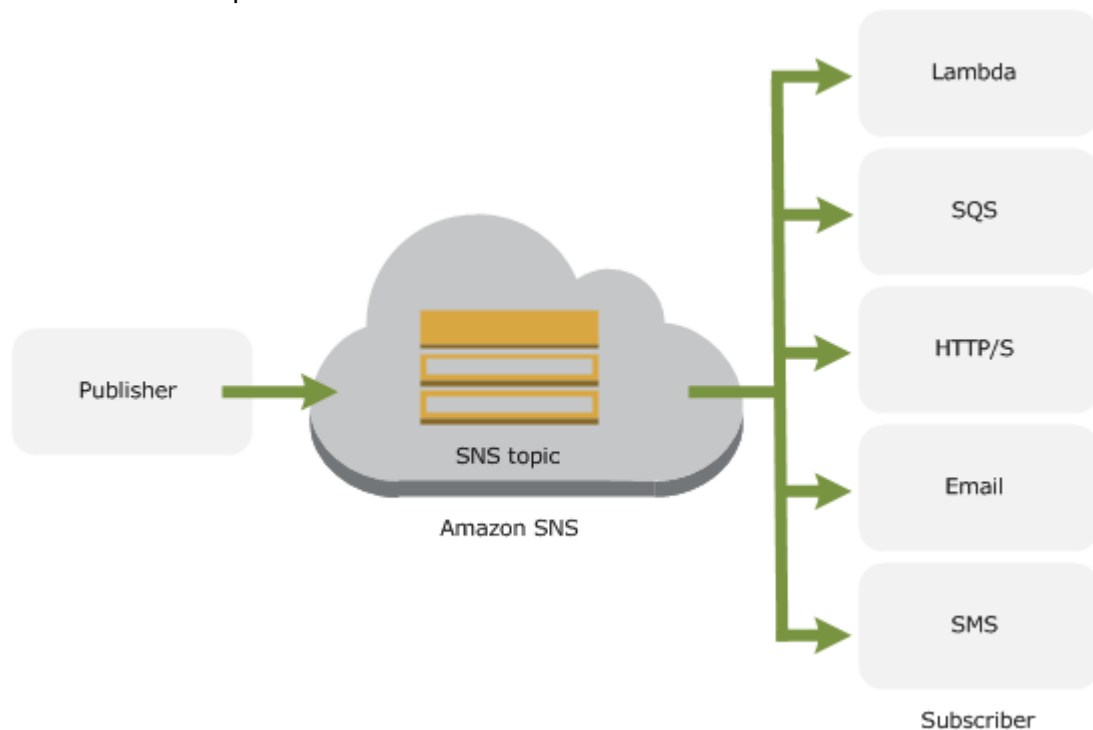
What is Amazon SNS?	1
Setting Up Access	2
Step 1: Create an AWS Account and an IAM Administrator User	2
Step 2: Create an IAM User and Get Your AWS Credentials	2
Next Steps	3
Getting Started	4
Prerequisites	4
Step 1: Create a Topic	4
Step 2: Create a Subscription for an Endpoint to the Topic	4
Step 3: Publish a Message to the Topic	5
Step 4: Delete the Subscription and Topic	6
Next Steps	7
Tutorials	8
Basic Workflows	8
Creating a Topic	8
Subscribing an Endpoint to a Topic	11
Publishing a Message to a Topic	13
Listing, Adding, and Removing Tags for a Topic	16
Configuring a Dead-Letter Queue	17
Deleting a Subscription and Topic	20
Publishing Workflows	22
Publishing a Message with Attributes	22
Working with Amazon SNS Securely	27
Enabling SSE for a Topic	27
Enabling SSE for a Topic with an Encrypted Queue Subscribed	30
Tutorial: Publishing Messages Privately from a VPC	32
Working with AWS Event Fork Pipelines	41
Deploying and Testing AWS Event Fork Pipelines	41
Subscribing AWS Event Fork Pipelines to a Topic	48
How Amazon SNS Works	54
Common Scenarios	54
Fanout	54
Application and System Alerts	55
Push Email and Text Messaging	55
Mobile Push Notifications	55
Message Durability	55
Message Delivery Status	55
Configuring Delivery Status Logging Using the AWS Management Console	56
Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS	
Endpoints Using the AWS SDKs	56
Message Delivery Retries	58
Delivery Protocols and Policies	58
Delivery Policy Stages	59
Creating a Delivery Policy	60
Dead-Letter Queues	62
Why Do Message Deliveries Fail?	62
How Do Dead-Letter Queues Work?	63
How Are Messages Moved into a Dead-Letter Queue?	63
How Can I Move Messages out of a Dead-Letter Queue?	64
How Can I Monitor and Log Dead-Letter Queues?	64
Message Attributes	64
Message Attribute Items and Validation	65
Data Types	65
Reserved Message Attributes for Mobile Push Notifications	65

Message Filtering	67
Subscription Filter Policies	67
Tutorial: Applying a Subscription Filter Policy	73
Tutorial: Removing a Subscription Filter Policy	76
Subscription Filter Policies as Java Collections	77
Message and JSON Formats	80
HTTP/HTTPS Headers	81
HTTP/HTTPS Subscription Confirmation JSON Format	81
HTTP/HTTPS Notification JSON Format	83
HTTP/HTTPS Unsubscribe Confirmation JSON Format	84
SetSubscriptionAttributes Delivery Policy JSON Format	85
SetTopicAttributes Delivery Policy JSON Format	86
Large Payload and Raw Message Delivery	87
Enabling Raw Message Delivery Using the AWS Management Console	87
Tags	87
System-to-System Messaging	89
With AWS Lambda Function as Subscriber	89
Prerequisites	89
Configuring Amazon SNS with Lambda Endpoints using the AWS Management Console	89
With Amazon SQS Queue as Subscriber	90
Step 1: Get the ARN of the Queue and Topic	91
Step 2: Give Permission to the Amazon SNS Topic to Send Messages to the Amazon SQS Queue ..	91
Step 3: Subscribe the Queue to the Amazon SNS Topic	92
Step 4: Give Users Permissions to the Appropriate Topic and Queue Actions	93
Step 5: Test the Topic's Queue Subscriptions	95
Sending Messages to a Queue in a Different Account	95
Using an AWS CloudFormation Template to Create a Topic that Sends Messages to Amazon	
SQS Queues	98
With HTTP/S Endpoint as Subscriber	103
Step 1: Make Sure Your Endpoint is Ready to Process Amazon SNS Messages	104
Step 2: Subscribe the HTTP/HTTPS Endpoint to the Amazon SNS Topic	107
Step 3: Confirm the subscription	107
Step 4: Set the delivery retry policy for the subscription (optional)	107
Step 5: Give users permissions to publish to the topic (optional)	108
Step 6: Send messages to the HTTP/HTTPS endpoint	109
Verifying Message Signatures	109
Example Code for an Endpoint Java Servlet	111
With AWS Event Fork Pipelines as Subscriber	114
How AWS Event Fork Pipelines Works	115
Deploying AWS Event Fork Pipelines	117
User Notifications	119
With Mobile Application as Subscriber (Mobile Push)	119
How User Notifications Work	120
Prerequisites for Amazon SNS User Notifications	120
User Notification Process Overview	121
Using Amazon SNS Mobile Push	121
Application Attributes for Message Delivery Status	133
Application Event Notifications	136
Amazon SNS TTL	138
Amazon SNS Mobile Push APIs	140
API Errors	142
With Mobile Phone Number as Subscriber (Send SMS)	148
Setting Preferences	148
Sending a Message	151
Sending a Message to Multiple Phone Numbers	155
Monitoring SMS Activity	161
Managing Subscriptions	166

Reserving a Short Code	170
Supported Regions and Countries	171
Troubleshooting	179
Troubleshooting Topics Using X-Ray	179
Security	180
Data Protection	180
Data Encryption	180
Internetwork Traffic Privacy	185
Identity and Access Management	188
Authentication	188
Access Control	189
Overview	189
Using Identity-Based Policies	202
Using Temporary Credentials	208
API Permissions Reference	208
Logging and Monitoring	209
Logging API Calls Using CloudTrail	210
Monitoring Topics Using CloudWatch	213
Compliance Validation	216
Resilience	217
Infrastructure Security	217
Best Practices	218
Preventative Best Practices	218
Release Notes	220
Document History	223
AWS Glossary	234

What is Amazon Simple Notification Service?

Amazon Simple Notification Service (Amazon SNS) is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. In Amazon SNS, there are two types of clients—publishers and subscribers—also referred to as producers and consumers. Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel. Subscribers (that is, web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (that is, Amazon SQS, HTTP/S, email, SMS, Lambda) when they are subscribed to the topic.



When using Amazon SNS, you (as the owner) create a topic and control access to it by defining policies that determine which publishers and subscribers can communicate with the topic. A publisher sends messages to topics that they have created or to topics they have permission to publish to. Instead of including a specific destination address in each message, a publisher sends a message to the topic. Amazon SNS matches the topic to a list of subscribers who have subscribed to that topic, and delivers the message to each of those subscribers. Each topic has a unique name that identifies the Amazon SNS endpoint for publishers to post messages and subscribers to register for notifications. Subscribers receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same messages.

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. 2)
- [Step 2: Create an IAM User and Get Your AWS Credentials](#) (p. 2)
- [Next Steps](#) (p. 3)

Step 1: Create an AWS Account and an IAM Administrator User

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

1. Navigate to the [AWS home page](#), and then choose **Create an AWS Account**.
2. Follow the instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

3. When you finish creating your AWS account, follow the instructions in the *IAM User Guide* to [create your first IAM administrator user and group](#).

Step 2: Create an IAM User and Get Your AWS Credentials

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

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

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

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

AKIAIOSFODNN7EXAMPLE

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

wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY

Important

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

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

Next Steps

Now that you're prepared to work with Amazon SNS, [get started \(p. 4\)](#) 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 AWS Management Console.

Prerequisites

Before you begin, complete the steps in [Setting Up Access for Amazon SNS \(p. 2\)](#).

Step 1: Create a Topic

1. Sign in to the [Amazon SNS console](#).
2. In the **Create topic** section, enter a **Topic name**, for example *MyTopic*.
3. 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.

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

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

Step 2: Create a Subscription for an Endpoint to the Topic

1. On the navigation panel, choose **Subscriptions**.
2. On the **Subscriptions** page, choose **Create subscription**.
3. On the **Create subscription** page, do the following:
 - a. Enter the **Topic ARN** of the topic you created earlier, for example:

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

Note

To see a list of the topics in the current AWS account, choose the **Topic ARN** field.

- b. For **Protocol**, choose an endpoint type, for example **Email**.
- c. For **Endpoint**, enter an email address that can receive notifications, for example:

```
name@example.com
```

Note

After your subscription is created, you must confirm it. Only HTTP/S endpoints, email addresses, and AWS resources in other AWS accounts require confirmation. (Amazon SQS queues and Lambda functions in the same AWS account—as well as mobile endpoints—don't require confirmation.)

- d. Choose **Create subscription**.

The subscription is created and the **Subscription: 1234a567-bc89-012d-3e45-6fg7h890123i** page is displayed.

The subscription's **ARN**, **Endpoint**, **Topic**, **Status** (Pending confirmation at this stage), and **Protocol** are displayed in the **Details** section.

4. In your email client, check the email address that you specified and choose **Confirm subscription** in the email from Amazon SNS.
5. In your web browser, a subscription confirmation with your subscription ID is displayed.

Step 3: Publish a Message to the Topic

1. On the navigation panel, choose **Topics**.
2. On the **Topics** page, choose the topic you created earlier and then choose **Publish message**.
3. On the **Publish message to topic** page, do the following:
 - a. (Optional) In the **Message details** section, enter the **Subject**, for example:

Hello from Amazon SNS!

- b. In the **Message body** section, do one of the following:

- Choose **Identical payload for all delivery protocols** and then enter the message, for example:

If you receive this message, publishing a message to an Amazon SNS topic works.

- Choose **Custom payload for each delivery protocol** and then use a JSON object to define the message to send to each protocol, for example:

```
{
  "default": "Sample fallback message",
  "email": "Sample message for email endpoints",
  "sqs": "Sample message for Amazon SQS endpoints",
  "lambda": "Sample message for AWS Lambda endpoints",
  "http": "Sample message for HTTP endpoints",
  "https": "Sample message for HTTPS endpoints",
  "sms": "Sample message for SMS endpoints",
  "APNS": "{\"aps\":{\"alert\":\"Sample message for iOS endpoints\"}}",
  "APNS_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for iOS development endpoints\"}}",
  "APNS_VOIP": "{\"aps\":{\"alert\":\"Sample message for Apple VoIP endpoints\"}}",
  "APNS_VOIP_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for Apple VoIP development endpoints\"}}",
  "MACOS": "{\"aps\":{\"alert\":\"Sample message for MacOS endpoints\"}}",
  "MACOS_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for MacOS development endpoints\"}}",
  "GCM": "{\"data\":{\"message\":\"Sample message for Android endpoints\"}}",
}
```

```
"ADM": "{ \"data\": { \"message\": \"Sample message for FireOS endpoints\n\" } }\",  
\"BAIDU\": \"{ \"title\": \"Sample message title\", \"description\": \"Sample message\nfor Baidu endpoints\" }\",  
\"MPNS\": \"<?xml version='1.0' encoding='utf-8'><wp:Notification xmlns:wp=\n\"WPNotification\"><wp:Title><wp:Count>ENTER COUNT</wp:Count><wp:Title>Sample\nmessage for Windows Phone 7+ endpoints</wp:Title></wp:Title></wp:Notification>\",  
\"WNS\": \"<badge version='1' value='42'>\"  
}
```

For more information, see [Send Custom Platform-Specific Payloads to Mobile Devices \(p. 131\)](#).

- c. 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.
 - i. Select an attribute **Type**, for example **String.Array**.

Note

If the attribute type is **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`.

- ii. Enter a **Name** for the attribute, for example `customer_interests`.
 - iii. Enter a **Value** for the attribute, for example `["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 (if present) before sending the message to the subscription.

For more information, see [Amazon SNS Message Attributes \(p. 64\)](#).

- d. Choose **Publish message**.

The message is published to the topic 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.

4. In your email client, check the email address that you specified earlier and read the email from Amazon SNS.

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 and a subscription and learned how to send messages to a topic and how to delete a subscription and topic, you might want to try the following:

- [Enable server-side encryption for a topic. \(p. 27\)](#)
- [Enable server-side encryption for a topic with an encrypted Amazon SQS queue subscribed. \(p. 30\)](#)
- [Subscribe AWS Event Fork Pipelines to an Amazon SNS topic. \(p. 48\)](#)
- [Deploy and test the AWS Event Fork Pipelines sample application \(p. 41\).](#)
- [Learn about message filtering. \(p. 67\)](#)
- Learn how to interact with Amazon SNS programmatically by exploring the [AWS Developer Center](#).
- Learn about keeping an eye on costs and resources in the [Amazon SNS Troubleshooting \(p. 179\)](#) section.
- Learn about protecting your data and access to it in the [Security \(p. 180\)](#) section.

Amazon SNS Tutorials

The following tutorials help you create an Amazon SNS topic, subscribe an endpoint to it, publish a message to the topic, and then delete the subscription and topic using the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET. If you want to use the example code, you must install the [Java Standard Edition Development Kit](#) or the [.NET SDKs for Visual Studio](#) and make some configuration changes to the example code.

Note

You can write code for Amazon SNS in other programming languages. For more information, see the [AWS SDK documentation](#).

You can explore Amazon SNS without writing code with tools such as the AWS Command Line Interface (AWS CLI) or Windows PowerShell. You can find AWS CLI examples in the [Amazon SNS section](#) of the *AWS CLI Command Reference*. You can find Windows PowerShell examples in the Amazon Simple Notification Service section of the *AWS Tools for PowerShell Cmdlet Reference*.

Topics

- [Basic Workflows](#) (p. 8)
- [Publishing Workflows](#) (p. 22)
- [Working with Amazon SNS Securely](#) (p. 27)
- [Working with AWS Event Fork Pipelines](#) (p. 41)

Basic Workflows

This section contains the tutorials that encompass the basic workflows of Amazon SNS.

Topics

- [Tutorial: Creating an Amazon SNS Topic](#) (p. 8)
- [Tutorial: Subscribing an Endpoint to an Amazon SNS Topic](#) (p. 11)
- [Tutorial: Publishing a Message to an Amazon SNS Topic](#) (p. 13)
- [Tutorial: Listing, Adding, and Removing Tags for an Amazon SNS Topic](#) (p. 16)
- [Tutorial: Configuring an Amazon SNS Dead-Letter Queue for a Subscription](#) (p. 17)
- [Tutorial: Deleting an Amazon SNS Subscription and Topic](#) (p. 20)

Tutorial: 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. The following tutorial shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to create a topic.

Topics

- [To Create a Topic Using the AWS Management Console](#) (p. 9)
- [To Create a Topic Using the AWS SDK for Java](#) (p. 10)
- [To Create a Topic Using the AWS SDK for .NET](#) (p. 10)

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. In the **Create topic** section, enter a **Topic name**, for example *MyTopic*.
4. (Optional) Expand the **Encryption** section and do the following. For more information, see [Encryption at Rest \(p. 180\)](#).
 - a. Choose **Enable encryption**.
 - b. Specify the customer master key (CMK). For more information, see [Key Terms \(p. 181\)](#).

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

Important

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

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

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

Note

Keep the following in mind:

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

Note

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

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

5. (Optional) To configure access permissions for your topic, expand the **Access policy** section. For more information, see [Identity and Access Management in Amazon SNS \(p. 188\)](#).

Important

As a security precaution, Amazon SNS uses the `aws:sourceOwner` permission to limit access to the topic to the owner of the current AWS account.

6. (Optional) To configure how Amazon SNS retries failed message delivery attempts, expand the **Delivery retry policy (HTTP/S)** section. For more information, see [Message Delivery Retries \(p. 58\)](#).
7. (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 \(p. 55\)](#).

8. (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 Tags \(p. 87\)](#).
9. 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.

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

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

To Create a Topic Using the AWS SDK for Java

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

The following code excerpt creates the topic **MyTopic** and then prints the topic ARN and the `CreateTopicRequest` request ID for a previously executed successful request.

```
// Create an Amazon SNS topic.
final CreateTopicRequest createTopicRequest = new CreateTopicRequest("MyTopic");
final CreateTopicResponse createTopicResponse =
    snsClient.createTopic(createTopicRequest);

// Print the topic ARN.
System.out.println("TopicArn:" + createTopicResponse.getTopicArn());

// Print the request ID for the CreateTopicRequest action.
System.out.println("CreateTopicRequest: " +
    snsClient.getCachedResponseMetadata(createTopicRequest));
```

3. Compile and run your code.

The topic is created and the topic ARN and `CreateTopicRequest` request ID are printed, for example:

```
TopicArn: arn:aws:sns:us-east-2:123456789012:MyTopic
CreateTopicRequest: {AWS_REQUEST_ID=1234a567-bc89-012d-3e45-6fg7h890123i}
```

4. You can assign the topic ARN to a String variable to use in additional operations, for example:

```
final String topicArn = "arn:aws:sns:us-east-2:123456789012:MyTopic";
```

To Create a Topic Using the AWS SDK for .NET

1. Specify your AWS credentials. For more information, see [Configuring AWS Credentials](#) in the *AWS SDK for .NET Developer Guide*.
2. Write your code. For more information, see [Programming with the AWS SDK for .NET](#).

The following code excerpt creates the topic **MyTopic** and then prints the topic ARN and the `CreateTopicRequest` request ID.

```
// Create an Amazon SNS topic.  
CreateTopicRequest createTopicRequest = new CreateTopicRequest("MyTopic");  
CreateTopicResponse createTopicResponse = snsClient.CreateTopic(createTopicRequest);  
  
// Print the topic ARN.  
Console.WriteLine("TopicArn: " + createTopicResponse.TopicArn);  
  
// Print the request ID for the CreateTopicRequest action.  
Console.WriteLine("CreateTopicRequest: " +  
    createTopicResponse.ResponseMetadata.RequestId);
```

3. Compile and run your code.

The topic is created and the topic ARN and `CreateTopicRequest` request ID are printed, for example:

```
TopicArn: arn:aws:sns:us-east-2:123456789012:MyTopic  
CreateTopicRequest: 1234a567-bc89-012d-3e45-6fg7h890123i
```

4. You can assign the topic ARN to a `String` variable to use in additional operations, for example:

```
String topicArn = createTopicResponse.TopicArn;
```

Tutorial: Subscribing an Endpoint to an Amazon SNS Topic

To receive messages published to [a topic \(p. 8\)](#), you must *subscribe* an endpoint (such as AWS Lambda, Amazon SQS, HTTP/S, or an email address) to the topic. When you subscribe an endpoint to a topic and confirm the subscription, the endpoint begins to receive messages published to the associated topic.

The following tutorial shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to create a subscription and then subscribe an endpoint to a topic.

Topics

- [To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS Management Console \(p. 11\)](#)
- [To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS SDK for Java \(p. 12\)](#)
- [To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS SDK for .NET \(p. 13\)](#)

To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS Management Console

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, do the following:
 - a. Enter the **Topic ARN** of the topic you created earlier, for example:

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


Note

To see a list of the topics in the current AWS account, choose the **Topic ARN** field.

- b. For **Protocol**, choose an endpoint type, for example **Email**.
- c. For **Endpoint**, enter an email address that can receive notifications, for example:

name@example.com

Note

After your subscription is created, you must confirm it. Only HTTP/S endpoints, email addresses, and AWS resources in other AWS accounts require confirmation. (Amazon SQS queues and Lambda functions in the same AWS account—as well as mobile endpoints—don't require confirmation.)

- d. Choose **Create subscription**.

The subscription is created and the **Subscription: 1234a567-bc89-012d-3e45-6fg7h890123i** page is displayed.

The subscription's **ARN**, **Endpoint**, **Topic**, **Status** (Pending confirmation at this stage), and **Protocol** are displayed in the **Details** section.

5. In your email client, check the email address that you specified and choose **Confirm subscription** in the email from Amazon SNS.
6. In your web browser, a subscription confirmation with your subscription ID is displayed.

To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS SDK for Java

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

The following code excerpt creates a subscription for an email endpoint and then prints the `SubscribeRequest` request ID.

```
// Subscribe an email endpoint to an Amazon SNS topic.
final SubscribeRequest subscribeRequest = new SubscribeRequest(topicArn, "email",
    "name@example.com");
snsClient.subscribe(subscribeRequest);

// Print the request ID for the SubscribeRequest action.
System.out.println("SubscribeRequest: " +
    snsClient.getCachedResponseMetadata(subscribeRequest));
System.out.println("To confirm the subscription, check your email.");
```

3. Compile and run your code.

The subscription is created and the `SubscribeRequest` request ID is printed, for example:

SubscribeRequest: {AWS_REQUEST_ID=1234a567-bc89-012d-3e45-6fg7h890123i}
To confirm the subscription, check your email.

To Subscribe an Endpoint to an Amazon SNS Topic Using the AWS SDK for .NET

1. Specify your AWS credentials. For more information, see [Configuring AWS Credentials](#) in the *AWS SDK for .NET Developer Guide*.
2. Write your code. For more information, see [Programming with the AWS SDK for .NET](#).

The following code excerpt creates a subscription for an email endpoint and then prints the `SubscribeRequest` request ID.

```
// Subscribe an email endpoint to an Amazon SNS topic.
SubscribeRequest subscribeRequest = new SubscribeRequest(topicArn, "email",
    "name@example.com");
SubscribeResponse subscribeResponse = snsClient.Subscribe(subscribeRequest);

// Print the request ID for the SubscribeRequest action.
Console.WriteLine("SubscribeRequest: " + subscribeResponse.ResponseMetadata.RequestId);
Console.WriteLine("To confirm the subscription, check your email.");
```

3. Compile and run your code.

The subscription is created and the `SubscribeRequest` request ID is printed, for example:

```
SubscribeRequest: 1234a567-bc89-012d-3e45-6fg7h890123i
To confirm the subscription, check your email.
```

Tutorial: Publishing a Message to an Amazon SNS Topic

After you [create a topic](#) (p. 8) and [subscribe an endpoint to it](#) (p. 11), you can *publish* messages to a topic. When the message is published, Amazon SNS attempts to deliver the message to every endpoint (such as AWS Lambda, Amazon SQS, HTTP/S, or an email address) subscribed to the topic.

The following tutorial shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to publish a message to a topic.

Topics

- [To Publish a Message to an Amazon SNS Topic Using the AWS Management Console](#) (p. 13)
- [To Publish a Message to an Amazon SNS Topic Using the AWS SDK for Java](#) (p. 15)
- [To Publish a Message to an Amazon SNS Topic Using the AWS SDK for .NET](#) (p. 15)

To Publish a Message to 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 the topic you created earlier and then choose **Publish message**.
4. On the **Publish message to topic** page, do the following:
 - a. (Optional) In the **Message details** section, enter the **Subject**, for example:

Hello from Amazon SNS!

- b. In the **Message body** section, do one of the following:

- Choose **Identical payload for all delivery protocols** and then enter the message, for example:

If you receive this message, publishing a message to an Amazon SNS topic works.

- Choose **Custom payload for each delivery protocol** and then use a JSON object to define the message to send to each protocol, for example:

```
{
  "default": "Sample fallback message",
  "email": "Sample message for email endpoints",
  "sqs": "Sample message for Amazon SQS endpoints",
  "lambda": "Sample message for AWS Lambda endpoints",
  "http": "Sample message for HTTP endpoints",
  "https": "Sample message for HTTPS endpoints",
  "sms": "Sample message for SMS endpoints",
  "APNS": "{\"aps\":{\"alert\":\"Sample message for iOS endpoints\"}}",
  "APNS_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for iOS development endpoints\"}}",
  "APNS_VOIP": "{\"aps\":{\"alert\":\"Sample message for Apple VoIP endpoints\"}}",
  "APNS_VOIP_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for Apple VoIP development endpoints\"}}",
  "MACOS": "{\"aps\":{\"alert\":\"Sample message for MacOS endpoints\"}}",
  "MACOS_SANDBOX": "{\"aps\":{\"alert\":\"Sample message for MacOS development endpoints\"}}",
  "GCM": "{ \"data\": { \"message\": \"Sample message for Android endpoints\" } }",
  "ADM": "{ \"data\": { \"message\": \"Sample message for FireOS endpoints\" } }",
  "BAIDU": "{\"title\":\"Sample message title\",\"description\":\"Sample message for Baidu endpoints\"}",
  "MPNS": "<?xml version='1.0' encoding='utf-8'?><wp:Notification xmlns:wp='WPNotification'><wp:Tile><wp:Count>ENTER COUNT</wp:Count><wp:Title>Sample message for Windows Phone 7+ endpoints</wp:Title></wp:Notification>",
  "WNS": "<badge version='1' value='42'>"
}
```

For more information, see [Send Custom Platform-Specific Payloads to Mobile Devices \(p. 131\)](#).

- c. 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.

- i. Select an attribute **Type**, for example **String.Array**.

Note

If the attribute type is **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`.

- ii. Enter a **Name** for the attribute, for example `customer_interests`.
iii. Enter a **Value** for the attribute, for example `["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 (if present) before sending the message to the subscription.

For more information, see [Amazon SNS Message Attributes \(p. 64\)](#).

- d. Choose **Publish message**.

The message is published to the topic 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.

5. In your email client, check the email address that you specified earlier and read the email from Amazon SNS.

To Publish a Message to an Amazon SNS Topic Using the AWS SDK for Java

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

The following code excerpt publishes a message to a topic and then prints the `MessageId`.

```
// Publish a message to an Amazon SNS topic.
final String msg = "If you receive this message, publishing a message to an Amazon SNS
topic works.";
final PublishRequest publishRequest = new PublishRequest(topicArn, msg);
final PublishResult publishResponse = snsClient.publish(publishRequest);

// Print the MessageId of the message.
System.out.println("MessageId: " + publishResponse.getMessageId());
```

3. Compile and run your code.

The message is published and the `MessageId` is printed, for example:

```
MessageId: 1234a567-bc89-012d-3e45-6fg7h890123i
```

To Publish a Message to an Amazon SNS Topic Using the AWS SDK for .NET

1. Specify your AWS credentials. For more information, see [Configuring AWS Credentials](#) in the *AWS SDK for .NET Developer Guide*.
2. Write your code. For more information, see [Programming with the AWS SDK for .NET](#).

The following code excerpt publishes a message to a topic and then prints the `MessageId`.

```
// Publish a message to an Amazon SNS topic.
String msg = "If you receive this message, publishing a message to an Amazon SNS topic
works.";
PublishRequest publishRequest = new PublishRequest(topicArn, msg);
PublishResponse publishResponse = snsClient.Publish(publishRequest);
```

```
// Print the MessageId of the published message.  
Console.WriteLine("MessageId: " + publishResponse.MessageId);
```

3. Compile and run your code.

The message is published and the `MessageId` is printed, for example:

```
MessageId: 1234a567-bc89-012d-3e45-6fg7h890123i
```

Tutorial: Listing, Adding, and Removing Tags for an Amazon SNS Topic

You can track your Amazon SNS resources (for example, for cost allocation) by adding, removing, and listing metadata tags for Amazon SNS topics. The following tutorial shows how to add, update, and remove tags for a topic using the AWS Management Console and the AWS SDK for Java. For more information, see [Amazon SNS Tags \(p. 87\)](#).

Topics

- [To List, Add, and Remove, Metadata Tags for an Amazon SNS Topic Using the AWS Management Console \(p. 16\)](#)
- [To List, Add, and Remove Metadata Tags for an Amazon SNS Topic using the AWS SDK for Java. \(p. 16\)](#)

Note

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

To List, Add, and Remove, Metadata Tags for an Amazon SNS Topic Using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic and choose **Edit**.
4. Expand the **Tags** section.

The tags added to the topic are listed.

5. Modify topic tags:
 - To add a tag, choose **Add tag** and enter a **Key** and **Value** (optional),
 - To remove a tag, choose **Remove tag** next to a key-value pair.
6. Choose **Save changes**

To List, Add, and Remove Metadata Tags for an Amazon SNS Topic using the AWS SDK for Java.

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

3. To list the tags added to a topic, add the following code:

```
final ListTagsForResourceRequest listTagsForResourceRequest = new
    ListTagsForResourceRequest();
listTagsForResourceRequest.setResourceArn(topicArn);
final ListTagsForResourceResult listTagsForResourceResult =
    snsClient.listTagsForResource(listTagsForResourceRequest);
System.out.println(String.format("ListTagsForResource: \tTags for topic %s are %s.\n",
    topicArn, listTagsForResourceResult.getTags()));
```

4. To add tags (or update the value of tags), add the following code:

```
final Tag tagTeam = new Tag();
tagTeam.setKey("Team");
tagTeam.setValue("Development");
final Tag tagEnvironment = new Tag();
tagEnvironment.setKey("Environment");
tagEnvironment.setValue("Gamma");

final List<Tag> tagList = new ArrayList<>();
tagList.add(tagTeam);
tagList.add(tagEnvironment);

final TagResourceRequest tagResourceRequest = new TagResourceRequest();
tagResourceRequest.setResourceArn(topicArn);
tagResourceRequest.setTags(tagList);
final TagResourceResult tagResourceResult = snsClient.tagResource(tagResourceRequest);
```

5. To remove a tag from the topic using the tag's key, add the following code:

```
final UntagResourceRequest untagResourceRequest = new UntagResourceRequest();
untagResourceRequest.setResourceArn(topicArn);
final List<String> tagKeyList = new ArrayList<>();
tagKeyList.add("Team");
untagResourceRequest.setTagKeys(tagKeyList);
final UntagResourceResult untagResourceResult =
    snsClient.untagResource(untagResourceRequest);
```

6. Compile and run your code.

The existing tags are listed, two are added, and one is removed from the topic.

Tutorial: 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 \(p. 62\)](#) and [Message Delivery Retries \(p. 58\)](#).

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

Prerequisites

Before you begin any of the following tutorials, complete the following prerequisites:

1. [Create an Amazon SNS topic \(p. 8\)](#) 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 \(p. 90\)](#).
4. [Create another Amazon SQS queue](#) named `MyDeadLetterQueue`, to be used as the dead-letter queue for the Amazon SNS subscription.
5. To give Amazon SNS principal access to the Amazon SQS API action, set the following queue policy for `MyDeadLetterQueue`.

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

Topics

- [To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using the AWS Management Console \(p. 18\)](#)
- [To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using the AWS SDK for Java \(p. 19\)](#)
- [To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using the AWS CLI \(p. 19\)](#)
- [To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using AWS CloudFormation \(p. 20\)](#)

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 \(p. 17\)](#).

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

Currently, you can't use an Amazon SQS FIFO queue as a dead-letter queue for an Amazon SNS subscription.

3. Sign in to the [Amazon SNS console](#).
4. On the navigation panel, choose **Subscriptions**.
5. On the **Subscriptions** page, select an existing subscription and then choose **Edit**.
6. On the **Edit `1234a567-bc89-012d-3e45-6fg7h890123i`** page, expand the **Redrive policy (dead-letter queue)** section, and then do the following:

- a. Choose **Enabled**.
- b. Specify the ARN of an Amazon SQS queue.
7. Choose **Save changes**.

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

To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using the AWS SDK for Java

Before you begin this tutorial, make sure you complete the [prerequisites](#) (p. 17).

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

For more information about creating Amazon SQS queues, see [To Configure an Amazon SQS Queue Using the AWS SDK for Java](#) in the *Amazon Simple Queue Service Developer Guide*.

The following code excerpt uses the ARN of an Amazon SNS subscription and an Amazon SQS queue to set the `RedrivePolicy` request parameter attribute.

```
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue
\"}";

// Set the specified Amazon SQS queue as a dead-letter queue
// of the specified Amazon SNS subscription.
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("RedrivePolicy")
    .withAttributeValue(redrivePolicy);
sns.setSubscriptionAttributes(request);
```

3. Compile and run your code.

The Amazon SQS queue is set as the dead-letter queue for the specified Amazon SNS subscription.

To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using the AWS CLI

Before you begin this tutorial, make sure you complete the [prerequisites](#) (p. 17).

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

```
aws sns set-subscription-attributes \
--subscription-arn arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i
```



```
--attribute-name RedrivePolicy
--attribute-value "{\\\"deadLetterTargetArn\\\": \\\"arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue\\\"}"
```

To Configure a Dead-Letter Queue for an Amazon SNS Subscription Using AWS CloudFormation

Before you begin this tutorial, make sure you complete the [prerequisites](#) (p. 17).

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

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

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

AWS CloudFormation begins to create the `MyDeadLetterQueue` stack and displays the **CREATE_IN_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

Tutorial: Deleting an Amazon SNS Subscription and Topic

When you no longer need a subscription or topic, you must first unsubscribe from the topic before you can delete the topic.

The following tutorial shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to publish a message to a topic.

Topics

- [To Delete an Amazon SNS Subscription and Topic Using the AWS Management Console](#) (p. 21)
- [To Delete an Amazon SNS Subscription and Topic Using the AWS SDK for Java](#) (p. 21)
- [To Delete an Amazon SNS Subscription and Topic Using the AWS SDK for .NET](#) (p. 21)

To Delete an Amazon SNS Subscription and Topic Using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Subscriptions**.
3. 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.

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

The subscription is deleted.

5. On the navigation panel, choose **Topics**.
6. 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.

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

The topic is deleted.

To Delete an Amazon SNS Subscription and Topic Using the AWS SDK for Java

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

The following code excerpt deletes a topic and then prints the `DeleteTopicRequest` request ID.

Important

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

```
// Delete an Amazon SNS topic.
final DeleteTopicRequest deleteTopicRequest = new DeleteTopicRequest(topicArn);
snsClient.deleteTopic(deleteTopicRequest);

// Print the request ID for the DeleteTopicRequest action.
System.out.println("DeleteTopicRequest: " +
    snsClient.getCachedResponseMetadata(deleteTopicRequest));
```

3. Compile and run your code.

The topic is deleted and the `DeleteTopicRequest` request ID is printed, for example:

```
DeleteTopicRequest: 1234a567-bc89-012d-3e45-6fg7h890123i
```

To Delete an Amazon SNS Subscription and Topic Using the AWS SDK for .NET

1. Specify your AWS credentials. For more information, see [Configuring AWS Credentials](#) in the *AWS SDK for .NET Developer Guide*.

2. Write your code. For more information, see [Programming with the AWS SDK for .NET](#).

The following code excerpt deletes a topic and then prints the `DeleteTopicRequest` request ID.

Important

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

```
// Delete an Amazon SNS topic.
DeleteTopicRequest deleteTopicRequest = new DeleteTopicRequest(topicArn);
DeleteTopicResponse deleteTopicResponse = snsClient.DeleteTopic(deleteTopicRequest);

// Print the request ID for the DeleteTopicRequest action.
Console.WriteLine("DeleteTopicRequest: " +
    deleteTopicResponse.ResponseMetadata.RequestId);
```

3. Compile and run your code.

The topic is deleted and the `DeleteTopicRequest` request ID is printed, for example:

```
DeleteTopicRequest: 1234a567-bc89-012d-3e45-6fg7h890123i
```

Publishing Workflows

This section contains additional tutorials on the publishing workflows of Amazon SNS.

Topics

- [Tutorial: Publishing a Message with Attributes to an Amazon SNS Topic \(p. 22\)](#)

Tutorial: Publishing a Message with Attributes to an Amazon SNS Topic

After you [create a topic \(p. 8\)](#) and [subscribe an endpoint to it \(p. 11\)](#), you can *publish* messages to a topic with message attributes, which let you provide structure metadata items about the message. For more information, see [Amazon SNS Message Attributes \(p. 64\)](#).

The following tutorial shows how you can use the AWS Management Console, the AWS SDK for Java, and the AWS SDK for .NET to publish a message to a topic.

Topics

- [To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS Management Console \(p. 22\)](#)
- [To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for Java \(p. 23\)](#)
- [To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for .NET \(p. 25\)](#)

To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS Management Console

For detailed instructions on publishing a message with attributes to an Amazon SNS topic using the AWS Management Console, see [To Publish a Message to an Amazon SNS Topic Using the AWS Management Console \(p. 13\)](#).

To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for Java

1. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Write your code. For more information, see [Using the SDK for Java 2.x](#).

The following code example helps simplify the process of publishing messages with attributes. The `SNSMessageAttributes` class stores the `messageAttributes` field as a map. You can use the overloaded `addAttribute` method to add attributes with the data types `String`, `String.Array`, and `Number`. To publish the message, use the `publish` method by providing the `AmazonSNS` client and the topic ARN.

```
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.model.MessageAttributeValue;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.PublishResult;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

public class SNSMessageAttributes {

    private String message;
    private Map<String, MessageAttributeValue> messageAttributes;

    public SNSMessageAttributes(final String message) {
        this.message = message;
        messageAttributes = new HashMap<>();
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(final String message) {
        this.message = message;
    }

    public void addAttribute(final String attributeName, final String attributeValue)
    {
        final MessageAttributeValue messageAttributeValue = new
MessageAttributeValue()
            .withDataType("String")
            .withStringValue(attributeValue);
        messageAttributes.put(attributeName, messageAttributeValue);
    }

    public void addAttribute(final String attributeName, final ArrayList<?>
attributeValues) {
        String valuesString, delimiter = ", ", prefix = "[", suffix = "]";
        if (attributeValues.get(0).getClass() == String.class) {
            delimiter = "\\", \";
            prefix = "[\"";
            suffix = "\"]";
        }
        valuesString = attributeValues
            .stream()
            .map(Object::toString)
            .collect(Collectors.joining(delimiter, prefix, suffix));
    }
```

```
        final MessageAttributeValue messageAttributeValue = new
MessageAttributeValue()
            .withDataType("String.Array")
            .withStringValue(valuesString);
        messageAttributes.put(attributeName, messageAttributeValue);
    }

    public void addAttribute(final String attributeName, final Number attributeValue)
    {
        final MessageAttributeValue messageAttributeValue = new
MessageAttributeValue()
            .withDataType("Number")
            .withStringValue(attributeValue.toString());
        messageAttributes.put(attributeName, messageAttributeValue);
    }

    public String publish(final AmazonSNS snsClient, final String topicArn) {
        final PublishRequest request = new PublishRequest(topicArn, message)
            .withMessageAttributes(messageAttributes);
        final PublishResult result = snsClient.publish(request);
        return result.getMessageId();
    }
}
```

The following code excerpt initializes and uses the example `SNSMessageAttributes` class.

```
// Initialize the example class.
final SNSMessageAttributes message = new SNSMessageAttributes(messageBody);

// Add message attributes with string values.
message.addAttribute("store", "example_corp");
message.addAttribute("event", "order_placed");

// Add a message attribute with a list of string values.
final ArrayList<String> interestsValues = new ArrayList<String>();
interestsValues.add("soccer");
interestsValues.add("rugby");
interestsValues.add("hockey");
message.addAttribute("customer_interests", interestsValues);

// Add a message attribute with a numeric value.
message.addAttribute("price_usd", 1000);

// Add a Boolean attribute for filtering using subscription filter policies.
// The class applies the String.Array data type to this attribute, allowing it
// to be evaluated by a filter policy.
final ArrayList<Boolean> encryptedVal = new ArrayList<Boolean>();
encryptedVal.add(false);
message.addAttribute("encrypted", encryptedVal);

// Publish the message.
message.publish(snsClient, topicArn);

// Print the MessageId of the message.
System.out.println("MessageId: " + publishResponse.getMessageId());
```

3. Compile and run your code.

The message is published and the `MessageId` is printed, for example:

```
MessageId: 1234a567-bc89-012d-3e45-6fg7h890123i
```

To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for .NET

1. Specify your AWS credentials. For more information, see [Configuring AWS Credentials](#) in the *AWS SDK for .NET Developer Guide*.
2. Write your code. For more information, see [Programming with the AWS SDK for .NET](#).
3. The following code example helps simplify the process of publishing messages with attributes. The `SNSMessageAttributes` class stores the `MessageAttributes` field as a dictionary. You can use the overloaded `addAttribute` method to add attributes with the data types `String`, `String.Array`, and `Number`. To publish the message, use the `publish` method by providing the `AmazonSimpleNotificationServiceClient` client and the topic ARN.

```
using System;
using System.Collections.Generic;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSCreatePlatformEndpoint
{
    class SNSMessageAttributes
    {
        private String message;
        private Dictionary<String, MessageAttributeValue> messageAttributes;

        public SNSMessageAttributes(String message)
        {
            this.message = message;
            messageAttributes = new Dictionary<string, MessageAttributeValue>();
        }

        public string Message
        {
            get => this.message;
            set => this.message = value;
        }

        public void AddAttribute(String attributeName, String attributeValue)
        {
            messageAttributes[attributeName] = new MessageAttributeValue
            {
                DataType = "String",
                StringValue = attributeValue
            };
        }

        public void AddAttribute(String attributeName, float attributeValue)
        {
            messageAttributes[attributeName] = new MessageAttributeValue
            {
                DataType = "Number",
                StringValue = attributeValue.ToString()
            };
        }

        public void AddAttribute(String attributeName, int attributeValue)
        {
            messageAttributes[attributeName] = new MessageAttributeValue
            {
                DataType = "Number",
                StringValue = attributeValue.ToString()
            };
        }
    }
}
```

```

        };
    }

    public void AddAttribute(String attributeName, List<String> attributeValue)
    {
        String valueString = "[" + String.Join(" ", attributeValue.ToArray()) +
attributeValue.ToArray() + "]";
        messageAttributes[attributeName] = new MessageAttributeValue
        {
            DataType = "String.Array",
            StringValue = valueString
        };
    }

    public void AddAttribute(String attributeName, List<float> attributeValue)
    {
        String valueString = "[" + String.Join(" ", attributeValue.ToArray()) +
"]";
        messageAttributes[attributeName] = new MessageAttributeValue
        {
            DataType = "String.Array",
            StringValue = valueString
        };
    }

    public void AddAttribute(String attributeName, List<int> attributeValue)
    {
        String valueString = "[" + String.Join(" ", attributeValue.ToArray()) +
"]";
        messageAttributes[attributeName] = new MessageAttributeValue
        {
            DataType = "String.Array",
            StringValue = valueString
        };
    }

    public void AddAttribute(String attributeName, List<Boolean> attributeValue)
    {
        String valueString = "[" + String.Join(" ", attributeValue.ToArray()) +
"]";
        messageAttributes[attributeName] = new MessageAttributeValue
        {
            DataType = "String.Array",
            StringValue = valueString
        };
    }

    public String Publish(AmazonSimpleNotificationServiceClient snsClient, String
topicArn)
    {
        PublishRequest request = new PublishRequest
        {
            TopicArn = topicArn,
            MessageAttributes = messageAttributes,
            Message = message
        };
        PublishResponse result = snsClient.Publish(request);
        return result.MessageId;
    }
}

```

The following code excerpt initializes and uses the example `SNSMessageAttributes` class.

```
// Initialize the example class.
SNSMessageAttributes message = new SNSMessageAttributes(messageBody);

// Add message attributes with string values.
message.AddAttribute("store", "example_corp");
message.AddAttribute("event", "order_placed");

// Add a message attribute with a list of string values.
List<String> interestsValues = new List<String>();
interestsValues.Add("soccer");
interestsValues.Add("rugby");
interestsValues.Add("hockey");
message.AddAttribute("customer_interests", interestsValues);

// Add a message attribute with a numeric value.
message.AddAttribute("price_usd", 1000);

// Add a Boolean attribute for filtering using subscription filter policies.
// The class applies a String.Array data type to this attribute, allowing it
// to be evaluated by a filter policy.
List<Boolean> encryptedVal = new List<Boolean>();
encryptedVal.Add(false);
message.AddAttribute("encrypted", encryptedVal);

// Publish the message.
String msgId = message.Publish(snsClient, topicArn);

// Print the MessageId of the published message.
Console.WriteLine("MessageId: " + msgId);
```

4. Compile and run your code.

The message is published and the MessageId is printed, for example:

```
MessageId: 1234a567-bc89-012d-3e45-6fg7h890123i
```

Working with Amazon SNS Securely

This section contains the tutorials related to working with Amazon SNS securely.

Topics

- [Tutorial: Enabling Server-Side Encryption \(SSE\) for an Amazon SNS Topic \(p. 27\)](#)
- [Tutorial: Enabling Server-Side Encryption \(SSE\) for an Amazon SNS Topic with an Encrypted Amazon SQS Queue Subscribed \(p. 30\)](#)
- [Tutorial: Publishing Amazon SNS Messages Privately from Amazon VPC \(p. 32\)](#)

Tutorial: 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 \(p. 180\)](#).

Important

All requests to topics with SSE enabled must use HTTPS and [Signature Version 4](#).

The following tutorial shows how to enable, disable, and configure SSE for an existing Amazon SNS topic using the AWS Management Console and the AWS SDK for Java (by setting the `KmsMasterKeyId` attribute using the `CreateTopic` and `SetTopicAttributes` API actions).

Topics

- [To Enable Server-Side Encryption \(SSE\) for an Amazon SNS Topic Using the AWS Management Console \(p. 28\)](#)
- [To Enable Server-Side Encryption \(SSE\) for an Amazon SNS Topic Using the AWS SDK for Java \(p. 29\)](#)

To Enable Server-Side Encryption (SSE) for an Amazon SNS Topic Using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic and choose **Actions, Edit**.
4. Expand the **Encryption** section and do the following:
 - a. Choose **Enable encryption**.
 - b. Specify the customer master key (CMK). For more information, see [Key Terms \(p. 181\)](#).

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

Important

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

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

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

Note

Keep the following in mind:

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

Note

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

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

SSE is enabled for your topic and the **MyTopic** page is displayed.

The topic's **Encryption** status, **AWS Account**, **Customer master key (CMK)**, **CMK ARN**, and **Description** are displayed on the **Encryption** tab.

To Enable Server-Side Encryption (SSE) for an Amazon SNS Topic Using the AWS SDK for Java

1. Configure AWS KMS key policies to allow encryption of topics and encryption and decryption of messages. For more information, see [Configuring AWS KMS Permissions](#) (p. 183)
2. Specify your AWS credentials. For more information, see [Set up AWS Credentials and Region for Development](#) in the *AWS SDK for Java 2.x Developer Guide*.
3. Obtain the customer master key (CMK) ID. For more information, see [Key Terms](#) (p. 181).

Note

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SNS for a topic, AWS KMS creates the AWS managed CMK for Amazon SNS.
 - Alternatively, the first time you use the `Publish` action on a topic with SSE enabled, AWS KMS creates the AWS managed CMK for Amazon SNS.
4. Write your code. For more information, see [Using the SDK for Java 2.x](#).

To enable server-side encryption, specify the CMK ID by setting the `KmsMasterKeyId` attribute using the [CreateTopic](#) or [SetTopicAttributes](#) action.

The following code excerpt enables SSE for an existing topic using the AWS managed CMK for Amazon SNS:

```
// Enable server-side encryption by specifying the alias ARN of the AWS managed CMK for Amazon SNS.
final String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/aws/sns";

final SetTopicAttributesRequest setAttributesRequest = new SetTopicAttributesRequest()
    .withTopicArn(topicArn)
    .withAttributeName("KmsMasterKeyId")
    .withAttributeValue(kmsMasterKeyAlias);

final SetTopicAttributesResponse setAttributesResponse =
    snsClient.setTopicAttributes(setAttributesRequest)
```

To disable server-side encryption for an existing topic, set the `KmsMasterKeyId` attribute to an empty string using the `SetTopicAttributes` action.

Important

`null` isn't a valid value for `KmsMasterKeyId`.

The following code excerpt creates a new topic with SSE using a custom CMK:

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

// Enable server-side encryption by specifying the alias ARN of the custom CMK.
final String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/MyAlias";
attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);

final CreateTopicRequest createRequest = new CreateTopicRequest("MyTopic")
    .withAttributes(attributes);

final CreateTopicResponse createResponse = snsClient.createTopic(createRequest);
```

Tutorial: 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. 180).

The following tutorial shows how you can enable SSE for an Amazon SNS topic to which an encrypted Amazon SQS queue is subscribed, using the AWS Management Console.

Step 1: To Create a Custom CMK

1. Sign in to the [AWS KMS console](#) with a user that has at least the `AWSKeyManagementServicePowerUser` policy.
2. Choose **Create a key**.
3. On the **Add alias and description** page, enter an **Alias** for your key (for example, `MyCustomCMK`) and then choose **Next**.
4. On the **Add tags** page, choose **Next**.
5. On the **Define key administrative permissions** page, in the **Key administrators** section, choose an IAM role or an IAM user and then choose **Next**.
6. On the **Define key usage permissions** page, in the **This account** section, choose an IAM role or an IAM user and then choose **Next**.
7. On the **Review and edit key policy** page, add the following statement to the key policy, and then choose **Finish**.

```
{
  "Sid": "Allow Amazon SNS to use this key",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*"
}
```

Your new custom CMK appears in the list of keys.

Step 2: To Create an Encrypted Amazon SNS Topic

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. Choose **Create topic**.
4. On the **Create new topic** page, for **Name**, enter a topic name (for example, `MyEncryptedTopic`) and then choose **Create topic**.
5. Expand the **Encryption** section and do the following:

- a. Choose **Enable server-side encryption**.
- b. Specify the customer master key (CMK). For more information, see [Key Terms \(p. 181\)](#).

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 \(p. 30\)](#) 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 AWS KMS Customer Master Key (CMK)**, choose **MyCustomCMK** [which you created earlier \(p. 30\)](#), 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 \(p. 31\)](#) is displayed.

4. Choose **More Details** to view your message.
5. When you're finished, choose **Close**.
6. Repeat the process for **MyEncryptedQueue2**.

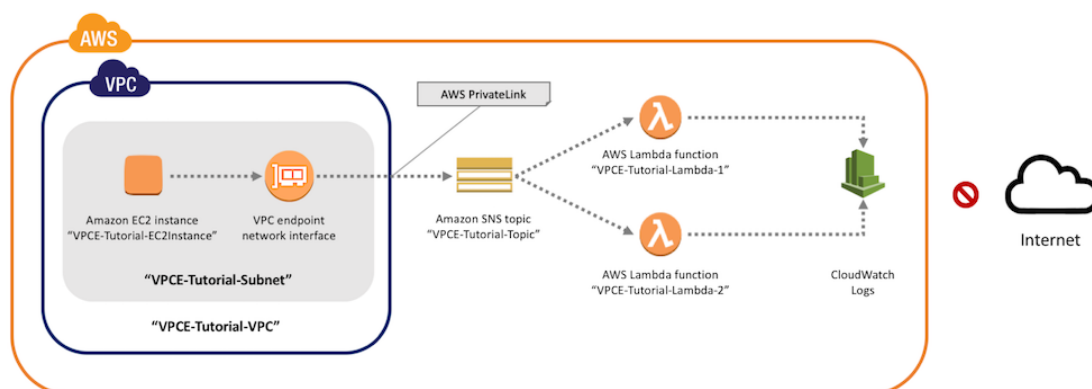
Tutorial: Publishing Amazon SNS Messages Privately from Amazon VPC

In this tutorial, you learn 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).

To complete this tutorial, you:

- 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 for this tutorial 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 this tutorial:



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.

This tutorial takes about 20 minutes to complete.

Topics

- [Before You Begin](#) (p. 33)
- [Step 1: Create an Amazon EC2 Key Pair](#) (p. 33)
- [Step 2: Create the AWS Resources](#) (p. 34)
- [Step 3: Confirm That Your Amazon EC2 Instance Lacks Internet Access](#) (p. 35)
- [Step 4: Create an Amazon VPC Endpoint for Amazon SNS](#) (p. 36)
- [Step 5: Publish a Message to Your Amazon SNS Topic](#) (p. 38)
- [Step 6: Verify Your Message Deliveries](#) (p. 39)
- [Step 7: Clean Up](#) (p. 40)
- [Related Resources](#) (p. 41)

Before You Begin

Before you start this tutorial, 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**.

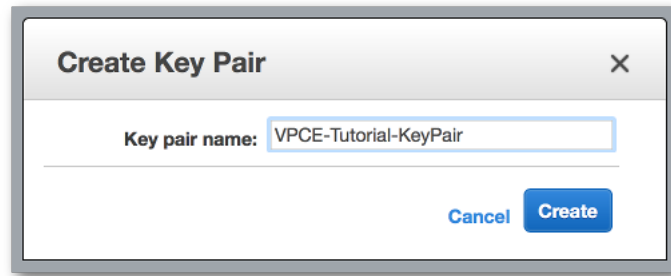
Step 1: Create an Amazon EC2 Key Pair

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 in this tutorial, you use the key pair to log in to an Amazon EC2 instance. To log in, you specify the name of the key pair, and you provide the private key.

To create the key pair

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation menu on the left, find the **Network & Security** section. Then, choose **Key Pairs**.

3. Choose **Create Key Pair**.
4. In the **Create Key Pair** window, for **Key pair name**, type **VPCE-Tutorial-KeyPair**. Then, choose **Create**.



5. The private key file is automatically downloaded by your browser. Save it in a safe place. Amazon EC2 gives the file an extension of `.pem`.
6. (Optional) If you're using an SSH client on a Mac or Linux computer to connect to your instance, use the `chmod` command to set the permissions of your private key file so that only you can read it:
 - a. Open a terminal and navigate to the directory that contains the private key:

```
$ cd /filepath_to_private_key/
```

- b. Set the permissions using the following command:

```
$ chmod 400 VPCE-Tutorial-KeyPair.pem
```

Step 2: Create the AWS Resources

To set up the infrastructure that supports this tutorial, 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 tutorial 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 the tutorial, 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 tutorial includes the following resources:

- A VPC and the associated networking resources, including a subnet, a security group, an internet gateway, and a route table.
- An Amazon EC2 instance that's launched into the subnet in the VPC.
- An Amazon SNS topic.
- Two AWS Lambda functions. These functions receive messages that are published to the Amazon SNS topic, and they log events in CloudWatch Logs.
- Amazon CloudWatch metrics and logs.
- An IAM role that allows the Amazon EC2 instance to use Amazon SNS, and an IAM role that allows the Lambda functions to write to CloudWatch logs.

To create the AWS resources

1. Download the [template file](#) from the GitHub website.

2. Sign in to the [AWS CloudFormation console](#).
3. Choose **Create Stack**.
4. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.
5. On the **Specify Details** page, specify stack and key names:
 - a. For **Stack name**, type **VPCE-Tutorial-Stack**.
 - b. For **KeyName**, choose **VPCE-Tutorial-KeyPair**.
 - c. For **SSHLocation**, keep the default value of **0.0.0.0/0**.

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

KeyName Name of an existing EC2 KeyPair to enable SSH access to the instance

SSHLocation The IP address range that can be used to SSH to the EC2 instance

- d. Choose **Next**.
6. On the **Options** page, keep all of the default values, and choose **Next**.
7. On the **Review** page, verify the stack details.
8. Under **Capabilities**, acknowledge that AWS CloudFormation might create IAM resources with custom names.
9. Choose **Create**.

The AWS CloudFormation console opens the **Stacks** page. The VPCE-Tutorial-Stack has a status of **CREATE_IN_PROGRESS**. In a few minutes, after the creation process completes, the status changes to **CREATE_COMPLETE**.

Create Stack

Actions

Design template

Filter: Active

By Stack Name

	Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/>	VPCE-Tutorial-Stack	2018-05-18 16:38:06 UTC-0700	CREATE_COMPLETE	CloudFormation Template for SNS VPC Endpoints Tutorial

Tip

Choose the **Refresh** button to see the latest stack status.

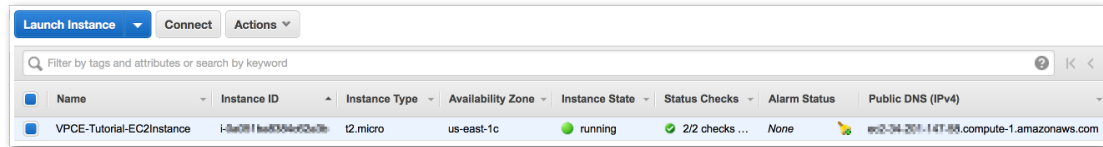
Step 3: Confirm That Your Amazon EC2 Instance Lacks Internet Access

The Amazon EC2 instance that was launched in your VPC in the previous step lacks internet access. It disallows outbound traffic, and it's unable to publish messages to Amazon SNS. Verify this by logging in to the instance. Then, attempt to connect to a public endpoint, and attempt to message Amazon SNS.

At this point in the tutorial, 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. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
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.

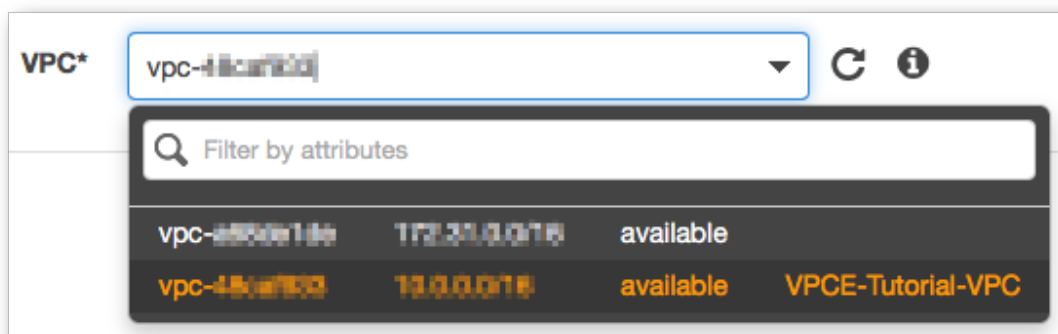
To create the endpoint

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.

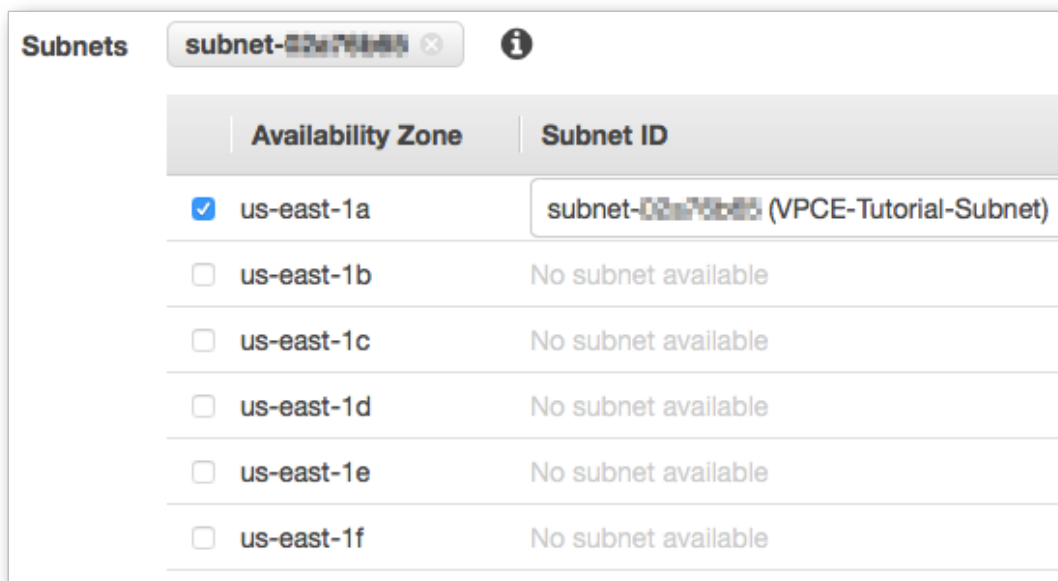
2. In the navigation menu on the left, choose **Endpoints**.
3. Choose **Create Endpoint**.
4. On the **Create Endpoint** page, for **Service category**, keep the default choice of **AWS services**.
5. For **Service Name**, choose the service name for Amazon SNS.

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**.

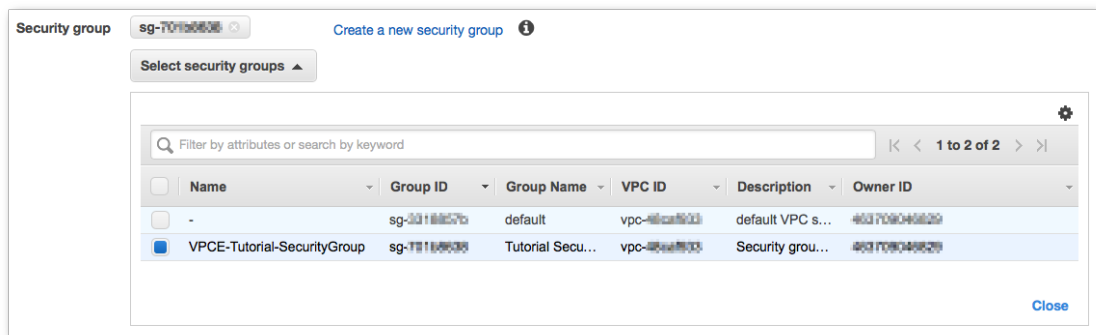
6. For **VPC**, choose the VPC that has the name **VPCE-Tutorial-VPC**.



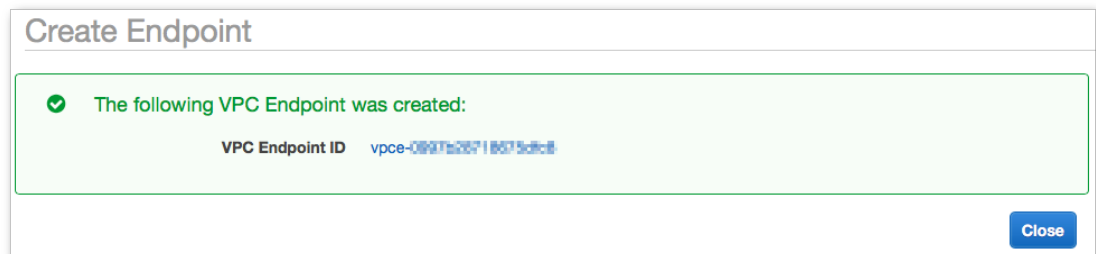
7. For **Subnets**, choose the subnet that has *VPCE-Tutorial-Subnet* in the subnet ID.



8. For **Enable Private DNS Name**, select **Enable for this endpoint**.
9. For **Security group**, choose **Select security group**, and choose **VPCE-Tutorial-SecurityGroup**.

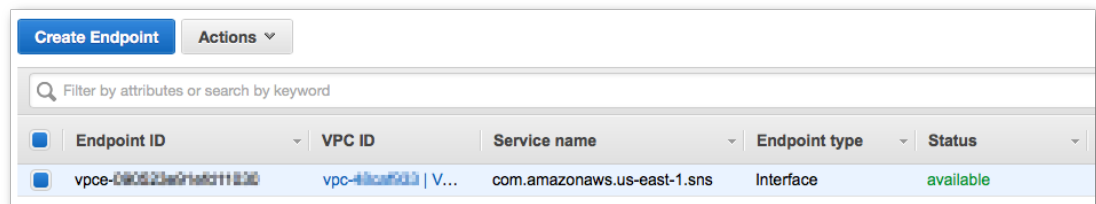


10. Choose **Create endpoint**. The Amazon VPC console confirms that a VPC endpoint was created.



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-d169-5be6-9042-410dfc9e86de"
}
```

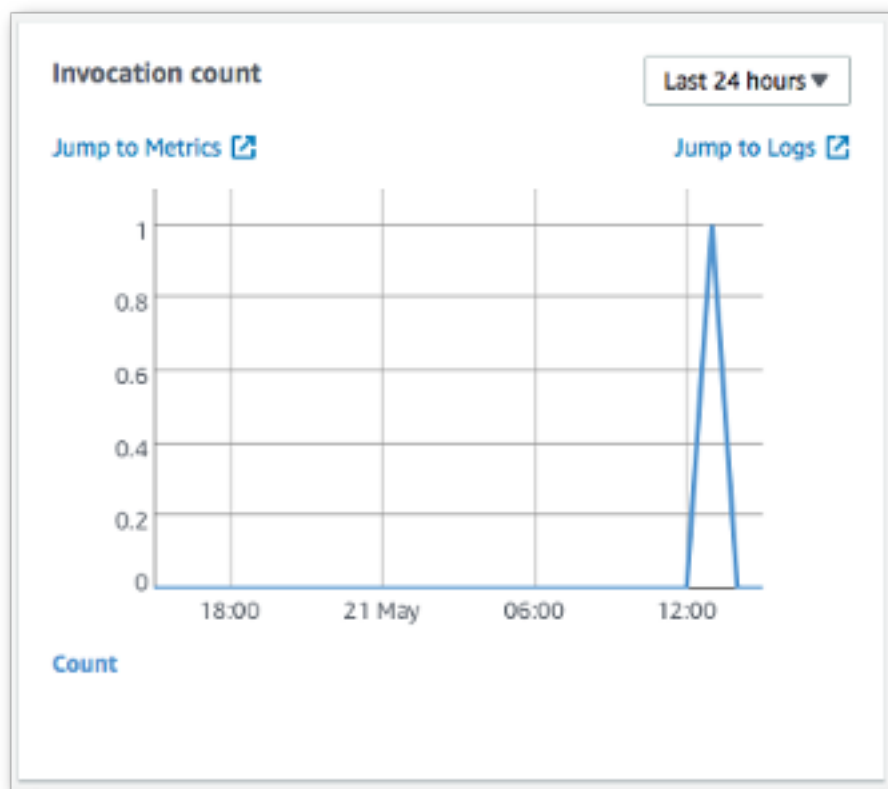
Step 6: Verify Your Message Deliveries

When the Amazon SNS topic receives a message, it fans out the message by sending it to the two subscribing Lambda functions. When these functions receive the message, they log the event to CloudWatch logs. To verify that your message delivery succeeded, check that the functions were invoked, and check that the CloudWatch logs were updated.

To verify that the Lambda functions were invoked

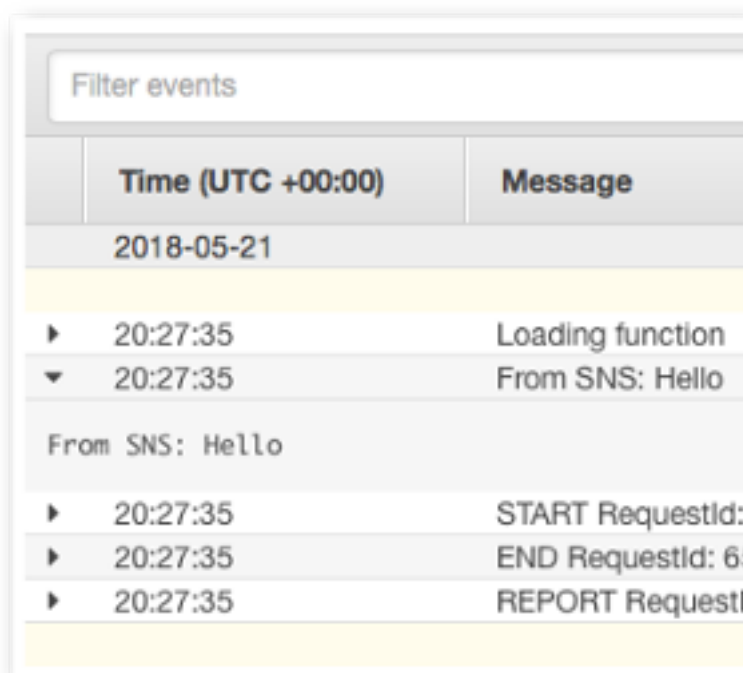
1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. On the **Functions** page, choose **VPCE-Tutorial-Lambda-1**.
3. Choose **Monitoring**.
4. Check the **Invocation count** graph. This graph shows the number of times that the Lambda function has been run.

The invocation count matches the number of times you published a message to the topic.



To verify that the CloudWatch logs were updated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation menu on the left, choose **Logs**.
3. Check the logs that were written by the Lambda functions:
 - a. Choose the `/aws/lambda/VPCE-Tutorial-Lambda-1/` log group.
 - b. Choose the log stream.
 - c. Check that the log includes the entry `From SNS: Hello`.



	Time (UTC +00:00)	Message
2018-05-21		
▶	20:27:35	Loading function
▼	20:27:35	From SNS: Hello
		From SNS: Hello
▶	20:27:35	START RequestId:
▶	20:27:35	END RequestId: 65
▶	20:27:35	REPORT RequestId:

- 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 for this tutorial, you can delete them now. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

First, delete your VPC endpoint using the Amazon VPC console. Then, delete the other resources that you created by deleting the stack in the AWS CloudFormation console. When you delete a stack, AWS CloudFormation removes the stack's resources from your AWS account.

To delete your VPC endpoint

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation menu on the left, choose **Endpoints**.
3. Select the endpoint that you created.
4. Choose **Actions**, and then choose **Delete Endpoint**.
5. In the **Delete Endpoint** window, choose **Yes, Delete**.

The endpoint status changes to **deleting**. When the deletion completes, the endpoint is removed from the page.

To delete your AWS CloudFormation stack

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
2. Select the stack **VPCE-Tutorial-Stack**.
3. Choose **Actions**, and then choose **Delete Stack**.
4. In the **Delete Stack** window, choose **Yes, Delete**.

The stack status changes to **DELETE_IN_PROGRESS**. When the deletion completes, the stack is removed from the page.

Related Resources

If you want to dive more deeply into the concepts introduced in this tutorial, 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](#)

Working with AWS Event Fork Pipelines

This section contains tutorials related to working with AWS Event Fork Pipelines.

Topics

- [Tutorial: Deploying and Testing the AWS Event Fork Pipelines Sample Application \(p. 41\)](#)
- [Subscribing AWS Event Fork Pipelines to an Amazon SNS Topic \(p. 48\)](#)

Tutorial: 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 \(p. 115\)](#).

The following tutorial 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*.

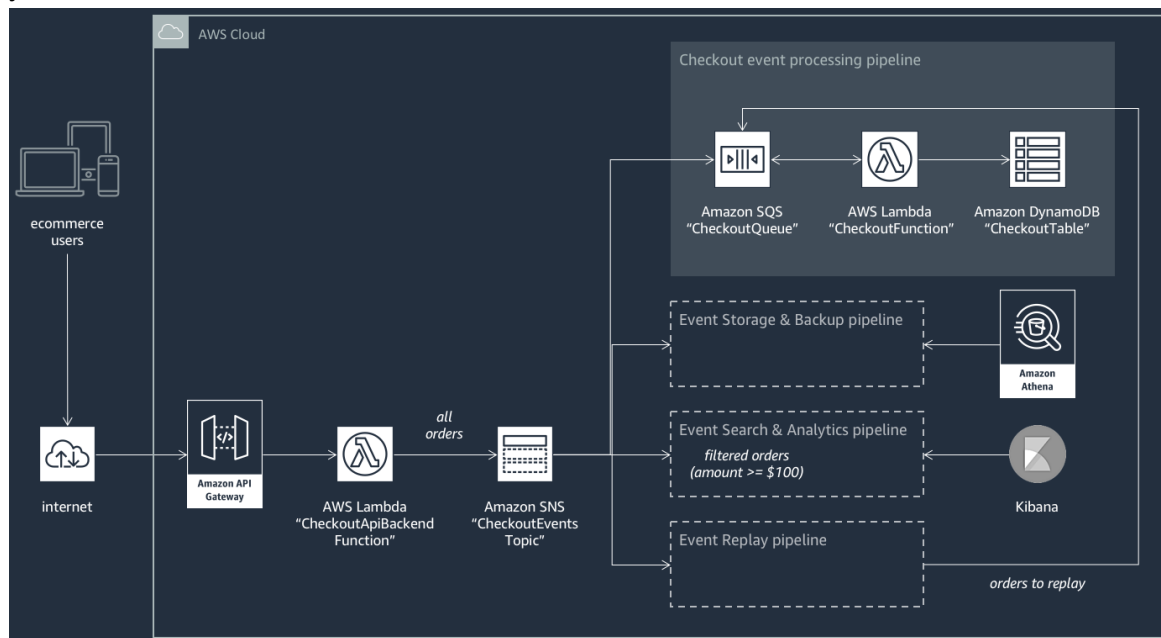
Topics

- [Example AWS Event Fork Pipelines Use Case \(p. 42\)](#)
- [Step 1: To Deploy the Sample Application \(p. 43\)](#)

- [Step 2: To Execute the Sample Application \(p. 44\)](#)
- [Step 3: To Verify the Execution of the Sample Application and Its Pipelines \(p. 45\)](#)
- [Step 4: To Simulate an Issue and Replay Events for Recovery \(p. 46\)](#)

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 \(p. 116\)](#) 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 \(p. 116\)](#) 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 Amazon ES for your Destination](#) in the *Amazon Kinesis Data Firehose Developer Guide*.

- [The Event Replay Pipeline \(p. 117\)](#) 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 [Message Filtering \(p. 67\)](#).

```
{
  "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://abcdefghijkl.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

4. Copy the following JSON to a file named `test_event.json`.

```
{
  "id": 15311,
  "date": "2019-03-25T23:41:11-08:00",
  "status": "confirmed",
  "customer": {
    "id": 65144,
    "name": "John Doe",
    "email": "john.doe@example.com"
  },
  "payment": {
    "id": 2509,
    "amount": 450.00,
    "currency": "usd",
    "method": "credit",
    "card-network": "visa",
    "card-number": "1234 5678 9012 3456",
    "card-expiry": "10/2022",
    "card-owner": "John Doe",
    "card-cvv": "123"
  },
  "shipping": {
    "id": 7600,
    "time": 2,
    "unit": "days",
  }
}
```

```
    "method": "courier"
  },
  "items": [{
    "id": 6512,
    "product": 8711,
    "name": "Hockey Jersey - Large",
    "quantity": 1,
    "price": 400.00,
    "subtotal": 400.00
  }, {
    "id": 9954,
    "product": 7600,
    "name": "Hockey Puck",
    "quantity": 2,
    "price": 25.00,
    "subtotal": 50.00
  }]
}
```

5. To send an HTTPS request to your API endpoint, pass the sample event payload as input by executing a `curl` command, for example:

```
curl -d "$(cat test_event.json)" https://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 Elasticsearch 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`.
3. Choose **Queue Actions**, **View/Delete Messages**.
4. In the **View/Delete Messages in fork-example-ecommerce-**my-app**...ReplayP-ReplayQueue-**123ABCD4E5F6**** dialog box, choose **Start Polling for Messages**.
5. To verify that the event is enqueued, choose **More Details** next to the message that appears in the queue.

Step 4: To Simulate an Issue and Replay Events for Recovery

Step 1: To Enable the Simulated Issue and Send a Second API Request

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutFunction`.
4. On the `fork-example-ecommerce-my-app-CheckoutFunction-ABCDEF...` page, in the **Environment variables** section, set the **BUG_ENABLED** variable to **true** and then choose **Save**.
5. Copy the following JSON to a file named `test_event_2.json`.

```
{
  "id": 9917,
  "date": "2019-03-26T21:11:10-08:00",
  "status": "confirmed",
  "customer": {
    "id": 56999,
    "name": "Marcia Oliveira",
    "email": "marcia.oliveira@example.com"
  },
  "payment": {
    "id": 3311,
    "amount": 75.00,
    "currency": "usd",
    "method": "credit",
    "card-network": "mastercard",
    "card-number": "1234 5678 9012 3456",
    "card-expiry": "12/2025",
    "card-owner": "Marcia Oliveira",
    "card-cvv": "321"
  },
  "shipping": {
    "id": 9900,
    "time": 20,
    "unit": "days",
```

```
        "method": "plane"
      },
      "items": [{
        "id": 9993,
        "product": 3120,
        "name": "Hockey Stick",
        "quantity": 1,
        "price": 75.00,
        "subtotal": 75.00
      }]
    }
  }
```

6. To send an HTTPS request to your API endpoint, pass the sample event payload as input by executing a `curl` command, for example:

```
curl -d "$(cat test_event_2.json)" https://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](#).
2. On the navigation panel, choose **Tables**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutTable`.
4. On the table details page, choose **Items** and then choose the created item.

The stored attributes are displayed, some marked as **CORRUPTED!**

Step 3: To Disable the Simulated Issue

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutFunction`.
4. On the `fork-example-ecommerce-my-app-CheckoutFunction-ABCDEF...` page, in the **Environment variables** section, set the **BUG_ENABLED** variable to **false** and then choose **Save**.

Step 4: To Enable Replay to Recover from the Issue

1. In the AWS Lambda console, on the navigation panel, choose **Functions**.
2. Search for `serverlessrepo-fork-example` and choose `ReplayFunction`.
3. Expand the **Designer** section, choose the **SQS** tile and then, in the **SQS** section, choose **Enabled**.

Note

It takes approximately 1 minute for the Amazon SQS event source trigger to become enabled.

4. Choose **Save**.
5. To view the recovered attributes, return to the Amazon DynamoDB console.
6. To disable replay, return to the AWS Lambda console and disable the Amazon SQS event source trigger for `ReplayFunction`.

Subscribing AWS Event Fork Pipelines to an Amazon SNS Topic

To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source [nested applications](#), based on the [AWS Serverless Application Model](#) (AWS SAM), which you can deploy directly from the [AWS Event Fork Pipelines suite](#) (choose **Show apps that create custom IAM roles or resource policies**) into your AWS account. For more information, see [How AWS Event Fork Pipelines Works](#) (p. 115).

The following tutorials 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. 8).

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

- [Tutorial: To Deploy and Subscribe the Event Storage and Backup Pipeline](#) (p. 48)
- [Tutorial: To Deploy and Subscribe the Event Search and Analytics Pipeline](#) (p. 50)
- [Tutorial: To Deploy and Subscribe the Event Replay Pipeline](#) (p. 52)

Tutorial: To Deploy and Subscribe the Event Storage and Backup Pipeline

This tutorial shows how to deploy the [Event Storage and Backup Pipeline](#) (p. 116) 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. 67) 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.

Tutorial: To Deploy and Subscribe the Event Search and Analytics Pipeline

This tutorial shows how to deploy the [Event Search and Analytics Pipeline \(p. 116\)](#) 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 Elasticsearch 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 \(p. 67\)](#) in this guide.

1. Sign in to the [AWS Lambda console](#).
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:
 - a. Choose **Browse serverless app repository, Public applications, Show apps that create custom IAM roles or resource policies**.
 - b. Search for `fork-event-search-analytics-pipeline` and then choose the application.
4. On the **fork-event-search-analytics-pipeline** page, do the following:
 - a. In the **Application settings** section, enter an **Application name** (for example, `my-app-search`).

Note
For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).
 - b. (Optional) For **DataTransformationFunctionArn**, enter the ARN of the Lambda function used for transforming incoming events. If you don't enter a value, data transformation is disabled.
 - c. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
 - **DEBUG**
 - **ERROR**
 - **INFO** (default)
 - **WARNING**
 - d. (Optional) For **SearchDomainArn**, enter the ARN of the Amazon ES 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 Amazon ES 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 Amazon ES 7.0)

- g. (Optional) Enter one of the following **SearchIndexRotationPeriod** settings for the rotation period of the Amazon ES 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 Amazon ES type for organizing the events in an index.

Note

- Amazon ES type names can contain any character (except null bytes) but can't begin with _.
- For Amazon ES 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 Amazon ES 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 Amazon ES 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 Amazon ES 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.

Tutorial: To Deploy and Subscribe the Event Replay Pipeline

This tutorial shows how to deploy the [Event Replay Pipeline \(p. 117\)](#) 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 \(p. 67\)](#) 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**.

How Amazon SNS Works

This section provides information about Amazon SNS topic and message attributes. It also describes message filtering, message and JSON formats, and large payload and raw message delivery.

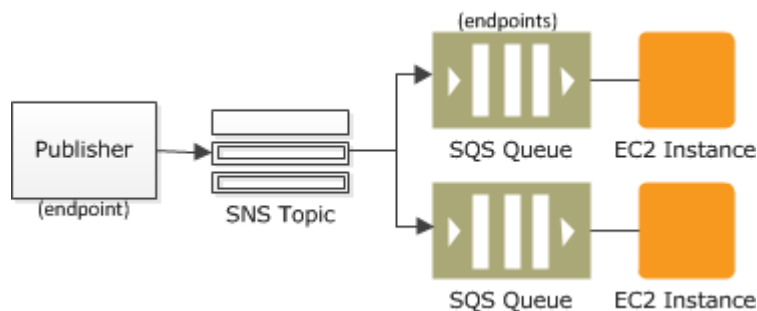
Topics

- [Common Amazon SNS Scenarios \(p. 54\)](#)
- [Amazon SNS Message Delivery Status \(p. 55\)](#)
- [Message Delivery Retries \(p. 58\)](#)
- [Amazon SNS Dead-Letter Queues \(p. 62\)](#)
- [Amazon SNS Message Attributes \(p. 64\)](#)
- [Amazon SNS Message Filtering \(p. 67\)](#)
- [Amazon SNS Message and JSON Formats \(p. 80\)](#)
- [Amazon SNS Large Payload and Raw Message Delivery \(p. 87\)](#)
- [Amazon SNS Tags \(p. 87\)](#)

Common Amazon SNS Scenarios

Fanout

The "fanout" scenario is when an Amazon SNS message is sent to a topic and then replicated and pushed to multiple Amazon SQS queues, HTTP endpoints, or email addresses. This allows for parallel asynchronous processing. For example, you could develop an application that sends an Amazon SNS message to a topic whenever an order is placed for a product. Then, the Amazon SQS queues that are subscribed to that topic would receive identical notifications for the new order. The Amazon EC2 server instance attached to one of the queues could handle the processing or fulfillment of the order, while the other server instance could be attached to a data warehouse for analysis of all orders received.



Another way to use "fanout" is to replicate data sent to your production environment with your development environment. Expanding upon the previous example, you could subscribe yet another queue to the same topic for new incoming orders. Then, by attaching this new queue to your development environment, you could continue to improve and test your application using data received from your production environment. For more information about sending Amazon SNS messages to Amazon SQS queues, see [With an Amazon SQS Queue as a Subscriber \(p. 90\)](#). For more information

about sending Amazon SNS messages to HTTP/S endpoints, see [With an HTTP/S Endpoint as a Subscriber](#) (p. 103).

Application and System Alerts

Application and system alerts are notifications that are triggered by predefined thresholds and sent to specified users by SMS and/or email. For example, since many AWS services use Amazon SNS, you can receive immediate notification when an event occurs, such as a specific change to your Amazon EC2 Auto Scaling group.

Push Email and Text Messaging

Push email and text messaging are two ways to transmit messages to individuals or groups via email and/or SMS. For example, you could use Amazon SNS to push targeted news headlines to subscribers by email or SMS. Upon receiving the email or SMS text, interested readers could then choose to learn more by visiting a website or launching an application. For more information about using Amazon SNS to send SMS notifications, see [Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber \(Send SMS\)](#) (p. 148).

Mobile Push Notifications

Mobile push notifications enable you to send messages directly to mobile apps. For example, you could use Amazon SNS for sending notifications to an app, indicating that an update is available. The notification message can include a link to download and install the update. For more information about using Amazon SNS to send direct notification messages to mobile endpoints, see [Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber \(Mobile Push\)](#) (p. 119)

Message Durability

Amazon SNS provides durable storage of all messages that it receives. When Amazon SNS receives your Publish request, it stores multiple copies of your message to disk. Before Amazon SNS confirms to you that it received your request, it stores the message in multiple isolated locations known as *Availability Zones*. The message is stored in Availability Zones that are located within your chosen AWS Region, such as the US East (N. Virginia) Region. Although rare, should a failure occur in one Availability Zone, Amazon SNS remains operational, and the durability of your messages persists.

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:

- Application
- HTTP
- Lambda
- SQS

After you configure the message delivery status attributes, log entries will be sent to CloudWatch Logs for messages sent to a topic subscribed to an Amazon SNS endpoint. 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 \(p. 56\)](#)
- [Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS Endpoints Using the AWS SDKs \(p. 56\)](#)

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 protocols for which you want to log delivery status, for example **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** subsection, do one of the following:
 - To choose an existing service role from your account, choose **Use existing service role** and then specify IAM roles for successful and failed deliveries.
 - To create a new service role in your account, choose **Create new service role**, choose **Create new roles** to define the IAM roles for successful and failed deliveries in the IAM console.

To give Amazon SNS write access to use CloudWatch Logs on your behalf, choose **Allow**.
8. Choose **Save changes**.

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

Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS Endpoints Using the AWS SDKs

The [AWS SDKs](#) provide APIs in several languages for using message delivery status attributes with Amazon SNS.

Topic Attributes

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

Application

- `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](#).

HTTP

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

Lambda

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

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 examples show how to configure topic attributes using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following Java example shows how to use the `SetTopicAttributes` API to configure topic attributes for message delivery status of notification messages sent to topics subscribed to Amazon SNS endpoints. In this example, it is assumed that string values have been set for `topicArn`, `attribName`, and `attribValue`.

```
final static String topicArn = ("arn:aws:sns:us-east-2:123456789012:MyTopic");  
final static String attribName = ("LambdaSuccessFeedbackRoleArn");
```

```
final static String attribValue = ("arn:aws:iam::123456789012:role/  
SNSSuccessFeedback");
```

```
SetTopicAttributesRequest setTopicAttributesRequest = new SetTopicAttributesRequest();  
setTopicAttributesRequest.withTopicArn(topicArn);  
setTopicAttributesRequest.setAttributeName(attribName);  
setTopicAttributesRequest.setAttributeValue(attribValue);
```

For more information about the SDK for Java, see [Getting Started with the AWS SDK for Java](#).
AWS SDK for .NET

The following .NET example shows how to use the SetTopicAttributes API to configure topic attributes for message delivery status of notification messages sent to topics subscribed to Amazon SNS endpoints. In this example, it is assumed that string values have been set for topicArn, attribName, and attribValue.

```
static String topicArn = "arn:aws:sns:us-east-2:123456789012:MyTopic";  
static String attribName = "LambdaSuccessFeedbackRoleArn";  
String attribValue = "arn:aws:iam::123456789012:role/SNSSuccessFeedback";
```

```
SetTopicAttributesRequest setTopicAttributesRequest = new SetTopicAttributesRequest {  
    TopicArn = topicArn,  
    AttributeName = attribName,  
    AttributeValue = attribValue  
};
```

For more information about the AWS SDK for .NET, see [Getting Started with the AWS SDK for .NET](#).

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 \(p. 62\)](#).

Delivery Protocols and Policies

Note

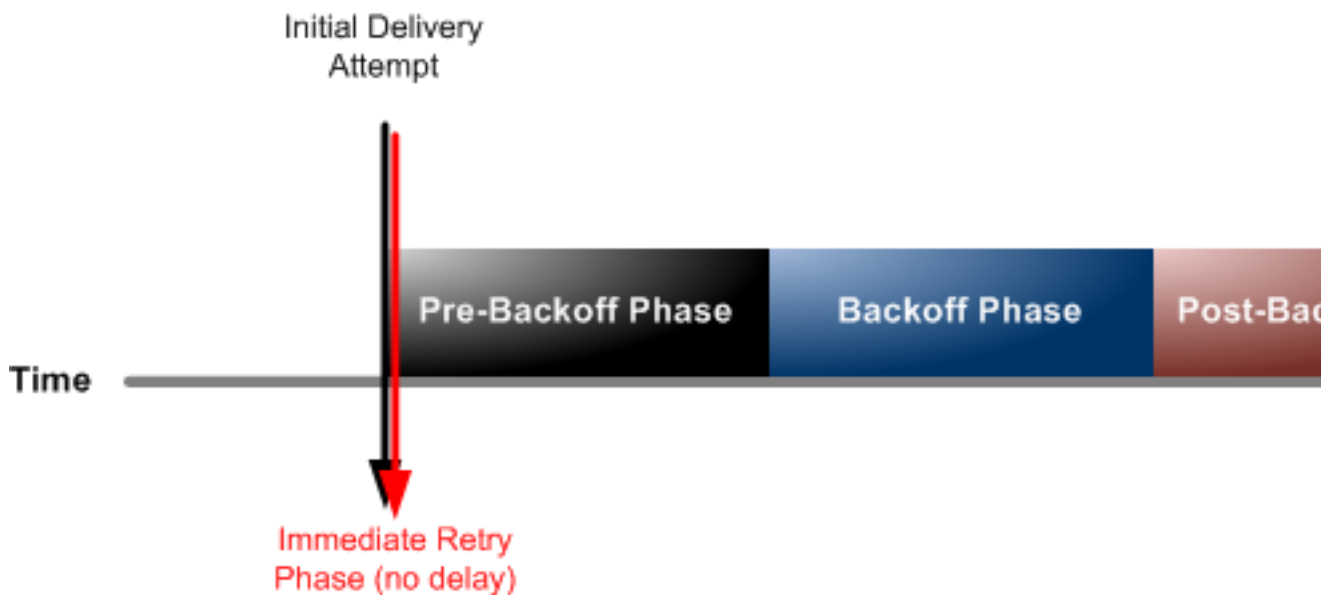
- Except for HTTP/S, you can't change Amazon SNS-defined delivery policies. Other delivery protocols don't support custom delivery policies.
- Amazon SNS applies jittering to delivery retries. For more information, see the [Exponential Backoff and Jitter](#) post on the *AWS Architecture Blog*.

Endpoint Type	Delivery Protocols	Immediate Retry (No Delay) Phase	Pre-Backoff Phase	Backoff Phase	Post-Backoff Phase	Total Attempts
	Amazon SQS		2 times, 1 second apart		100,000 times, 20	

Endpoint Type	Delivery Protocols	Immediate Retry (No Delay) Phase	Pre-Backoff Phase	Backoff Phase	Post-Backoff Phase	Total Attempts
AWS-managed endpoints	AWS Lambda	3 times, without delay		10 times, with exponential backoff, from 1 second to 20 seconds	seconds apart	100,015 times, over 23 days
Customer-managed endpoints	SMTP	0 times, without delay	2 times, 10 seconds apart	10 times, with exponential backoff, from 10 seconds to 600 seconds (10 minutes)	38 times, 600 seconds (10 minutes) apart	50 attempts, over 6 hours
	SMS					
	Mobile push					

Delivery Policy Stages

The following diagram shows the phases of a delivery policy.



Each delivery policy is comprised of four phases.

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

quickly the delay increases from the minimum to the maximum delay. The backoff function can be arithmetic, exponential, geometric, or linear.

4. **Post-Backoff Phase** – This phase follows the backoff phase. It specifies a number of retries and the amount of delay between them. This is the final phase.

Creating a Delivery Policy

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

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

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

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

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

Amazon SNS makes a total of 50 attempts before discarding the message.

Note

This delivery policy also instructs Amazon SNS to throttle deliveries to no more than 10 per second.

```
{
  "healthyRetryPolicy": {
    "minDelayTarget": 1,
    "maxDelayTarget": 60,
    "numRetries": 50,
    "numNoDelayRetries": 3,
    "numMinDelayRetries": 2,
    "numMaxDelayRetries": 35,
    "backoffFunction": "exponential"
  },
  "throttlePolicy": {
    "maxReceivesPerSecond": 10
  }
}
```

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

Policy	Description	Constraint
minDelayTarget	The minimum delay for a retry.	0 to maximum delay

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

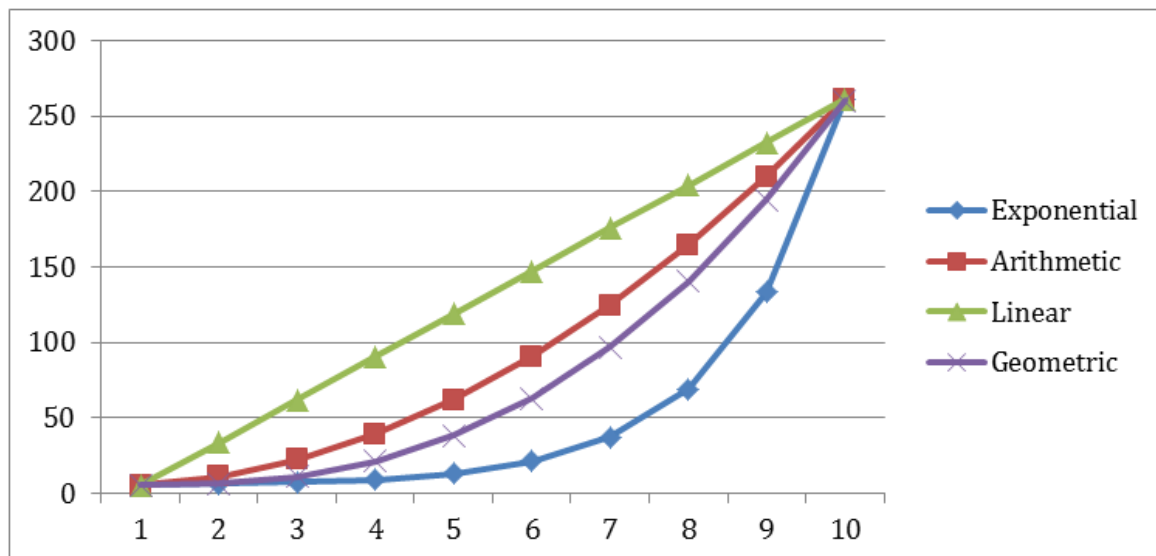
Amazon SNS uses the following formula to calculate the number of retries in the backoff phase:

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

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

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

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



Amazon SNS Dead-Letter Queues

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 an Amazon SNS Subscription](#) (p. 17) and [Message Delivery Retries](#) (p. 58).

Note

- The Amazon SNS subscription and Amazon SQS queue must be under the same AWS account and Region.
- Currently, you can't use an Amazon SQS FIFO queue as a dead-letter queue for an Amazon SNS subscription.
- To use an encrypted Amazon SQS queue as a dead-letter queue, you must use a custom CMK with a key policy that grants the Amazon SNS service principal access to AWS KMS API actions. For more information, see [Encryption at Rest](#) (p. 180) 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. 62)
- [How Do Dead-Letter Queues Work?](#) (p. 63)
- [How Are Messages Moved into a Dead-Letter Queue?](#) (p. 63)
- [How Can I Move Messages out of a Dead-Letter Queue?](#) (p. 64)
- [How Can I Monitor and Log Dead-Letter Queues?](#) (p. 64)

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 [Message Delivery Retries](#) (p. 58).

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 an Amazon SNS Subscription](#) (p. 17).

Note

Currently, you can't use an Amazon SQS FIFO queue as a dead-letter queue for an 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 five-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 \(p. 213\)](#).

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 \(p. 55\)](#).

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. Each message can have up to 10 attributes.

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

For information about sending messages with attributes using the AWS Management Console or the AWS SDK for Java, see the [Publishing a Message with Attributes to an Amazon SNS Topic \(p. 22\)](#) 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 [Message Filtering \(p. 67\)](#).

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. 65) 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:

- **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.
- **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`.
- **Number** – Numbers are positive or negative integers or floating-point numbers. Numbers have sufficient range and precision to encompass most of the possible values that integers, floats, and doubles typically support. A number can have a value from -10^9 to 10^9 , with 5 digits of accuracy after the decimal point. Leading and trailing zeroes are trimmed.
- **Binary** – Binary type attributes can store any binary data; for example, compressed data, encrypted data, or images.

Reserved Message Attributes for Mobile Push Notifications

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

Push Notification Service	Reserved Message Attribute
ADM	<code>AWS.SNS.MOBILE.ADM.TTL</code>
APNs	<code>AWS.SNS.MOBILE.APNS_MDM.TTL</code>

Push Notification Service	Reserved Message Attribute
	AWS.SNS.MOBILE.APNS_MDM_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_PASSBOOK.TTL
	AWS.SNS.MOBILE.APNS_PASSBOOK_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_VOIP.TTL
	AWS.SNS.MOBILE.APNS_VOIP_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_COLLAPSE_ID
	AWS.SNS.MOBILE.APNS_PRIORITY
	AWS.SNS.MOBILE.APNS_PUSH_TYPE
	AWS.SNS.MOBILE.APNS_TOPIC
	AWS.SNS.MOBILE.APNS_TTL
	AWS.SNS.MOBILE.PREFERRED_AUTHENTICATION_METHOD
Baidu	AWS.SNS.MOBILE.BAIDU.DeployStatus
	AWS.SNS.MOBILE.BAIDU.MessageKey
	AWS.SNS.MOBILE.BAIDU.MessageType
	AWS.SNS.MOBILE.BAIDU.TTL
FCM	AWS.SNS.MOBILE.FCM.TTL
	AWS.SNS.MOBILE.GCM.TTL
macOS	AWS.SNS.MOBILE.MACOS_SANDBOX.TTL
	AWS.SNS.MOBILE.MACOS.TTL
MPNS	AWS.SNS.MOBILE.MPNS.NotificationClass
	AWS.SNS.MOBILE.MPNS.TTL
	AWS.SNS.MOBILE.MPNS.Type
WNS	AWS.SNS.MOBILE.WNS.CachePolicy
	AWS.SNS.MOBILE.WNS.Group
	AWS.SNS.MOBILE.WNS.Match
	AWS.SNS.MOBILE.WNS.SuppressPopup
	AWS.SNS.MOBILE.WNS.Tag
	AWS.SNS.MOBILE.WNS.TTL
	AWS.SNS.MOBILE.WNS.Type

Amazon SNS Message Filtering

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

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

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

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

For more information, see the following:

- [Filter Messages Published to Topics](#)
- [Attribute String Value Matching \(Whitelisting, Blacklisting, and Prefix Matching\)](#) (p. 70)
- [Attribute Numeric Value Matching \(Whitelisting, Blacklisting, and Range Matching\)](#) (p. 71)
- [Attribute Key Matching](#) (p. 72)

Topics

- [Amazon SNS Subscription Filter Policies](#) (p. 67)
- [Tutorial: Applying a Subscription Filter Policy](#) (p. 73)
- [Tutorial: Removing a Subscription Filter Policy](#) (p. 76)
- [Subscription Filter Policies as Java Collections](#) (p. 77)

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 [Message Filtering](#) (p. 67).

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

A subscription accepts a message under the following conditions:

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

Topics

- [Example Message with Attributes \(p. 68\)](#)
- [Example Filter Policies \(p. 68\)](#)
- [Filter Policy Constraints \(p. 69\)](#)
- [Attribute String Value Matching \(p. 70\)](#)
- [Attribute Numeric Value Matching \(p. 71\)](#)
- [Attribute Key Matching \(p. 72\)](#)
- [AND/OR Logic \(p. 73\)](#)

Example Message with Attributes

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

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

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

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

For information about applying attributes to a message, see [Amazon SNS Message Attributes \(p. 64\)](#).

Example Filter Policies

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

A Policy that Accepts Messages

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

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

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

A Policy that Rejects Messages

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

If any mismatches occur, the policy rejects the message.

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

Filter Policy Constraints

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

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

Consider the following policy:

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

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

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

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

Attribute String Value Matching

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

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

Exact Matching (Whitelisting)

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"}
```

Anything-But Matching (Blacklisting)

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"}
```

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"}
```

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.

Exact Matching (Whitelisting)

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 (Blacklisting)

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 check whether an incoming message has an attribute whose key is listed in the filter policy.

Consider the following policy attribute:

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

It matches any message with at least 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"]
```

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`.

Tutorial: 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, such as the AWS SDK for Java.

AWS Management Console

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

Amazon SNS applies your filter policy to the subscription.

AWS CLI

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

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

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

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

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

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

AWS SDK for Java

The following examples show how to apply filter policies using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

To apply a filter policy with the AWS SDK for Java, use the `setSubscriptionAttributes` method of the `AmazonSNS` client. Provide a `SetSubscriptionAttributesRequest` object as the argument, as shown in the following example:

```
AmazonSNS snsClient = AmazonSNSClientBuilder.defaultClient();
String filterPolicyString = '{"store":["example_corp"],"event":["order_placed"]}'
SetSubscriptionAttributesRequest request =
    new SetSubscriptionAttributesRequest(subscriptionArn, "FilterPolicy",
        filterPolicyString);
snsClient.setSubscriptionAttributes(request);
```

To initialize the `SetSubscriptionAttributesRequest` object, provide the following arguments:

- `subscriptionArn` – The Amazon Resource Name (ARN) of the subscription to which the policy is applied.

- `attributeName` – Must be "FilterPolicy".
- `attributeValue` – Your JSON filter policy as a string. Because you must enclose the string policy in double quotes, remember to escape the double quotes that enclose the attribute names and values, as in `\\"store\\"`.

The `SetSubscriptionAttributesRequest` class accepts the filter policy as a string. If you want to define your policy as a Java collection, create a map that associates each attribute name with a list of values. To assign the policy to a subscription, you first produce a string version of the policy from the contents of the map. You then pass the string as the `attributeValue` argument to `SetSubscriptionAttributesRequest`.

AWS SDK for .NET

To apply a filter policy with the AWS SDK for .NET, use the `SetSubscriptionAttributes` method of the `AmazonSNS` client. Provide a `SetSubscriptionAttributesRequest` object as the argument, as shown in the following example:

```
AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient();
String filterPolicyString = "{\"store\":[\"example_corp\"],\"event\":[\"order_placed\"]]";
SetSubscriptionAttributesRequest request = new
    SetSubscriptionAttributesRequest(subscriptionArn, "FilterPolicy", filterPolicyString);
snsClient.setSubscriptionAttributes(request);
```

To initialize the `SetSubscriptionAttributesRequest` object, provide the following arguments:

- `subscriptionArn` – The Amazon Resource Name (ARN) of the subscription to which the policy is applied.
- `attributeName` – Must be "FilterPolicy".
- `attributeValue` – Your JSON filter policy as a string. Because you must enclose the string policy in double quotes, remember to escape the double quotes that enclose the attribute names and values, as in `\\"store\\"`.

The `SetSubscriptionAttributesRequest` class accepts the filter policy as a string. If you want to define your policy as a C# collection, create a dictionary that associates each attribute name with a list of values. To assign the policy to a subscription, you first produce a string version of the policy from the contents of the dictionary. You then pass the string as the `attributeValue` argument to `SetSubscriptionAttributesRequest`.

Amazon SNS API

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

AWS CloudFormation

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

1. Sign in to the [AWS CloudFormation console](#).
2. Choose **Create Stack**.

3. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.
4. On the **Specify Details** page, do the following:
 - a. For **Stack Name**, type `MyFilterPolicyStack`.
 - b. For **myHttpEndpoint**, type the HTTP endpoint to be subscribed to your topic.

Tip

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

5. On the **Options** page, choose **Next**.
6. On the **Review** page, choose **Create**.

Tutorial: 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-ef12g3hi.j456` 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 "{}"
```

AWS SDK for Java

The following examples show how to remove filter policies using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

To remove a filter policy with the AWS SDK for Java, use the `setSubscriptionAttributes` method of the `AmazonSNS` client. Provide a string that contains an empty JSON body as your filter policy:

```
AmazonSNS snsClient = AmazonSNSClientBuilder.defaultClient();
SetSubscriptionAttributesRequest request =
    new SetSubscriptionAttributesRequest(subscriptionArn, "FilterPolicy", "{}");
```

```
snsClient.setSubscriptionAttributes(request);
```

AWS SDK for .NET

To remove a filter policy with the AWS SDK for .NET, use the [SetSubscriptionAttributes](#) method of the `AmazonSNS` client. Provide a string that contains an empty JSON body as your filter policy:

```
AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient();
SetSubscriptionAttributesRequest request =
    new SetSubscriptionAttributesRequest(subscriptionArn, "FilterPolicy", "{}");
snsClient.SetSubscriptionAttributes(request);
```

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.

Subscription Filter Policies as Java Collections

To provide a subscription filter policy to the Amazon SNS client using the AWS SDK for Java, you must pass it as a string using the following process:

1. To define your policy as a collection, create a map that associates each attribute name with a list of values.
2. To assign the policy to a subscription, produce a string version of the policy from the contents of the map.
3. Pass the string to the Amazon SNS client.

Working Java Example

The following example Java code demonstrates the process of providing the subscription filter policy to the Amazon SNS client.

```
/*
 * Copyright 2010-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

// The class stores the filterPolicy field as a map.
public class SNSMessageFilterPolicy {

    private final Map<String, Attribute> filterPolicy = new HashMap<>();

    // You can use the addAttribute(), addAttributePrefix(), and addAttributeAnythingBut()
    // methods to add attributes to your policy. These methods accept the attribute name as
    // a string,
    // and they are specialized to accept different value types. You can pass values as
    // strings,
    // lists of strings, numbers, or number ranges and you can also add the attributes
    // anything-but
    // and prefix.
    public void addAttribute(final String attributeName, final String attributeValue) {
        filterPolicy.put(attributeName, new Attribute<>(AttributeType.String,
attributeValue));
    }

    public void addAttribute(final String attributeName, final ArrayList<String>
attributeValues) {
        final ArrayList<Attribute> attributes = new ArrayList<>();
        for (final String s : attributeValues) {
            attributes.add(new Attribute<>(AttributeType.String, s));
        }
        filterPolicy.put(attributeName, new Attribute<>(AttributeType.List, attributes));
    }

    public void addAttributePrefix(final String attributeName, final String prefix) {
        filterPolicy.put(attributeName, new Attribute<>(AttributeType.Prefix, prefix));
    }

    public void addAttributeAnythingBut(final String attributeName, final String value) {
        filterPolicy.put(attributeName, new Attribute<>(AttributeType.AnythingBut, value));
    }

    public <T extends Number> void addAttribute(final String attributeName, final String
op, final T value) {
        filterPolicy.put(attributeName, new Attribute<>(AttributeType.Numeric, new
NumericValue<>(op, value)));
    }

    public <T extends Number> void addAttributeRange(
        final String attributeName,
        final String lowerOp,
        final T lower,
        final String upperOp,
        final T upper) {
        filterPolicy.put(attributeName,
            new Attribute<>(AttributeType.Numeric, new NumericValue<>(lowerOp, lower,
upperOp, upper)));
    }

    // To apply your policy to a subscription, use the apply() method, providing the client
    // and the
    // subscription ARN. This method produces a policy string from the contents of the
    // filterPolicy map,
    // and then applies the policy to the specified subscription.
    public void apply(final AmazonSNS snsClient, final String subscriptionArn) {
        final SetSubscriptionAttributesRequest request =
            new SetSubscriptionAttributesRequest(subscriptionArn,
                "FilterPolicy", formatFilterPolicy());
    }
}
```

```
snsClient.setSubscriptionAttributes(request);
}

private String formatFilterPolicy() {
    return filterPolicy.entrySet()
        .stream()
        .map(entry -> "\"" + entry.getKey() + "\": [" + entry.getValue() + "]")
        .collect(Collectors.joining(", ", "{", "}"));
}

private enum AttributeType {
    String, Numeric, Prefix, List, AnythingBut
}

private class Attribute<T> {
    final T value;
    final AttributeType type;

    Attribute(final AttributeType type, final T value) {
        this.value = value;
        this.type = type;
    }

    public String toString() {
        switch (type) {
            case Prefix:
                return String.format("{\"prefix\":\"%s\"}", value.toString());
            case Numeric:
                return String.format("{\"numeric\":%s}", value.toString());
            case List:
                final List list = (List)value;
                final ArrayList<T> values = new ArrayList<T>(list);
                return values
                    .stream()
                    .map(Object::toString)
                    .collect(Collectors.joining(", "));
            case AnythingBut:
                return String.format("{\"anything-but\":\"%s\"}", value);
            default:
                return String.format("\"%s\"", value);
        }
    }
}

private class NumericValue<T extends Number> {
    private final T lower;
    private final T upper;
    private final String lowerOp;
    private final String upperOp;

    NumericValue(final String op, final T value) {
        lower = value;
        lowerOp = op;
        upper = null;
        upperOp = null;
    }

    NumericValue(final String lowerOp, final T lower, final String upperOp, final T
upper) {
        this.lower = lower;
        this.lowerOp = lowerOp;
        this.upper = upper;
        this.upperOp = upperOp;
    }

    public String toString() {
```

```
        final StringBuilder s = new StringBuilder("[")
            .append('\n')
            .append(lowerOp)
            .append("\n,")
            .append(lower);
        if (upper != null) {
            s.append("\n")
                .append(upperOp)
                .append("\n,")
                .append(upper);
        }
        s.append("]");
        return s.toString();
    }
}
```

The following Java code excerpt shows how to initialize and use the example `SNSMessageFilterPolicy` class:

```
// Initialize the example filter policy class.
final 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.
filterPolicy.addAttributePrefix("customer_interests", "bas");

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

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

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

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

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

Amazon SNS Message and JSON Formats

Amazon SNS uses the following formats.

Topics

- [HTTP/HTTPS Headers \(p. 81\)](#)
- [HTTP/HTTPS Subscription Confirmation JSON Format \(p. 81\)](#)
- [HTTP/HTTPS Notification JSON Format \(p. 83\)](#)
- [HTTP/HTTPS Unsubscribe Confirmation JSON Format \(p. 84\)](#)
- [SetSubscriptionAttributes Delivery Policy JSON Format \(p. 85\)](#)

- [SetTopicAttributes Delivery Policy JSON Format \(p. 86\)](#)

HTTP/HTTPS Headers

When Amazon SNS sends a subscription confirmation, notification, or unsubscribe confirmation message to HTTP/HTTPS endpoints, it sends a POST message with a number of Amazon SNS-specific header values. You can use these header values to do things such as identify the type of message without having to parse the JSON message body to read the `Type` value.

x-amz-sns-message-type

The type of message. The possible values are `SubscriptionConfirmation`, `Notification`, and `UnsubscribeConfirmation`.

x-amz-sns-message-id

A Universally Unique Identifier, unique for each message published. For a notification that Amazon SNS resends during a retry, the message ID of the original message is used.

x-amz-sns-topic-arn

The Amazon Resource Name (ARN) for the topic that this message was published to.

x-amz-sns-subscription-arn

The ARN for the subscription to this endpoint.

The following HTTP POST header is an example of a header for a Notification message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-
beaa-fcfcc21c8f55
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
```

HTTP/HTTPS Subscription Confirmation JSON Format

After you subscribe an HTTP/HTTPS endpoint, Amazon SNS sends a subscription confirmation message to the HTTP/HTTPS endpoint. This message contains a `SubscribeURL` value that you must visit to confirm the subscription (alternatively, you can use the `Token` value with the [ConfirmSubscription](#)).

Note

Amazon SNS doesn't send notifications to this endpoint until the subscription is confirmed

The subscription confirmation message is a POST message with a message body that contains a JSON document with the following name-value pairs.

Message

A string that describes the message. For subscription confirmation, this string looks like this:

```
You have chosen to subscribe to the topic arn:aws:sns:us-east-2:123456789012:MyTopic.
\nTo confirm the subscription, visit the SubscribeURL included in this message.
```

MessageId

A Universally Unique Identifier, unique for each message published. For a message that Amazon SNS resends during a retry, the message ID of the original message is used.

Signature

Base64-encoded "SHA1withRSA" signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

SignatureVersion

Version of the Amazon SNS signature used.

SigningCertURL

The URL to the certificate that was used to sign the message.

SubscribeURL

The URL that you must visit in order to confirm the subscription. Alternatively, you can instead use the Token with the [ConfirmSubscription](#) action to confirm the subscription.

Timestamp

The time (GMT) when the subscription confirmation was sent.

Token

A value you can use with the [ConfirmSubscription](#) action to confirm the subscription. Alternatively, you can simply visit the SubscribeURL.

TopicArn

The Amazon Resource Name (ARN) for the topic that this endpoint is subscribed to.

Type

The type of message. For a subscription confirmation, the type is `SubscriptionConfirmation`.

The following HTTP POST message is an example of a `SubscriptionConfirmation` message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "SubscriptionConfirmation",
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL included
in this message.",
```

```
"SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
west-2:123456789012:MyTopic&Token=2336412f37...",
"Timestamp" : "2012-04-26T20:45:04.751Z",
"SignatureVersion" : "1",
"Signature" : "EXAMPLEPH
+DcEwjAPg8O9mY8dReBSwksfg2S7WKQcikcNKWLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDvrY0Ad8L10Hs3zH8
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

HTTP/HTTPS Notification JSON Format

When Amazon SNS sends a notification to a subscribed HTTP or HTTPS endpoint, the POST message sent to the endpoint has a message body that contains a JSON document with the following name-value pairs.

Message

The Message value specified when the notification was published to the topic.

MessageId

A Universally Unique Identifier, unique for each message published. For a notification that Amazon SNS resends during a retry, the message ID of the original message is used.

Signature

Base64-encoded SHA1withRSA signature of the Message, MessageId, Subject (if present), Type, Timestamp, and TopicArn values.

SignatureVersion

Version of the Amazon SNS signature used.

SigningCertURL

The URL to the certificate that was used to sign the message.

Subject

The Subject parameter specified when the notification was published to the topic.

Note

This is an optional parameter. If no Subject was specified, then this name-value pair does not appear in this JSON document.

Timestamp

The time (GMT) when the notification was published.

TopicArn

The Amazon Resource Name (ARN) for the topic that this message was published to.

Type

The type of message. For a notification, the type is `Notification`.

UnsubscribeURL

A URL that you can use to unsubscribe the endpoint from this topic. If you visit this URL, Amazon SNS unsubscribes the endpoint and stops sending notifications to this endpoint.

The following HTTP POST message is an example of a Notification message to an HTTP endpoint.


```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEw6JRN...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

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-fcfc21c8f55.\n\nTo cancel this operation and restore the subscription, visit the SubscribeURL included in this message.
```

MessageId

A Universally Unique Identifier, unique for each message published. For a message that Amazon SNS resends during a retry, the message ID of the original message is used.

Signature

Base64-encoded "SHA1withRSA" signature of the Message, MessageId, Type, Timestamp, and TopicArn values.

SignatureVersion

Version of the Amazon SNS signature used.

SigningCertURL

The URL to the certificate that was used to sign the message.

SubscribeURL

The URL that you must visit in order to re-confirm the subscription. Alternatively, you can instead use the Token with the [ConfirmSubscription](#) action to re-confirm the subscription.

Timestamp

The time (GMT) when the unsubscribe confirmation was sent.

Token

A value you can use with the [ConfirmSubscription](#) action to re-confirm the subscription. Alternatively, you can simply visit the SubscribeURL.

TopicArn

The Amazon Resource Name (ARN) for the topic that this endpoint has been unsubscribed from.

Type

The type of message. For a unsubscribe confirmation, the type is UnsubscribeConfirmation.

The following HTTP POST message is an example of a UnsubscribeConfirmation message to an HTTP endpoint.

```
POST / HTTP/1.1
x-amz-sns-message-type: UnsubscribeConfirmation
x-amz-sns-message-id: 47138184-6831-46b8-8f7c-afc488602d7d
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1399
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "UnsubscribeConfirmation",
  "MessageId" : "47138184-6831-46b8-8f7c-afc488602d7d",
  "Token" : "2336412f37...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this operation and restore the subscription, visit the SubscribeURL included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37fb6...",
  "Timestamp" : "2012-04-26T20:06:41.581Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEHXgJm...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

SetSubscriptionAttributes Delivery Policy JSON Format

If you send a request to the SetSubscriptionAttributes action and set the AttributeName parameter to a value of DeliveryPolicy, the value of the AttributeValue parameter must be a valid JSON object. For example, the following example sets the delivery policy to 5 total retries.

```
http://sns.us-east-2.amazonaws.com/  
?Action=SetSubscriptionAttributes  
&SubscriptionArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic  
%3A80289ba6-0fd4-4079-afb4-ce8c8260f0ca  
&AttributeName=DeliveryPolicy  
&AttributeValue={"healthyRetryPolicy":{"numRetries":5}}  
...
```

Use the following JSON format for the value of the AttributeValue parameter.

```
{  
  "healthyRetryPolicy" : {  
    "minDelayTarget" : int,  
    "maxDelayTarget" : int,  
    "numRetries" : int,  
    "numMaxDelayRetries" : int,  
    "backoffFunction" : "linear/arithmetic/geometric/exponential"  
  },  
  "throttlePolicy" : {  
    "maxReceivesPerSecond" : int  
  }  
}
```

For more information about the SetSubscriptionAttribute action, go to [SetSubscriptionAttributes](#) in the *Amazon Simple Notification Service API Reference*.

SetTopicAttributes Delivery Policy JSON Format

If you send a request to the SetTopicAttributes action and set the AttributeName parameter to a value of DeliveryPolicy, the value of the AttributeValue parameter must be a valid JSON object. For example, the following example sets the delivery policy to 5 total retries.

```
http://sns.us-east-2.amazonaws.com/  
?Action=SetTopicAttributes  
&TopicArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic  
&AttributeName=DeliveryPolicy  
&AttributeValue={"http":{"defaultHealthyRetryPolicy":{"numRetries":5}}}  
...
```

Use the following JSON format for the value of the AttributeValue parameter.

```
{  
  "http" : {  
    "defaultHealthyRetryPolicy" : {  
      "minDelayTarget": int,  
      "maxDelayTarget": int,  
      "numRetries": int,  
      "numMaxDelayRetries": int,  
      "backoffFunction": "linear/arithmetic/geometric/exponential"  
    },  
    "disableSubscriptionOverrides" : Boolean,  
    "defaultThrottlePolicy" : {  
      "maxReceivesPerSecond" : int  
    }  
  }  
}
```

For more information about the SetTopicAttribute action, go to [SetTopicAttributes](#) in the *Amazon Simple Notification Service API Reference*.

Amazon SNS Large Payload and Raw Message Delivery

Amazon SNS and Amazon SQS let you send and receive large payloads (from 64 to 256 kilobytes in size).

Note

To send large payloads, you must use an AWS SDK that supports Signature Version 4.

To avoid having Amazon SQS and HTTP/S endpoints process the JSON formatting of messages, Amazon SNS also allows raw message delivery:

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

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

Enabling Raw Message Delivery Using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic subscribed an 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-ef12g3hi.j456` page, in the **Details** section, choose **Enable raw message delivery**.
6. Choose **Save changes**.

Amazon SNS Tags

To organize and identify your Amazon SNS resources (for example, for cost allocation or for organizing and discovering tagged topics), you can add metadata *tags* that identify a topic's purpose, owner, or environment—this is especially useful when you have many topics. For information about managing Amazon SNS topics using the AWS Management Console or the AWS SDK for Java (and the [TagResource](#), [UntagResource](#), and [ListTagsForResource](#) API actions), see the [Listing, Adding, and Removing Tags for an Amazon SNS Topic](#) (p. 16) tutorial.

Note

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

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 tag keys and values. For more information, see [Setting Up a Monthly Cost Allocation Report](#) in the *AWS Billing and Cost Management User Guide*.

Each tag consists of a key-value pair that you define. For example, you can easily identify your *production* and *testing* topics if you tag your topics as follows:

Topic	Key	Value
MyTopicA	TopicType	Production
MyTopicB	TopicType	Testing

Note

When you use tags, keep the following guidelines in mind:

- We don't recommend adding more than 50 tags to a topic.
- Tags don't have any semantic meaning. Amazon SNS interprets tags as character strings.
- Tags are case-sensitive.
- A new tag with a key identical to that of an existing tag overwrites the existing tag.
- Tagging actions are limited to 10 TPS per AWS account, per AWS region. If your application requires a higher throughput, [submit a request](#).

Using Amazon SNS for System-to-System Messaging

This section provides information about using Amazon SNS for system-to-system messaging with subscribers such as Lambda functions, Amazon SQS queues, HTTP/S endpoints, and AWS Event Fork Pipelines.

Topics

- [Using Amazon SNS for System-to-System Messaging with an AWS Lambda Function as a Subscriber \(p. 89\)](#)
- [Using Amazon SNS for System-to-System Messaging with an Amazon SQS Queue as a Subscriber \(p. 90\)](#)
- [Using Amazon SNS for System-to-System Messaging with an HTTP/S Endpoint as a Subscriber \(p. 103\)](#)
- [Using Amazon SNS for System-to-System Messaging with AWS Event Fork Pipelines as a Subscriber \(p. 114\)](#)

Using Amazon SNS for System-to-System Messaging with an AWS Lambda Function as a Subscriber

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.

In addition, Amazon SNS also supports message delivery status attributes for message notifications sent to Lambda endpoints. For more information, see [Amazon SNS Message Delivery Status \(p. 55\)](#).

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, see [Getting Started with AWS Lambda](#). For information about creating an Amazon SNS topic, see [Create a Topic](#).

Configuring Amazon SNS with Lambda Endpoints 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.
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 create a sample message history store using Amazon SNS, Lambda, and Amazon DynamoDB, see the AWS Mobile Development blog [Invoking AWS Lambda functions via Amazon SNS](#).

Using Amazon SNS for System-to-System Messaging with an Amazon SQS Queue as a Subscriber

[Amazon SNS](#) works closely with Amazon Simple Queue Service (Amazon SQS). Both services provide different benefits for developers. Amazon SNS allows applications to send time-critical messages to multiple subscribers through a “push” mechanism, eliminating the need to periodically check or “poll” for updates. Amazon SQS is a message queue service used by distributed applications to exchange messages through a polling model, and can be used to decouple sending and receiving components—without requiring each component to be concurrently available. Using Amazon SNS and Amazon SQS together, messages can be delivered to applications that require immediate notification of an event, and also persisted in an Amazon SQS queue for other applications to process at a later time.

When you subscribe an Amazon SQS queue to an Amazon SNS topic, you can publish a message to the topic and Amazon SNS sends an Amazon SQS message to the subscribed queue. The Amazon SQS message contains the subject and message that were published to the topic along with metadata about the message in a JSON document. The Amazon SQS message will look similar to the following JSON document.

```
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEnTrFPa3...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-
ab0e-4ec2-88e0-db410a0f2bee"
}
```

Important

Amazon SNS isn't currently compatible with Amazon SQS FIFO queues.

Instead of following the steps listed below, you can now subscribe an Amazon SQS queue to an Amazon SNS topic using the Amazon SQS console, which simplifies the process. For more information, see [Subscribe Queue to Amazon SNS Topic](#)

To enable an Amazon SNS topic to send messages to an Amazon SQS queue, 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. 91)
2. [Give `sqs:SendMessage` permission to the Amazon SNS topic so that it can send messages to the queue.](#) (p. 91)
3. [Subscribe the queue to the Amazon SNS topic.](#) (p. 92)
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. 93)
5. [Test it out by publishing a message to the topic and reading the message from the queue.](#) (p. 95)

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. 95).

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. 98).

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** tab, 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 **Topic Details** section, copy the **Topic ARN** value so that you can use it to give permission for the Amazon SNS topic to send messages to the queue.

Step 2: Give Permission to the Amazon SNS Topic to Send Messages to the Amazon SQS Queue

For an Amazon SNS topic to be able to send messages to a queue, you must set a policy on the queue that allows the Amazon SNS topic to perform the `sqs:SendMessage` action.

Before you subscribe a queue to a topic, you need a topic and a queue. If you haven't already created a topic or queue, create them now. For more information, see [Creating a Topic](#), and see [Creating a Queue](#) in the *Amazon Simple Queue Service Developer Guide*.

To set a policy on a queue, you can use the Amazon SQS console or the [SetQueueAttributes](#) API action. Before you start, make sure you have the ARN for the topic that you want to allow to send messages to the queue.

To set a `SendMessage` policy on a queue using the Amazon SQS console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Select the box for the queue whose policy you want to set, choose the **Permissions** tab, and then choose **Add a Permission**.
3. In the **Add a Permission** dialog box, select **Allow** for **Effect**, choose **Everybody (*)** for **Principal**, and then select **SendMessage** from the **Actions** drop-down.
4. Add a condition that allows the action for the topic. Choose **Add Conditions (optional)**, choose **ArnEquals** for **Condition**, choose **aws:SourceArn** for **Key**, and paste in the topic ARN for **Value**. Choose **Add Condition**. The new condition should appear at the bottom of the box (you may have to scroll down to see this).
5. Choose **Add Permission**.

If you wanted to create the policy document yourself, you would create a policy like the following. The policy allows `MyTopic` to send messages to `MyQueue`.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:MyTopic"
      }
    }
  }]
}
```

Step 3: Subscribe the Queue to the Amazon SNS Topic

To send messages to a queue through a topic, you must subscribe the queue to the Amazon SNS topic. You specify the queue by its ARN. To subscribe to a topic, you can use the Amazon SNS console, the [sns-subscribe](#) CLI command, or the [Subscribe](#) API action. Before you start, make sure you have the ARN for the queue that you want to subscribe.

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic.
4. On the **MyTopic** page, in the **Subscriptions** page, choose **Create subscription**.
5. On the **Create subscription** page, in the **Details** section, do the following:
 - a. Verify the **Topic ARN**.
 - b. For **Protocol**, choose **Amazon SQS**.

- c. For **Endpoint**, enter the ARN of an Amazon SQS queue.
- d. Choose **Create Subscription**.

When the subscription is confirmed, your new subscription's **Subscription ID** displays its subscription ID. If the owner of the queue creates the subscription, the subscription is automatically confirmed and the subscription should be active almost immediately.

Usually, you'll be subscribing your own queue to your own topic in your own account. However, you can also subscribe a queue from a different account to your topic. If the user who creates the subscription is not the owner of the queue (for example, if a user from account A subscribes a queue from account B to a topic in account A), the subscription must be confirmed. For more information about subscribing a queue from a different account and confirming the subscription, see [Sending Amazon SNS Messages to an Amazon SQS Queue in a Different Account](#) (p. 95).

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 SQS](#) (p. 202), and [Controlling User Access to Your AWS Account](#) 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. 93). 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. 94). If you want to give permissions to a topic or queue to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

You should use the first method for most cases (apply policies to groups and manage permissions for users by adding or removing the appropriate users to the groups). If you need to give permissions to a user in another account, you should use the second method.

Adding a Policy to an IAM User or Group

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sns:Publish` action on the topic `MyTopic`.

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

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on the queues `MyQueue1` and `MyQueue2`.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "sqs:ReceiveMessage",
      "sqs:DeleteMessage"
    ],
    "Resource": [
      "arn:aws:sns:us-east-2:123456789012:MyQueue1",
      "arn:aws:sns:us-east-2:123456789012:MyQueue2"
    ]
  }]
}
```

Adding a Policy to a Topic or Queue

The following example policies show how to give another account permissions to a topic and queue.

Note

When you give another AWS account access to a resource in your account, you are also giving IAM users who have admin-level access (wildcard access) permissions to that resource. All other IAM users in the other account are automatically denied access to your resource. If you want to give specific IAM users in that AWS account access to your resource, the account or an IAM user with admin-level access must delegate permissions for the resource to those IAM users. For more information about cross-account delegation, see [Enabling Cross-Account Access](#) in the *Using IAM Guide*.

If you added the following policy to a topic MyTopic in account 123456789012, you would give account 111122223333 permission to perform the `sns:Publish` action on that topic.

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

If you added the following policy to a queue MyQueue in account 123456789012, you would give account 111122223333 permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on that queue.

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

Step 5: Test the Topic's Queue Subscriptions

You can test a topic's queue subscriptions by publishing to the topic and viewing the message that the topic sends to the queue.

To publish to a topic using the Amazon SNS console

1. Using the credentials of the AWS account or IAM user with permission to publish to the topic, sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. On the navigation panel, choose the topic and choose **Publish to Topic**.
3. In the **Subject** box, enter a subject (for example, **Testing publish to queue**) in the **Message** box, enter some text (for example, **Hello world!**), and choose **Publish Message**. The following message appears: Your message has been successfully published.

To view the message from the topic using the Amazon SQS console

1. Using the credentials of the AWS account or IAM user with permission to view messages in the queue, sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Check the box for the queue that is subscribed to the topic.
3. From the **Queue Action** drop-down, choose **View/Delete Messages** and choose **Start Polling for Messages**. A message with a type of **Notification** appears.
4. In the **Body** column, choose **More Details**. The **Message Details** box contains a JSON document that contains the subject and message that you published to the topic. The message looks similar to the following JSON document.

```
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEnTrFPa3...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-db410a0f2bee"
}
```

5. Choose **Close**. You have successfully published to a topic that sends notification messages to a queue.

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 [With an Amazon SQS Queue as a Subscriber \(p. 90\)](#)). The only difference is how you handle subscription confirmation, and that depends on how you subscribe the queue to the topic.

Topics

- [Queue Owner Creates Subscription](#) (p. 96)
- [A User Who Does Not Own the Queue Creates Subscription](#) (p. 97)

Queue Owner Creates Subscription

When the queue owner creates a subscription, the subscription doesn't require confirmation. The queue starts 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**.
4. On the **Edit *MyTopic*** page, expand the **Access policy** section.
5. Enter the following policy:

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

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

6. Choose **Save changes**.

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

Step 2: To Add an Amazon SQS Queue Subscription to a Topic in Another AWS Account Using the AWS Management Console

Before you begin, make sure you have the ARNs for your topic and queue and that you have [given permission to the topic to send messages to the queue](#) (p. 91).

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

Note

- To be able to communicate with the service, the queue must have permissions for Amazon SNS.

- Because you are the owner of the queue, you don't have to confirm the subscription.

A User Who Does Not Own the Queue Creates Subscription

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

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

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

Note

Before you subscribe an endpoint to the topic, make sure that the queue can receive messages from the topic by setting the `sqs:SendMessage` permission for the queue. For more information, see [Give Permission to the Topic to Send Messages to the Queue \(p. 91\)](#).

To Confirm a Subscription Using the AWS Management Console

1. Sign in to the [Amazon SQS console](#).
2. Select the queue that has a pending subscription to the topic.
3. Choose **Queue Actions, View/Delete Messages** and then choose **Start Polling for Messages**.

A message with the subscription confirmation is received in the queue.

4. In the **Body** column, do the following:
 - a. Choose **More Details**.
 - b. In the **Message Details** dialog box, find and note the **SubscribeURL** value, for example:

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

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

An XML response is displayed, for example:

```
<ConfirmSubscriptionResponse>  
  <ConfirmSubscriptionResult>  
    <SubscriptionArn>arn:aws:sns:us-east-2:123456789012:MyTopic:1234a567-  
bc89-012d-3e45-6fg7h890123i</SubscriptionArn>  
  </ConfirmSubscriptionResult>  
  <ResponseMetadata>  
    <RequestId>abcd1efg-23hi-jkl4-m5no-p67q8rstuvw9</RequestId>  
  </ResponseMetadata>  
</ConfirmSubscriptionResponse>
```

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

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

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 can download this template (<https://s3.amazonaws.com//cloudformation-templates-us-east-1/SNSToSQS.template>) from the [AWS CloudFormation Sample Templates page](#).

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.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",

  "Description": "This Template creates an Amazon SNS topic that can send messages to two Amazon 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 Amazon 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 Amazon 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. You will be billed for the AWS resources used if you create a stack from this template.",

  "Parameters": {
    "MyPublishUserPassword": {
      "NoEcho": "true",
      "Type": "String",
      "Description": "Password for the IAM user MyPublishUser",
      "MinLength": "1",
      "MaxLength": "41",
```

```
        "AllowedPattern": "[a-zA-Z0-9]*",
        "ConstraintDescription": "must contain only alphanumeric characters."
    },
    "MyQueueUserPassword": {
        "NoEcho": "true",
        "Type": "String",
        "Description": "Password for the IAM user MyQueueUser",
        "MinLength": "1",
        "MaxLength": "41",
        "AllowedPattern": "[a-zA-Z0-9]*",
        "ConstraintDescription": "must contain only alphanumeric characters."
    }
},

"Resources": {
    "MySNSTopic": {
        "Type": "AWS::SNS::Topic",
        "Properties": {
            "Subscription": [{
                "Endpoint": {
                    "Fn::GetAtt": ["MyQueue1", "Arn"]
                },
                "Protocol": "sqs"
            },
            {
                "Endpoint": {
                    "Fn::GetAtt": ["MyQueue2", "Arn"]
                },
                "Protocol": "sqs"
            }
        ]
    },
    "MyQueue1": {
        "Type": "AWS::SQS::Queue"
    },
    "MyQueue2": {
        "Type": "AWS::SQS::Queue"
    },
    "MyPublishUser": {
        "Type": "AWS::IAM::User",
        "Properties": {
            "LoginProfile": {
                "Password": {
                    "Ref": "MyPublishUserPassword"
                }
            }
        }
    },
    "MyPublishUserKey": {
        "Type": "AWS::IAM::AccessKey",
        "Properties": {
            "UserName": {
                "Ref": "MyPublishUser"
            }
        }
    },
    "MyPublishTopicGroup": {
        "Type": "AWS::IAM::Group",
        "Properties": {
            "Policies": [{
                "PolicyName": "MyTopicGroupPolicy",
                "PolicyDocument": {
                    "Statement": [{
                        "Effect": "Allow",
```



```

        "Action": [
            "sns:Publish"
        ],
        "Resource": {
            "Ref": "MySNSTopic"
        }
    }
}
}
}
},
"AddUserToMyPublishTopicGroup": {
    "Type": "AWS::IAM::UserToGroupAddition",
    "Properties": {
        "GroupName": {
            "Ref": "MyPublishTopicGroup"
        },
        "Users": [{
            "Ref": "MyPublishUser"
        }]
    }
},
"MyQueueUser": {
    "Type": "AWS::IAM::User",
    "Properties": {
        "LoginProfile": {
            "Password": {
                "Ref": "MyQueueUserPassword"
            }
        }
    }
},
"MyQueueUserKey": {
    "Type": "AWS::IAM::AccessKey",
    "Properties": {
        "UserName": {
            "Ref": "MyQueueUser"
        }
    }
},
"MyRDMessageQueueGroup": {
    "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"]
                    }
                ]
            }
        }]
    }
},
"AddUserToMyQueueGroup": {
    "Type": "AWS::IAM::UserToGroupAddition",

```

```
"Properties": {
  "GroupName": {
    "Ref": "MyRDMessageQueueGroup"
  },
  "Users": [{
    "Ref": "MyQueueUser"
  }]
},
},
"MyQueuePolicy": {
  "Type": "AWS::SQS::QueuePolicy",
  "Properties": {
    "PolicyDocument": {
      "Statement": [{
        "Effect": "Allow",
        "Principal": "*",
        "Action": ["sqs:SendMessage"],
        "Resource": "*",
        "Condition": {
          "ArnEquals": {
            "aws:SourceArn": {
              "Ref": "MySNSTopic"
            }
          }
        }
      }]
    },
    "Queues": [{
      "Ref": "MyQueue1"
    }, {
      "Ref": "MyQueue2"
    }]
  }
},
},
"Outputs": {
  "MySNSTopicTopicARN": {
    "Value": {
      "Ref": "MySNSTopic"
    }
  },
  "MyQueue1Info": {
    "Value": {
      "Fn::Join": [
        " ",
        [
          "ARN:",
          {
            "Fn::GetAtt": ["MyQueue1", "Arn"]
          },
          "URL:",
          {
            "Ref": "MyQueue1"
          }
        ]
      ]
    }
  },
  "MyQueue2Info": {
    "Value": {
      "Fn::Join": [
        " ",
        [
          "ARN:",
          {
            "Fn::GetAtt": ["MyQueue2", "Arn"]
          }
        ]
      ]
    }
  }
}
```

```
        },
        "URL:",
        {
            "Ref": "MyQueue2"
        }
    ]
}
},
"MyPublishUserInfo": {
    "Value": {
        "Fn::Join": [
            " ",
            [
                "ARN:",
                {
                    "Fn::GetAtt": ["MyPublishUser", "Arn"]
                },
                "Access Key:",
                {
                    "Ref": "MyPublishUserKey"
                },
                "Secret Key:",
                {
                    "Fn::GetAtt": ["MyPublishUserKey", "SecretAccessKey"]
                }
            ]
        ]
    }
},
"MyQueueUserInfo": {
    "Value": {
        "Fn::Join": [
            " ",
            [
                "ARN:",
                {
                    "Fn::GetAtt": ["MyQueueUser", "Arn"]
                },
                "Access Key:",
                {
                    "Ref": "MyQueueUserKey"
                },
                "Secret Key:",
                {
                    "Fn::GetAtt": ["MyQueueUserKey", "SecretAccessKey"]
                }
            ]
        ]
    }
}
}
```

Using Amazon SNS for System-to-System Messaging with an HTTP/S Endpoint as a Subscriber

You can use [Amazon SNS](#) to send notification messages to one or more HTTP or HTTPS endpoints. When you subscribe an endpoint to a topic, you can publish a notification to the topic and Amazon SNS sends an HTTP POST request delivering the contents of the notification to the subscribed endpoint. When you subscribe the endpoint, you choose whether Amazon SNS uses HTTP or HTTPS to send the POST request to the endpoint. If you use HTTPS, then you can take advantage of the support in Amazon SNS for the following:

- **Server Name Indication (SNI)**—This allows Amazon SNS to support HTTPS endpoints that require SNI, such as a server requiring multiple certificates for hosting multiple domains. For more information about SNI, see [Server Name Indication](#).
- **Basic and Digest Access Authentication**—This allows you to specify a username and password in the HTTPS URL for the HTTP POST request, such as `https://user:password@domain.com` or `https://user@domain.com`. The username and password are encrypted over the SSL connection established when using HTTPS. Only the domain name is sent in plaintext. For more information about Basic and Digest Access Authentication, see [RFC-2617](#).

Note

The client service must be able to support the HTTP/1.1 401 Unauthorized header response

The request contains the subject and message that were published to the topic along with metadata about the notification in a JSON document. The request will look similar to the following HTTP POST request. For details about the HTTP header and the JSON format of the request body, see [HTTP/HTTPS Headers \(p. 81\)](#) and [HTTP/HTTPS Notification JSON Format \(p. 83\)](#).

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: da41e39f-ea4d-435a-b922-c6aae3915ebe
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfc21c8f55
Content-Length: 761
Content-Type: text/plain; charset=UTF-8
Host: ec2-50-17-44-49.compute-1.amazonaws.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "da41e39f-ea4d-435a-b922-c6aae3915ebe",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "test",
  "Message" : "test message",
  "Timestamp" : "2012-04-25T21:49:25.719Z",
  "SignatureVersion" : "1",
  "Signature" :
    "EXAMPLElDMXvB8r9R83tGoNn0ecwd5UjllzsvSvbItzfaMpN2nk5HVSsw7XnOn/49IkxDKz8Yr1H2qJXj2iZB0Zo2071c4qQk1fMUL
    f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
```

```
"UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfc21c8f55"
}
```

To enable an Amazon SNS topic to send messages to an HTTP or HTTPS endpoint, follow these steps:

[Make Sure Your Endpoint is Ready to Process Amazon SNS Messages \(p. 104\)](#)

[Subscribe the HTTP/HTTPS Endpoint to the Amazon SNS Topic \(p. 107\)](#)

[Step 3: Confirm the subscription \(p. 107\)](#)

[Step 4: Set the delivery retry policy for the subscription \(optional\) \(p. 107\)](#)

[Step 5: Give users permissions to publish to the topic \(optional\) \(p. 108\)](#)

[Step 6: Send messages to the HTTP/HTTPS endpoint \(p. 109\)](#)

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 \(p. 81\)](#). The following HTTP POST request is an example of a subscription confirmation message.

```
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "SubscriptionConfirmation",
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37f...",
}
```

```
"TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
"Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL
included in this message.",
"SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
west-2:123456789012:MyTopic&Token=2336412f37...",
"Timestamp" : "2012-04-26T20:45:04.751Z",
"SignatureVersion" : "1",
"Signature" : "EXAMPLEpH+...",
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

2. Your code should parse the JSON document in the body of the HTTP POST request to read the name-value pairs that make up the Amazon SNS message. Use a JSON parser that handles converting the escaped representation of control characters back to their ASCII character values (for example, converting `\n` to a newline character). You can use an existing JSON parser such as the [Jackson JSON Processor](#) or write your own. In order to send the text in the subject and message fields as valid JSON, Amazon SNS must convert some control characters to escaped representations that can be included in the JSON document. When you receive the JSON document in the body of the POST request sent to your endpoint, you must convert the escaped characters back to their original character values if you want an exact representation of the original subject and messages published to the topic. This is critical if you want to verify the signature of a notification because the signature uses the message and subject in their original forms as part of the string to sign.
3. Your code should verify the authenticity of a notification, subscription confirmation, or unsubscribe confirmation message sent by Amazon SNS. Using information contained in the Amazon SNS message, your endpoint can recreate the signature so that you can verify the contents of the message by matching your signature with the signature that Amazon SNS sent with the message. For more information about verifying the signature of a message, see [Verifying the Signatures of Amazon SNS Messages](#) (p. 109).
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. 81). When you visit the URL, you will get back a response that looks like the following XML document. The document returns the subscription ARN for the endpoint within the `ConfirmSubscriptionResult` element.

```
<ConfirmSubscriptionResponse xmlns="http://sns.amazonaws.com/doc/2010-03-31/">
  <ConfirmSubscriptionResult>
    <SubscriptionArn>arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55</SubscriptionArn>
  </ConfirmSubscriptionResult>
  <ResponseMetadata>
    <RequestId>075ecce8-8dac-11e1-bf80-f781d96e9307</RequestId>
  </ResponseMetadata>
</ConfirmSubscriptionResponse>
```

As an alternative to visiting the `SubscribeURL`, you can confirm the subscription using the [ConfirmSubscription](#) action with the `Token` set to its corresponding value in the `SubscriptionConfirmation` message. If you want to allow only the topic

owner and subscription owner to be able to unsubscribe the endpoint, you call the `ConfirmSubscription` action with an AWS signature.

Notification

Read the values for `Subject` and `Message` to get the notification information that was published to the topic.

For details about the format of the Notification message, see [HTTP/HTTPS Headers \(p. 81\)](#). The following HTTP POST request is an example of a notification message sent to the endpoint `example.com`.

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEw6JRN...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

5. Make sure that your endpoint responds to the HTTP POST message from Amazon SNS with the appropriate status code. The connection will time out in 15 seconds. If your endpoint does not respond before the connection times out or if your endpoint returns a status code outside the range of 200–4xx, Amazon SNS will consider the delivery of the message as a failed attempt.
6. Make sure that your code can handle message delivery retries from Amazon SNS. If Amazon SNS doesn't receive a successful response from your endpoint, it attempts to deliver the message again. This applies to all messages, including the subscription confirmation message. By default, if the initial delivery of the message fails, Amazon SNS attempts up to three retries with a delay between failed attempts set at 20 seconds.

Note

The message request times out after 15 seconds. This means that, if the message delivery failure is caused by a timeout, Amazon SNS retries for approximately 35 seconds after the previous delivery attempt. You can set a different delivery policy for the endpoint.

To be clear, Amazon SNS attempts to retry only after a delivery `x-amz-sns-message-id` header field. By comparing the IDs of the messages you have processed with incoming messages, you can determine whether the message is a retry attempt.

7. If you are subscribing an HTTPS endpoint, make sure that your endpoint has a server certificate from a trusted Certificate Authority (CA). Amazon SNS will only send messages to HTTPS endpoints that have a server certificate signed by a CA trusted by Amazon SNS.

8. Deploy the code that you have created to receive Amazon SNS messages. When you subscribe the endpoint, the endpoint must be ready to receive at least the subscription confirmation message.

Step 2: Subscribe the HTTP/HTTPS Endpoint to the Amazon SNS Topic

To send messages to an HTTP or HTTPS endpoint through a topic, you must subscribe the endpoint to the Amazon SNS topic. You specify the endpoint using its URL. To subscribe to a topic, you can use the Amazon SNS console, the `sns-subscribe` command, or the `Subscribe` API action. Before you start, make sure you have the URL for the endpoint that you want to subscribe and that your endpoint is prepared to receive the confirmation and notification messages as described in Step 1.

To subscribe an HTTP or HTTPS endpoint to a topic using the Amazon SNS console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Topics** and then choose the topic.
3. Choose the **Other actions** drop-down list and select **Subscribe to topic**.
4. In the **Protocol** drop-down list, select **HTTP** or **HTTPS**.
5. In the **Endpoint** box, paste in the URL for the endpoint that you want the topic to send messages to and then choose **Create subscription**.
6. For the **Subscription request received!** message, choose **Close**.

Your new subscription's **Subscription ID** displays PendingConfirmation. When you confirm the subscription, **Subscription ID** will display the subscription ID.

Step 3: Confirm the subscription

After you subscribe your endpoint, Amazon SNS will send a subscription confirmation message to the endpoint. You should already have code that performs the actions described in [Step 1 \(p. 104\)](#) 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 \(p. 104\)](#), 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 SQS \(p. 202\)](#).

There are two ways to control access to a topic:

- Add a policy to an IAM user or group. The simplest way to give users permissions to topics is to create a group and add the appropriate policy to the group and then add users to that group. It's much easier to add and remove users from a group than to keep track of which policies you set on individual users.
- Add a policy to the topic. If you want to give permissions to a topic to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

You should use the first method for most cases (apply policies to groups and manage permissions for users by adding or removing the appropriate users to the groups). If you need to give permissions to a user in another account, use the second method.

If you added the following policy to an IAM user or group, you would give that user or members of that group permission to perform the `sns:Publish` action on the topic `MyTopic`.

```
{
  "Statement": [{
    "Sid": "AllowPublishToMyTopic",
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
  }]
}
```

The following example policy shows how to give another account permissions to a topic.

Note

When you give another AWS account access to a resource in your account, you are also giving IAM users who have admin-level access (wildcard access) permissions to that resource. All other IAM users in the other account are automatically denied access to your resource. If you want to give specific IAM users in that AWS account access to your resource, the account or an IAM user with admin-level access must delegate permissions for the resource to those IAM users. For more information about cross-account delegation, see [Enabling Cross-Account Access](#) in the *Using IAM Guide*.

If you added the following policy to a topic `MyTopic` in account `123456789012`, you would give account `111122223333` permission to perform the `sns:Publish` action on that topic.

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

Step 6: Send messages to the HTTP/HTTPS endpoint

You can send a message to a topic's subscriptions by publishing to the topic. To publish to a topic, you can use the Amazon SNS console, the `sns-publish` CLI command, or the `Publish` API.

If you followed [Step 1 \(p. 104\)](#), the code that you deployed at your endpoint should process the notification.

To publish to a topic using the Amazon SNS console

1. Using the credentials of the AWS account or IAM user with permission to publish to the topic, sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. On the navigation panel, choose **Topics** and then choose a topic.
3. Choose the **Publish to topic** button.
4. In the **Subject** box, enter a subject (for example, **Testing publish to my endpoint**).
5. In the **Message** box, enter some text (for example, **Hello world!**), and choose **Publish message**.

The following message appears: Your message has been successfully published.

Verifying the Signatures of Amazon SNS Messages

You should verify the authenticity of a notification, subscription confirmation, or unsubscribe confirmation message sent by Amazon SNS. Using information contained in the Amazon SNS message, your endpoint can recreate the string to sign and the signature so that you can verify the contents of the message by matching the signature you recreated from the message contents with the signature that Amazon SNS sent with the message.

To help prevent spoofing attacks, you should do the following when verifying messages sent by Amazon SNS:

- Always use HTTPS when getting the certificate from Amazon SNS.
- Validate the authenticity of the certificate.
- Verify the certificate was received from Amazon SNS.
- When possible, use one of the supported AWS SDKs for Amazon SNS to validate and verify messages. For example, with the AWS SDK for PHP you would use the `isValid` method from the `MessageValidator` class.

For example code for a Java servlet that handles Amazon SNS messages, see [Example Code for an Amazon SNS Endpoint Java Servlet \(p. 111\)](#).

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 [Make Sure Your Endpoint is Ready to Process Amazon SNS Messages \(p. 104\)](#).

The `SignatureVersion` tells you the signature version. From the signature version, you can determine the requirements for how to generate the signature. For Amazon SNS notifications,

Amazon SNS currently supports signature version 1. This section provides the steps for creating a signature using signature version 1.

2. Get the X509 certificate that Amazon SNS used to sign the message. The `SigningCertURL` value points to the location of the X509 certificate used to create the digital signature for the message. Retrieve the certificate from this location.
3. Extract the public key from the certificate. The public key from the certificate specified by `SigningCertURL` is used to verify the authenticity and integrity of the message.
4. Determine the message type. The format of the string to sign depends on the message type, which is specified by the `Type` value.
5. Create the string to sign. The string to sign is a newline character–delimited list of specific name-value pairs from the message. Each name-value pair is represented with the name first followed by a newline character, followed by the value, and ending with a newline character. The name-value pairs must be listed in byte-sort order.

Depending on the message type, the string to sign must have the following name-value pairs.

Notification

Notification messages must contain the following name-value pairs:

```
Message
MessageId
Subject (if included in the message)
Timestamp
TopicArn
Type
```

The following example is a string to sign for a Notification.

```
Message
My Test Message
MessageId
4d4dc071-ddbf-465d-bba8-08f81c89da64
Subject
My subject
Timestamp
2019-01-31T04:37:04.321Z
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXC0P
Type
Notification
```

SubscriptionConfirmation and UnsubscribeConfirmation

SubscriptionConfirmation and UnsubscribeConfirmation messages must contain the following name-value pairs:

```
Message
MessageId
SubscribeURL
Timestamp
Token
TopicArn
Type
```

The following example is a string to sign for a SubscriptionConfirmation.

```
Message
```

```
My Test Message
MessageId
3d891288-136d-417f-bc05-901c108273ee
SubscribeURL
https://sns.us-east-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-2:123456789012:s4-
MySNSTopic-1G1WEFCOXCOP&Token=233...
Timestamp
2019-01-31T19:25:13.719Z
Token
233...
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXCOP
Type
SubscriptionConfirmation
```

6. Decode the *Signature* value from Base64 format. The message delivers the signature in the *Signature* value, which is encoded as Base64. Before you compare the signature value with the signature you have calculated, make sure that you decode the *Signature* value from Base64 so that you compare the values using the same format.
7. Generate the derived hash value of the Amazon SNS message. Submit the Amazon SNS message, in canonical format, to the same hash function used to generate the signature.
8. Generate the asserted hash value of the Amazon SNS message. The asserted hash value is the result of using the public key value (from step 3) to decrypt the signature delivered with the Amazon SNS message.
9. Verify the authenticity and integrity of the Amazon SNS message. Compare the derived hash value (from step 7) to the asserted hash value (from step 8). If the values are identical, then the receiver is assured that the message has not been modified while in transit and the message must have originated from Amazon SNS. If the values are not identical, it should not be trusted by the receiver.

Example Code for an Amazon SNS Endpoint Java Servlet

Important

The following code snippets help you understand a Java servlet that processes Amazon SNS HTTP POST requests. You should make sure that any portions of these snippets are suitable for your purposes before implementing them in your production environment. For example, in a production environment to help prevent spoofing attacks, you should verify that the identity of the received Amazon SNS messages is from Amazon SNS. You can do this by checking that the *DNS Name* value (*DNS Name=sns.us-west-2.amazonaws.com* in *us-west-2*; this will vary by Region) for the *Subject Alternative Name* field, as presented in the Amazon SNS Certificate, is the same for the received Amazon SNS messages. For more information about verifying server identity, see section [3.1. Server Identity in RFC 2818](#). Also see [Verifying the Signatures of Amazon SNS Messages \(p. 109\)](#)

The following method implements an example of a handler for HTTP POST requests from Amazon SNS in a Java servlet.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException, SecurityException {
    //Get the message type header.
    String messagetype = request.getHeader("x-amz-sns-message-type");
    //If message doesn't have the message type header, don't process it.
    if (messagetype == null) {
        return;
    }

    // Parse the JSON message in the message body
```

```
// and hydrate a Message object with its contents
// so that we have easy access to the name-value pairs
// from the JSON message.
Scanner scan = new Scanner(request.getInputStream());
StringBuilder builder = new StringBuilder();
while (scan.hasNextLine()) {
    builder.append(scan.nextLine());
}
Message msg = readMessageFromJson(builder.toString());

// The signature is based on SignatureVersion 1.
// If the sig version is something other than 1,
// throw an exception.
if (msg.getSignatureVersion().equals("1")) {
    // Check the signature and throw an exception if the signature verification fails.
    if (isMessageSignatureValid(msg)) {
        log.info(">>Signature verification succeeded");
    } else {
        log.info(">>Signature verification failed");
        throw new SecurityException("Signature verification failed.");
    }
} else {
    log.info(">>Unexpected signature version. Unable to verify signature.");
    throw new SecurityException("Unexpected signature version. Unable to verify
signature.");
}

// Process the message based on type.
if (messagetype.equals("Notification")) {
    //Do something with the Message and Subject.
    //Just log the subject (if it exists) and the message.
    String logMsgAndSubject = ">>Notification received from topic " +
msg.getTopicArn();
    if (msg.getSubject() != null) {
        logMsgAndSubject += " Subject: " + msg.getSubject();
    }
    logMsgAndSubject += " Message: " + msg.getMessage();
    log.info(logMsgAndSubject);
} else if (messagetype.equals("SubscriptionConfirmation")) {
    //You should make sure that this subscription is from the topic you expect. Compare
topicARN to your list of topics
    //that you want to enable to add this endpoint as a subscription.

    //Confirm the subscription by going to the subscribeURL location
    //and capture the return value (XML message body as a string)
    Scanner sc = new Scanner(new URL(msg.getSubscribeURL()).openStream());
    StringBuilder sb = new StringBuilder();
    while (sc.hasNextLine()) {
        sb.append(sc.nextLine());
    }
    log.info(">>Subscription confirmation (" + msg.getSubscribeURL() + ") Return value:
" + sb.toString());
    //Process the return value to ensure the endpoint is subscribed.
} else if (messagetype.equals("UnsubscribeConfirmation")) {
    //Handle UnsubscribeConfirmation message.
    //For example, take action if unsubscribing should not have occurred.
    //You can read the SubscribeURL from this message and
    //re-subscribe the endpoint.
    log.info(">>Unsubscribe confirmation: " + msg.getMessage());
} else {
    //Handle unknown message type.
    log.info(">>Unknown message type.");
}
log.info(">>Done processing message: " + msg.getMessageId());
}
```

The following example Java method creates a signature using information from a `Message` object that contains the data sent in the request body and verifies that signature against the original Base64-encoded signature of the message, which is also read from the `Message` object.

```
private static boolean isMessageSignatureValid(Message msg) {
    try {
        URL url = new URL(msg.getSigningCertURL());
        verifyMessageSignatureURL(msg, url);

        InputStream inStream = url.openStream();
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        X509Certificate cert = (X509Certificate) cf.generateCertificate(inStream);
        inStream.close();

        Signature sig = Signature.getInstance("SHA1withRSA");
        sig.initVerify(cert.getPublicKey());
        sig.update(getMessageBytesToSign(msg));
        return sig.verify(Base64.decodeBase64(msg.getSignature()));
    } catch (Exception e) {
        throw new SecurityException("Verify method failed.", e);
    }
}

private static void verifyMessageSignatureURL(Message msg, URL endpoint) {
    URI certUri = URI.create(msg.getSigningCertURL());

    if (!"https".equals(certUri.getScheme())) {
        throw new SecurityException("SigningCertURL was not using HTTPS: " +
            certUri.toString());
    }

    if (!endpoint.equals(certUri.getHost())) {
        throw new SecurityException(
            String.format("SigningCertUrl does not match expected endpoint. " +
                "Expected %s but received endpoint was %s.",
                endpoint, certUri.getHost()));
    }
}
```

The following example Java methods work together to create the string to sign for an Amazon SNS message. The `getMessageBytesToSign` method calls the appropriate string-to-sign method based on the message type and runs the string to sign as a byte array. The `buildNotificationStringToSign` and `buildSubscriptionStringToSign` methods create the string to sign based on the formats described in [Verifying the Signatures of Amazon SNS Messages \(p. 109\)](#).

```
private static byte [] getMessageBytesToSign (Message msg) {
    byte [] bytesToSign = null;
    if (msg.getType().equals("Notification"))
        bytesToSign = buildNotificationStringToSign(msg).getBytes();
    else if (msg.getType().equals("SubscriptionConfirmation") ||
        msg.getType().equals("UnsubscribeConfirmation"))
        bytesToSign = buildSubscriptionStringToSign(msg).getBytes();
    return bytesToSign;
}

//Build the string to sign for Notification messages.
public static String buildNotificationStringToSign(Message msg) {
    String stringToSign = null;

    //Build the string to sign from the values in the message.
    //Name and values separated by newline characters
    //The name value pairs are sorted by name
```

```
//in byte sort order.
stringToSign = "Message\n";
stringToSign += msg.getMessage() + "\n";
stringToSign += "MessageId\n";
stringToSign += msg.getMessageId() + "\n";
if (msg.getSubject() != null) {
    stringToSign += "Subject\n";
    stringToSign += msg.getSubject() + "\n";
}
stringToSign += "Timestamp\n";
stringToSign += msg.getTimestamp() + "\n";
stringToSign += "TopicArn\n";
stringToSign += msg.getTopicArn() + "\n";
stringToSign += "Type\n";
stringToSign += msg.getType() + "\n";
return stringToSign;
}

//Build the string to sign for SubscriptionConfirmation
//and UnsubscribeConfirmation messages.
public static String buildSubscriptionStringToSign(Message msg) {
    String stringToSign = null;
    //Build the string to sign from the values in the message.
    //Name and values separated by newline characters
    //The name value pairs are sorted by name
    //in byte sort order.
    stringToSign = "Message\n";
    stringToSign += msg.getMessage() + "\n";
    stringToSign += "MessageId\n";
    stringToSign += msg.getMessageId() + "\n";
    stringToSign += "SubscribeURL\n";
    stringToSign += msg.getSubscribeURL() + "\n";
    stringToSign += "Timestamp\n";
    stringToSign += msg.getTimestamp() + "\n";
    stringToSign += "Token\n";
    stringToSign += msg.getToken() + "\n";
    stringToSign += "TopicArn\n";
    stringToSign += msg.getTopicArn() + "\n";
    stringToSign += "Type\n";
    stringToSign += msg.getType() + "\n";
    return stringToSign;
}
```

Using Amazon SNS for System-to-System Messaging with AWS Event Fork Pipelines as a Subscriber

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. 41).

Topics

- [How AWS Event Fork Pipelines Works](#) (p. 115)
- [Deploying AWS Event Fork Pipelines](#) (p. 117)

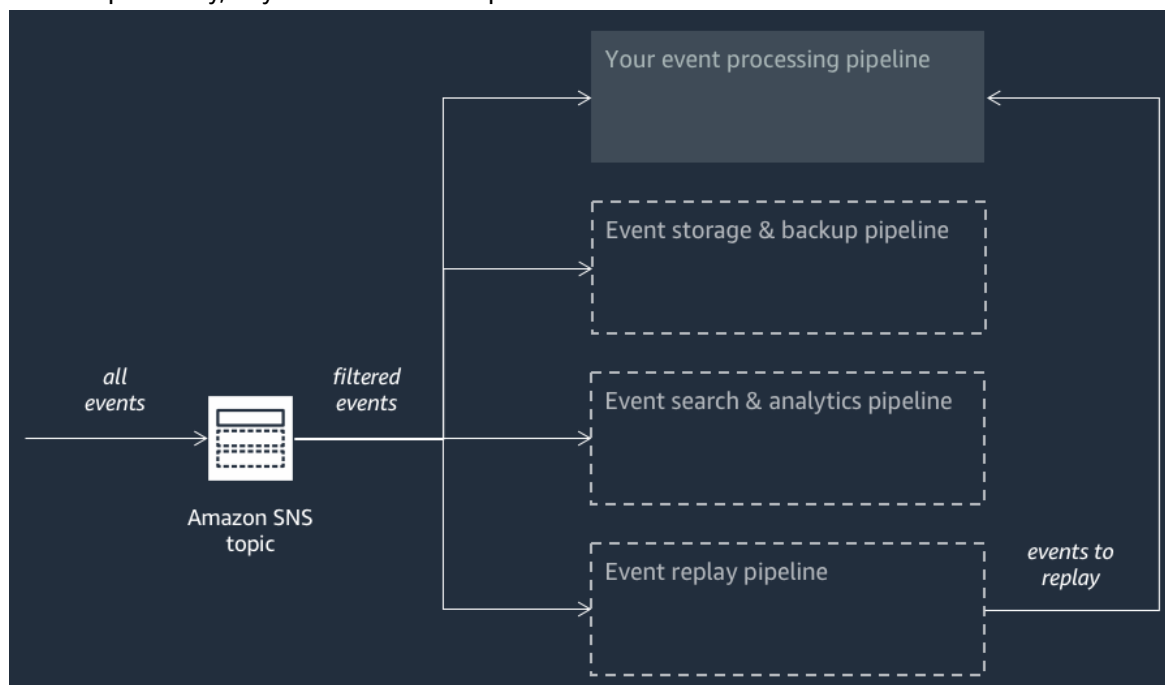
How AWS Event Fork Pipelines Works

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. 116)
- [The Event Search and Analytics Pipeline](#) (p. 116)
- [The Event Replay Pipeline](#) (p. 117)

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. 67). 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

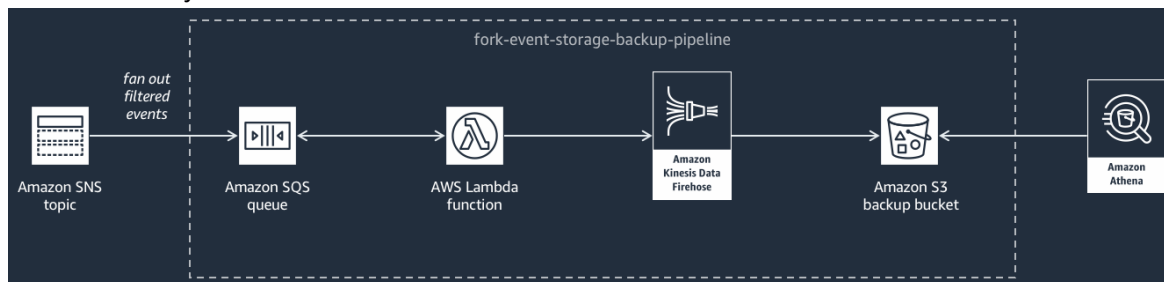
Because you place the three AWS Event Fork Pipelines alongside your regular event processing pipelines (possibly already subscribed to your Amazon SNS topic), you don't need to change any

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.

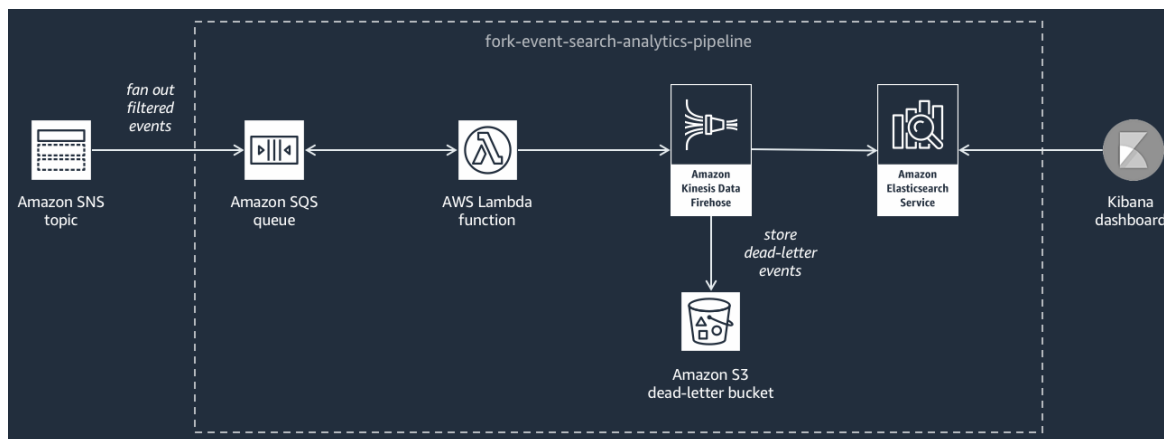


To fine-tune the behavior of your Firehose stream, you can configure it to buffer, transform, and compress your events prior to loading them into the bucket. As events are loaded, you can use Amazon Athena to query the bucket using standard SQL queries. You can also configure the pipeline to reuse an 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 Elasticsearch 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.



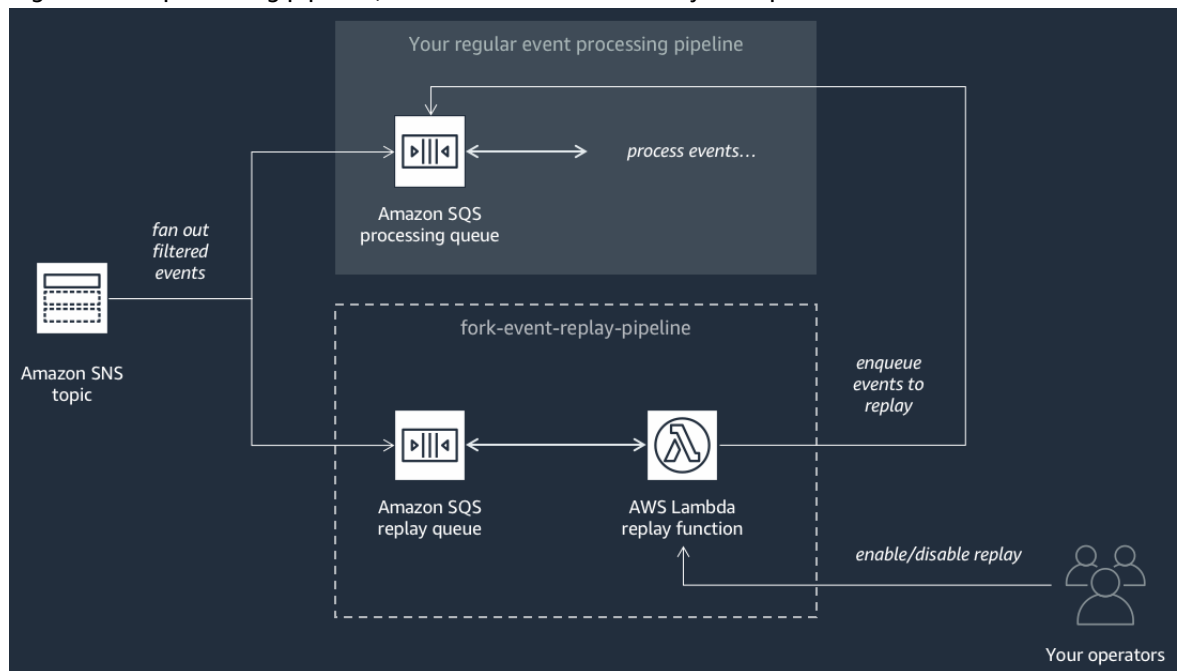
To fine-tune your Firehose stream in terms of event buffering, transformation, and compression, you can configure this pipeline.

You can also configure whether the pipeline should reuse an existing Elasticsearch 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 AWS Event Fork Pipelines to an Amazon SNS Topic](#) (p. 48).

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.

Using Amazon SNS for User Notifications

This section provides information about using Amazon SNS for user notifications with subscribers such as mobile applications, mobile phone numbers, and email addresses.

Topics

- [Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber \(Mobile Push\)](#) (p. 119)
- [Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber \(Send SMS\)](#) (p. 148)

Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber (Mobile Push)

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.

Important

Currently, you can create mobile applications only 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)

Topics

- [How User Notifications Work](#) (p. 120)
- [Prerequisites for Amazon SNS User Notifications](#) (p. 120)
- [User Notification Process Overview](#) (p. 121)
- [Using Amazon SNS Mobile Push](#) (p. 121)

- [Using Amazon SNS Application Attributes for Message Delivery Status \(p. 133\)](#)
- [Application Event Notifications \(p. 136\)](#)
- [Using the Amazon SNS Time To Live \(TTL\) Message Attribute for Mobile Push Notifications \(p. 138\)](#)
- [Using Amazon SNS Mobile Push APIs \(p. 140\)](#)
- [API Errors for Amazon SNS Mobile Push \(p. 142\)](#)

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 \(p. 121\)](#)

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 Simple Notification Service? \(p. 1\)](#). 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.

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 \(p. 121\)](#).

User Notification Process Overview

1. [Obtain the credentials and device token \(p. 120\)](#) 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 [Create a Platform Endpoint and Manage Device Tokens \(p. 124\)](#).
3. Use the returned credentials to request a device token for your mobile app and device from the mobile platforms. The token you receive represents your mobile app and device.
4. Use the device token and the `PlatformApplicationArn` to create a platform endpoint object (`EndpointArn`) using Amazon SNS. For more information, see [Create a Platform Endpoint and Manage Device Tokens \(p. 124\)](#).
5. Use the `EndpointArn` to [publish a message to an app on a mobile device \(p. 121\)](#). For more information, see [Send a Direct Message to a Mobile Device \(p. 131\)](#) and the [Publish](#) API in the Amazon Simple Notification Service API Reference.

Using Amazon SNS Mobile Push

This section describes how to use the AWS Management Console with the information described in [Prerequisites for Amazon SNS User Notifications \(p. 120\)](#) to register your mobile app with AWS, add device tokens (also referred to as registration IDs), send a direct message to a mobile device, and send a message to mobile devices subscribed to an Amazon SNS topic.

Topics

- [Register Your Mobile App with AWS \(p. 122\)](#)
- [Add Device Tokens or Registration IDs \(p. 122\)](#)
- [Create a Platform Endpoint and Manage Device Tokens \(p. 124\)](#)
- [Send a Direct Message to a Mobile Device \(p. 131\)](#)
- [Send Messages to Mobile Devices Subscribed to a Topic \(p. 131\)](#)
- [Send Custom Platform-Specific Payloads to Mobile Devices \(p. 131\)](#)

Register Your Mobile App with AWS

For Amazon SNS to send notification messages to mobile endpoints, whether it is direct or with subscriptions to a topic, you first need to register the app with AWS. To register your mobile app with AWS, enter a name to represent your app, choose the platform that will be supported, and provide your credentials for the notification service platform. After the app is registered with AWS, the next step is to create an endpoint for the app and mobile device. The endpoint is then used by Amazon SNS for sending notification messages to the app and device.

To register your mobile app with AWS

1. Sign in to the [Amazon SNS console](#).
2. Choose **Create platform application**.
3. In the **Application name** box, enter a name to represent your app.

App names must be made up of only uppercase and lowercase ASCII letters, numbers, underscores, hyphens, and periods, and must be between 1 and 256 characters long.

4. In the **Push notification platform** box, choose the platform that the app is registered with and then enter the appropriate credentials.

Note

If you are using one of the APNs platforms, then you can select **Choose file** to upload the .p12 file (exported from Keychain Access) to Amazon SNS.

5. After you have entered this information, then choose **Add New App**.

This registers the app with Amazon SNS, which creates a platform application object for the selected platform and then returns a corresponding PlatformApplicationArn.

Add Device Tokens or Registration IDs

When you first register an app and mobile device with a notification service, such as Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM), device tokens or registration IDs are returned from the notification service. When you add the device tokens or registration IDs to Amazon SNS, they are used with the PlatformApplicationArn API to create an endpoint for the app and device. When Amazon SNS creates the endpoint, an EndpointArn is returned. The EndpointArn is how Amazon SNS knows which app and mobile device to send the notification message to.

You can add device tokens and registration IDs to Amazon SNS using the following methods:

- Manually add a single token to AWS using the AWS Management Console
- Migrate existing tokens from a CSV file to AWS using the AWS Management Console
- Upload several tokens using the CreatePlatformEndpoint API
- Register tokens from devices that will install your apps in the future

To manually add a device token or registration ID

1. Sign in to the [Amazon SNS console](#).
2. Choose **Apps**, choose your app, and then choose **Add Endpoints**.
3. In the **Endpoint Token** box, enter either the token ID or registration ID, depending on which notification service. For example, with ADM and FCM you enter the registration ID.
4. (Optional) In the **User Data** box, enter arbitrary information to associate with the endpoint. Amazon SNS does not use this data. The data must be in UTF-8 format and less than 2KB.

5. Finally, choose **Add Endpoints**.

Now with the endpoint created, you can either send messages directly to a mobile device or send messages to mobile devices that are subscribed to a topic.

To migrate existing tokens from a CSV file to AWS

You can migrate existing tokens contained in a CSV file. The CSV file cannot be larger than 2MB. When migrating several tokens, it is recommended to use the `CreatePlatformEndpoint` API. Each of the tokens in the CSV file must be followed by a newline. For example, your CSV file should look similar to the following:

```
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWxwRkxMaDNST2luZz01,"User data
with spaces requires quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--
KMTlmMWxwRkxMaDNST2luZz04,"Data,with,commas,requires,quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWxwRkxMaDNST2luZz02,"Quoted data
requires ""escaped"" quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWxwRkxMaDNST2luZz03,"{"key":
  "json is allowed", "value":"endpoint", "number": 1}"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--
KMTlmMWxwRkxMaDNST2luZz05,SimpleDataNoQuotes
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWxwRkxMaDNST2luZz06,"The following
line has no user data"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWxwRkxMaDNST2luZz07
APBTkzPGLCyT6E6oOfpdwLpcRNxQp5vCPFiFeru9oZylc22HvZSwQTDgmmw9WdNlXMerUPxmpX0w1,"Different
token style"
```

1. Sign in to the [Amazon SNS console](#).
2. Choose **Apps**, choose your app, and then choose **Add Endpoints**.
3. Choose **Migrate existing tokens over to AWS**, choose **Choose File**, choose your CSV file, and then choose **Add Endpoints**.

To upload several tokens using the `CreatePlatformEndpoint` API

The following steps show how to use the sample Java app (bulkupload package) provided by AWS to upload several tokens (device tokens or registration IDs) to Amazon SNS. You can use this sample app to help you get started with uploading your existing tokens.

Note

The following steps use the Eclipse Java IDE. The steps assume you have installed the AWS SDK for Java and you have the AWS security credentials for your AWS account. For more information, see [AWS SDK for Java](#). For more information about credentials, see [How Do I Get Security Credentials?](#) in the *AWS General Reference*.

1. Download and unzip the [snsmobilepush.zip](#) file.
2. Create a new Java Project in Eclipse.
3. Import the `SNSSamples` folder to the top-level directory of the newly created Java Project. In Eclipse, right-choose the name of the Java Project and then choose **Import**, expand **General**, choose **File System**, choose **Next**, browse to the `SNSSamples` folder, choose **OK**, and then choose **Finish**.
4. Download a copy of the [OpenCSV library](#) and add it to the Build Path of the bulkupload package.
5. Open the `BulkUpload.properties` file contained in the bulkupload package.
6. Add the following to `BulkUpload.properties`:
 - The `ApplicationArn` to which you want to add endpoints.

- The absolute path for the location of your CSV file containing the tokens.
- The names for CSV files (such as `goodTokens.csv` and `badTokens.csv`) to be created for logging the tokens that Amazon SNS parses correctly and those that fail.
- (Optional) The characters to specify the delimiter and quote in the CSV file containing the tokens.
- (Optional) The number of threads to use to concurrently create endpoints. The default is 1 thread.

Your completed `BulkUpload.properties` should look similar to the following:

```
applicationarn:arn:aws:sns:us-west-2:111122223333:app/FCM/fcmpushapp
csvfilename:C:\\mytokendirectory\\mytokens.csv
goodfilename:C:\\mylogfiles\\goodtokens.csv
badfilename:C:\\mylogfiles\\badtokens.csv
delimiterchar:'
quotechar:"
numofthreads:5
```

7. Run the `BatchCreatePlatformEndpointSample.java` application to upload the tokens to Amazon SNS.

In this example, the endpoints that were created for the tokens that were uploaded successfully to Amazon SNS would be logged to `goodTokens.csv`, while the malformed tokens would be logged to `badTokens.csv`. In addition, you should see STD OUT logs written to the console of Eclipse, containing content similar to the following:

```
<1>[SUCCESS] The endpoint was created with Arn arn:aws:sns:us-west-2:111122223333:app/
FCM/fcmpushapp/165j2214-051z-3176-b586-138o3d420071
<2>[ERROR: MALFORMED CSV FILE] Null token found in /mytokendirectory/mytokens.csv
```

To register tokens from devices that will install your apps in the future

You can use one of the following two options:

- **Use the Amazon Cognito service:** Your mobile app will need credentials to create endpoints associated with your Amazon SNS platform application. We recommend that you use temporary credentials that expire after a period of time. For most scenarios, we recommend that you use Amazon Cognito to create temporary security credentials. For more information, see the [Amazon Cognito Developer Guide](#). If you would like to be notified when an app registers with Amazon SNS, you can register to receive an Amazon SNS event that will provide the new endpoint ARN. You can also use the `ListEndpointByPlatformApplication` API to obtain the full list of endpoints registered with Amazon SNS.
- **Use a proxy server:** If your application infrastructure is already set up for your mobile apps to call in and register on each installation, you can continue to use this setup. Your server will act as a proxy and pass the device token to Amazon SNS mobile push notifications, along with any user data you would like to store. For this purpose, the proxy server will connect to Amazon SNS using your AWS credentials and use the `CreatePlatformEndpoint` API call to upload the token information. The newly created endpoint Amazon Resource Name (ARN) will be returned, which your server can store for making subsequent publish calls to Amazon SNS.

Create a Platform Endpoint and Manage Device Tokens

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. 120) and [User Notification Process Overview](#) (p. 121).

This section describes the recommended approach for creating a platform endpoint and managing device tokens.

Topics

- [Create a Platform Endpoint \(p. 125\)](#)
- [Pseudo Code \(p. 125\)](#)
- [AWS SDK Examples \(p. 126\)](#)
- [Troubleshooting \(p. 130\)](#)

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 Examples

The following examples show how to implement the previous pseudo code using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

Here is an implementation of the previous pseudo code in Java:

```
class RegistrationExample {

    AmazonSNSClient client = new AmazonSNSClient(); //provide credentials here
    String arnStorage = null;

    public void registerWithSNS() {

        String endpointArn = retrieveEndpointArn();
        String token = "Retrieved from the mobile operating system";

        boolean updateNeeded = false;
        boolean createNeeded = (null == endpointArn);

        if (createNeeded) {
            // No platform endpoint ARN is stored; need to call createEndpoint.
            endpointArn = createEndpoint();
            createNeeded = false;
        }

        System.out.println("Retrieving platform endpoint data...");
        // Look up the platform endpoint and make sure the data in it is current, even if
        // it was just created.
        try {
            GetEndpointAttributesRequest geaReq =
                new GetEndpointAttributesRequest()
                    .withEndpointArn(endpointArn);
            GetEndpointAttributesResult geaRes =
                client.getEndpointAttributes(geaReq);

            updateNeeded = !geaRes.getAttributes().get("Token").equals(token)
        }
    }
}
```

```

        || !geaRes.getAttributes().get("Enabled").equalsIgnoreCase("true");
    } catch (NotFoundException nfe) {
        // We had a stored ARN, but the platform endpoint associated with it
        // disappeared. Recreate it.
        createNeeded = true;
    }

    if (createNeeded) {
        createEndpoint(token);
    }

    System.out.println("updateNeeded = " + updateNeeded);

    if (updateNeeded) {
        // The platform endpoint is out of sync with the current data;
        // update the token and enable it.
        System.out.println("Updating platform endpoint " + endpointArn);
        Map attribs = new HashMap();
        attribs.put("Token", token);
        attribs.put("Enabled", "true");
        SetEndpointAttributesRequest saeReq =
            new SetEndpointAttributesRequest()
                .withEndpointArn(endpointArn)
                .withAttributes(attribs);
        client.setEndpointAttributes(saeReq);
    }
}

/**
 * @return never null
 */
private String createEndpoint(String token) {

    String endpointArn = null;
    try {
        System.out.println("Creating platform endpoint with token " + token);
        CreatePlatformEndpointRequest cpeReq =
            new CreatePlatformEndpointRequest()
                .withPlatformApplicationArn(applicationArn)
                .withToken(token);
        CreatePlatformEndpointResult cpeRes = client
            .createPlatformEndpoint(cpeReq);
        endpointArn = cpeRes.getEndpointArn();
    } catch (InvalidParameterException ipe) {
        String message = ipe.getMessage();
        System.out.println("Exception message: " + message);
        Pattern p = Pattern
            .compile(".*Endpoint (arn:aws:sns:[^ ]+) already exists " +
                "with the same [Tt]oken.*");
        Matcher m = p.matcher(message);
        if (m.matches()) {
            // The platform endpoint already exists for this token, but with
            // additional custom data that
            // createEndpoint doesn't want to overwrite. Just use the
            // existing platform endpoint.
            endpointArn = m.group(1);
        } else {
            // Rethrow the exception, the input is actually bad.
            throw ipe;
        }
    }
    storeEndpointArn(endpointArn);
    return endpointArn;
}

```

```
/**
 * @return the ARN the app was registered under previously, or null if no
 *         platform endpoint ARN is stored.
 */
private String retrieveEndpointArn() {
    // Retrieve the platform endpoint ARN from permanent storage,
    // or return null if null is stored.
    return arnStorage;
}

/**
 * Stores the platform endpoint ARN in permanent storage for lookup next time.
 */
private void storeEndpointArn(String endpointArn) {
    // Write the platform endpoint ARN to permanent storage.
    arnStorage = endpointArn;
}
}
```

An interesting thing to note about this implementation is how the `InvalidParameterException` is handled in the `createEndpoint` method. Amazon SNS rejects create platform endpoint requests when an existing platform endpoint has the same device token and a non-null `CustomUserData` field, because the alternative is to overwrite (and therefore lose) the `CustomUserData`. The `createEndpoint` method in the preceding code captures the `InvalidParameterException` thrown by Amazon SNS, checks whether it was thrown for this particular reason, and if so, extracts the ARN of the existing platform endpoint from the exception. This succeeds, since a platform endpoint with the correct device token exists.

For more information, see [Using Amazon SNS Mobile Push APIs \(p. 140\)](#).

AWS SDK for .NET

Here is an implementation of the previous pseudo code in C#:

```
class RegistrationExample
{
    private AmazonSimpleNotificationServiceClient client = new
    AmazonSimpleNotificationServiceClient();
    private String arnStorage = null;

    public void RegisterWithSNS()
    {
        String endpointArn = EndpointArn;
        String token = "Retrieved from the mobile operating system";
        String applicationArn = "Set this based on your application";

        bool updateNeeded = false;
        bool createNeeded = (null == endpointArn);

        if (createNeeded)
        {
            // No platform endpoint ARN is stored; need to call createEndpoint.
            EndpointArn = CreateEndpoint(token, applicationArn);
            createNeeded = false;
        }

        Console.WriteLine("Retrieving platform endpoint data...");
        // Look up the platform endpoint and make sure the data in it is current, even
    if
        // it was just created.
        try
        {
            GetEndpointAttributesRequest geaReq = new GetEndpointAttributesRequest();
            geaReq.EndpointArn = EndpointArn;
```

```

        GetEndpointAttributesResponse geaRes =
client.GetEndpointAttributes(geaReq);
        updateNeeded = !(geaRes.Attributes["Token"] == token) || !
(geaRes.Attributes["Enabled"] == "true");
    }
    catch (NotFoundException)
    {
        // We had a stored ARN, but the platform endpoint associated with it
        // disappeared. Recreate it.
        createNeeded = true;
    }

    if (createNeeded)
    {
        CreateEndpoint(token, applicationArn);
    }

    Console.WriteLine("updateNeeded = " + updateNeeded);

    if (updateNeeded)
    {
        // The platform endpoint is out of sync with the current data;
        // update the token and enable it.
        Console.WriteLine("Updating platform endpoint " + endpointArn);
        Dictionary<String,String> attribs = new Dictionary<String,String>();
        attribs["Token"] = token;
        attribs["Enabled"] = "true";
        SetEndpointAttributesRequest saeReq = new SetEndpointAttributesRequest();
        saeReq.EndpointArn = EndpointArn;
        saeReq.Attributes = attribs;
        client.SetEndpointAttributes(saeReq);
    }
}

private String CreateEndpoint(String token, String applicationArn)
{
    String endpointArn = null;

    try
    {
        Console.WriteLine("Creating platform endpoint with token " + token);
        CreatePlatformEndpointRequest cpeReq = new CreatePlatformEndpointRequest();
        cpeReq.PlatformApplicationArn = applicationArn;
        cpeReq.Token = token;
        CreatePlatformEndpointResponse cpeRes =
client.CreatePlatformEndpoint(cpeReq);
        endpointArn = cpeRes.EndpointArn;
    }
    catch (InvalidParameterException ipe)
    {
        String message = ipe.Message;
        Console.WriteLine("Exception message: " + message);
        Regex rgx = new Regex(".*Endpoint (arn:aws:sns[^ ]+) already exists with
the same [Tt]oken.*",
            RegexOptions.IgnoreCase);
        MatchCollection m = rgx.Matches(message);
        if (m.Count > 0 && m[0].Groups.Count > 1)
        {
            // The platform endpoint already exists for this token, but with
            // additional custom data that createEndpoint doesn't want to
            overwrite.
            // Just use the existing platform endpoint.
            endpointArn = m[0].Groups[1].Value;
        }
        else
        {

```

```
        // Rethrow the exception, the input is actually bad.
        throw ipe;
    }
}
EndpointArn = endpointArn;
return endpointArn;
}

// Get/Set arn
public String EndpointArn
{
    get
    {
        return arnStorage;
    }
    set
    {
        arnStorage = value;
    }
}
}
```

For more information, see [Using Amazon SNS Mobile Push APIs \(p. 140\)](#).

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.

Send a Direct Message 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 **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.

Send Messages to Mobile Devices Subscribed 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 Simple Notification Service? \(p. 1\)](#). The difference is that Amazon SNS communicates through the notification services in order for the subscribed mobile endpoints to receive notifications sent to the topic.

Send Custom Platform-Specific Payloads to Mobile Devices

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 [Using Amazon SNS Mobile Push APIs \(p. 140\)](#) and the `SNSMobilePush.java` file in [snsmobilepush.zip](#).

Sending JSON-Formatted Messages

When you send platform-specific payloads, the data must be formatted as JSON key-value pair strings, with the quotation marks escaped.

The following examples show a custom message for the FCM platform.

```
{
  "GCM": "{\"data\":{\"message\":\"Check out these awesome deals!\",\"url\":\"www.amazon.com\"}}"
```


Sending Platform-Specific Messages

In addition to sending custom data as key-value pairs, you can send platform-specific key-value pairs.

The following example shows the inclusion of the FCM parameters `time_to_live` and `collapse_key` after the custom data key-value pairs in the FCM data parameter.

```
{
  "GCM": "{\"data\":{\"message\":\"Check out these awesome deals!\",\"url\":\"www.amazon.com\"},\"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 \(p. 65\)](#) using the Amazon SNS Publish API action, AWS SDKs, or the AWS CLI. The following CLI example sets `content-available` to 1 and `apns-push-type` to `background` for the specified topic.

```
aws sns publish \  
--endpoint-url https://sns.us-east-1.amazonaws.com \  
--target-arn arn:aws:sns:us-east-1:123456789012:endpoint/APNS_PLATFORM/MYAPP/1234a567-  
bc89-012d-3e45-6fg7h890123i \  
--message '{"APNS_PLATFORM":{"aps":{"content-available":1}}}' \  
--message-attributes '{ \  
  "AWS.SNS.MOBILE.APNS.TOPIC": \  
{ "DataType": "String", "StringValue": "com.amazon.mobile.messaging.myapp"}, \  
  "AWS.SNS.MOBILE.APNS.PRIORITY": { "DataType": "String", "StringValue": "10" } }, \  
  "AWS.SNS.MOBILE.APNS.PUSH_TYPE": { "DataType": "String", "StringValue": "background" } \  
--message-structure json
```

Inferring the APNs Push Type Header from the Payload

If you don't set the `apns-push-type` APNs header, Amazon SNS sets header to `alert` or `background` depending on the `content-available` key in the `aps` dictionary of your JSON-formatted APNs payload configuration.

Note

Amazon SNS is able to infer only `alert` or `background` headers, although the `apns-push-type` header can be set to other values.

- `apns-push-type` is set to `alert`
 - If the `aps` dictionary contains `content-available` set to 1 and *one or more keys* that trigger user interactions.
 - If the `aps` dictionary contains `content-available` set to 0 or if the `content-available` key is absent.
 - If the value of the `content-available` key isn't an integer or a Boolean.
- `apns-push-type` is set to `background`
 - If the `aps` dictionary *only* contains `content-available` set to 1 and *no other keys* that trigger user interactions.

Important

If Amazon SNS sends a raw configuration object for APNs as a background-only notification, you must include `content-available` set to 1 in the `aps` dictionary. Although you can include custom keys, the `aps` dictionary must not contain any keys that trigger user interactions (for example, alerts, badges, or sounds).

The following is an example raw configuration object.

```
{  
  "APNS": "{ \"aps\": { \"content-available\": 1, \"Foo1\": \"\\Bar\\\", \"Foo2\": 123 } }"
```

In this example, Amazon SNS sets the `apns-push-type` APNs header for the message to `background`. When Amazon SNS detects that the `apn` dictionary contains the `content-available` key set to 1—and doesn't contain any other keys that can trigger user interactions—it sets the header to `background`.

Using Amazon SNS Application Attributes for Message Delivery Status

Amazon Simple Notification Service (Amazon SNS) provides support to log the delivery status of push notification messages. After you configure application attributes, log entries will be sent to CloudWatch Logs for messages sent from Amazon SNS to mobile endpoints. Logging message delivery status helps provide better operational insight, such as the following:

- Know whether a push notification message was delivered from Amazon SNS to the push notification service.
- Identify the response sent from the push notification service to Amazon SNS.
- Determine the message dwell time (the time between the publish timestamp and just before handing off to a push notification service).

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

Topics

- [Configuring Message Delivery Status Attributes Using the AWS Management Console \(p. 134\)](#)
- [Amazon SNS Message Delivery Status CloudWatch Log Examples \(p. 134\)](#)
- [Configuring Message Delivery Status Attributes with the AWS SDKs \(p. 135\)](#)
- [Platform Response Codes \(p. 136\)](#)

Configuring Message Delivery Status Attributes Using the AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Apps**, and then choose the app containing the endpoints for which you want receive CloudWatch Logs.
3. Choose **Application Actions** and then choose **Delivery Status**.
4. On the **Delivery Status** dialog box, choose **Create IAM Roles**.

You will then be redirected to the IAM console.

5. Choose **Allow** to give Amazon SNS write access to use CloudWatch Logs on your behalf.
6. 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.

7. Finally, choose **Save Configuration**. You will now be able to view and parse the CloudWatch Logs containing the message delivery status. For more information about using CloudWatch, see the [CloudWatch Documentation](#).

Amazon SNS Message Delivery Status CloudWatch Log Examples

After you configure message delivery status attributes for an application endpoint, CloudWatch Logs will be generated. Example logs, in JSON format, are shown as follows:

SUCCESS

```
{
  "status": "SUCCESS",
  "notification": {
    "timestamp": "2015-01-26 23:07:39.54",
    "messageId": "9655abe4-6ed6-5734-89f7-e6a6a42de02a"
  },
  "delivery": {
```

```
"statusCode": 200,
"dwellTimeMs": 65,
"token": "Examplei7fFachkJ1xjlqT64RaBkcGHochmf1VQAr9k-
IBJtKjp7fedYPzEwT_Pq3Tu0lroqro1cwWJUvgkcPPYcaXCpPWmG3Bqn-
wiqIEzp5zZ7y_jsMOPKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw",
"attempts": 1,
"providerResponse": "{\"multicast_id\":\"5138139752481671853\",\"success\":1,\"failure\":0,
\"canonical_ids\":0,\"results\": [{\"message_id\":\"0:1422313659698010%d6ba8edff9fd7ecd
\"}]}",
"destination": "arn:aws:sns:us-east-2:111122223333:endpoint/FCM/FCMPushApp/
c23e42de-3699-3639-84dd-65f84474629d"
}
}
```

FAILURE

```
{
  "status": "FAILURE",
  "notification": {
    "timestamp": "2015-01-26 23:29:35.678",
    "messageId": "c3ad79b0-8996-550a-8bfa-24f05989898f"
  },
  "delivery": {
    "statusCode": 8,
    "dwellTimeMs": 1451,
    "token": "example29z6j5c4df46f80189c4c83fjcgf7f6257e98542d2jt3395kj73",
    "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 \(p. 136\)](#).

Configuring Message Delivery Status Attributes with the AWS SDKs

The [AWS SDKs](#) provide APIs in several languages for using message delivery status attributes with Amazon SNS.

The following Java example shows how to use the `SetPlatformApplicationAttributes` API to configure application attributes for message delivery status of push notification messages. You can use the following attributes for message delivery status: `SuccessFeedbackRoleArn`, `FailureFeedbackRoleArn`, and `SuccessFeedbackSampleRate`. The `SuccessFeedbackRoleArn` and `FailureFeedbackRoleArn` attributes are used to give Amazon SNS write access to use CloudWatch Logs on your behalf. The `SuccessFeedbackSampleRate` attribute is for specifying the sample rate percentage (0-100) of successfully delivered messages. After you configure the `FailureFeedbackRoleArn` attribute, then all failed message deliveries generate CloudWatch Logs.

```
SetPlatformApplicationAttributesRequest setPlatformApplicationAttributesRequest = new
SetPlatformApplicationAttributesRequest();
Map<String, String> attributes = new HashMap<>();
attributes.put("SuccessFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("FailureFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("SuccessFeedbackSampleRate", "5");
setPlatformApplicationAttributesRequest.withAttributes(attributes);
setPlatformApplicationAttributesRequest.setPlatformApplicationArn("arn:aws:sns:us-
west-2:111122223333:app/FCM/FCMPushApp");
```

```
sns.setPlatformApplicationAttributes(setPlatformApplicationAttributesRequest);
```

For more information about the SDK for Java, see [Getting Started with the AWS SDK for Java](#).

Platform Response Codes

The following is a list of links for the push notification service response codes:

Push Notification Service	Response Codes
Amazon Device Messaging (ADM)	See Response Format in the ADM documentation.
Apple Push Notification Service (APNs)	See <i>HTTP/2 Response from APNs</i> in the <i>Local and Remote Notification Programming Guide</i> .
Firebase Cloud Messaging (FCM)	See Downstream Message Error Response Codes in the Firebase Cloud Messaging documentation.
Microsoft Push Notification Service for Windows Phone (MPNS)	See Push Notification Service Response Codes for Windows Phone 8 in the Windows 8 Development documentation.
Windows Push Notification Services (WNS)	See "Response codes" in Push Notification Service Request and Response Headers (Windows Runtime Apps) in the Windows 8 Development documentation.

Application Event Notifications

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 \(p. 136\)](#)
- [Sending Mobile Push Notifications \(p. 137\)](#)

Available Application Events

Application event notifications track when individual platform endpoints are created, deleted, and updated, as well as delivery failures. The following are the attribute names for the application events.

Attribute Name	Notification Trigger
EventEndpointCreated	A new platform endpoint is added to your application.
EventEndpointDeleted	Any platform endpoint associated with your application is deleted.
EventEndpointUpdated	Any of the attributes of the platform endpoints associated with your application are changed.

Attribute Name	Notification Trigger
EventDeliveryFailure	<p>A delivery to any of the platform endpoints associated with your application encounters a permanent failure.</p> <p>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.</p>

You can associate any attribute with an application which can then receive these event notifications.

Sending Mobile Push Notifications

To send application event notifications, you specify a topic to receive the notifications for each type of event. As Amazon SNS sends the notifications, the topic can route them to endpoints that will take programmatic action.

Important

High-volume applications will create a large number of application event notifications (for example, tens of thousands), which will overwhelm endpoints meant for human use, such as email addresses, phone numbers, and mobile applications. Consider the following guidelines when you send application event notifications to a topic:

- Each topic that receives notifications should contain only subscriptions for programmatic endpoints, such as HTTP or HTTPS endpoints, Amazon SQS queues, or AWS Lambda functions.
- To reduce the amount of processing that is triggered by the notifications, limit each topic's subscriptions to a small number (for example, five or fewer).

You can send application event notifications using the Amazon SNS console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

AWS Management Console

1. Sign in to the [Amazon SNS console](#).
2. On the navigation panel, choose **Mobile, Push notifications**.
3. On the **Mobile push notifications** page, in the the **Platform applications** section, choose an application and then choose **Edit**.
4. Expand the **Event notifications** section.
5. Choose **Actions, Configure events**.
6. Enter the ARNs for topics to be used for the following events:
 - Endpoint Created
 - Endpoint Deleted
 - Endpoint Updated
 - Delivery Failure
7. Choose **Save changes**.

AWS CLI

Run the [set-platform-application-attributes](#) command.

The following example sets the same Amazon SNS topic for all four application events:

```
aws sns set-platform-application-attributes
--platform-application-arn arn:aws:sns:us-east-1:12345EXAMPLE:app/FCM/
MyFCMPlatformApplication
--attributes EventEndpointCreated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointDeleted="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointUpdated="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventDeliveryFailure="arn:aws:sns:us-east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents"
```

AWS SDKs

Call one of the following APIs, depending on your target programming language or platform:

Programming language or platform	API reference links
Android	setPlatformApplicationAttributes
iOS	AWSSNSSetPlatformApplicationAttributesInput
Java	setPlatformApplicationAttributes
JavaScript	setPlatformApplicationAttributes
.NET	SetPlatformApplicationAttributes
PHP	SetPlatformApplicationAttributes
Python (boto)	set_platform_application_attributes
Ruby	set_platform_application_attributes
Unity	SetPlatformApplicationAttributesAsync
Windows PowerShell	Set-SNSPlatformApplicationAttributes

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 \(p. 139\)](#)
- [Precedence Order for Determining TTL \(p. 139\)](#)

- [Specifying TTL Using the AWS Management Console \(p. 140\)](#)
- [Specifying TTL with the AWS SDKs \(p. 140\)](#)

TTL Message Attributes for Push Notification Services

The following is a list of the TTL message attributes for push notification services that you can use to set when using the AWS SDKs or query API:

Push Notification Service	TTL Message Attribute
Amazon Device Messaging (ADM)	<code>AWS.SNS.MOBILE.ADM.TTL</code>
Apple Push Notification Service (APNs)	<code>AWS.SNS.MOBILE.APNS.TTL</code>
Apple Push Notification Service Sandbox (APNs_SANDBOX)	<code>AWS.SNS.MOBILE.APNS_SANDBOX.TTL</code>
Baidu Cloud Push (Baidu)	<code>AWS.SNS.MOBILE.BAIDU.TTL</code>
Firebase Cloud Messaging (FCM)	<code>AWS.SNS.MOBILE.FCM.TTL</code>
Windows Push Notification Services (WNS)	<code>AWS.SNS.MOBILE.WNS.TTL</code>

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**.

Specifying TTL with the AWS SDKs

The [AWS SDKs](#) provide APIs in several languages for using TTL with Amazon SNS.

For more information about the SDK for Java, see [Getting Started with the AWS SDK for Java](#).

The following Java example shows how to configure a TTL message attribute and publish the message to an endpoint, which in this example is registered with Baidu Cloud Push:

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

// Insert your desired value (in seconds) of TTL here. For example, a TTL of 1 day would be
// 86,400 seconds.
messageAttributes.put("AWS.SNS.MOBILE.BAIDU.TTL", new
    MessageAttributeValue().withDataType("String").withStringValue("86400"));

PublishRequest publishRequest = new PublishRequest();
publishRequest.setMessageAttributes(messageAttributes);
String message = "{\"title\":\"Test_Title\",\"description\":\"Test_Description\"}";
publishRequest.setMessage(message);
publishRequest.setMessageStructure("json");
publishRequest.setTargetArn("arn:aws:sns:us-east-2:999999999999:endpoint/BAIDU/
    TestApp/318fc7b3-bc53-3d63-ac42-e359468ac730");
PublishResult publishResult = snsClient.publish(publishRequest);
```

For more information about using message attributes with Amazon SNS, see [Amazon SNS Message Attributes \(p. 64\)](#).

Using Amazon SNS Mobile Push APIs

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 \(p. 120\)](#).

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 \(p. 121\)](#).

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.

API Errors for Amazon SNS Mobile Push

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 [Using Amazon SNS Mobile Push APIs](#) (p. 140).

Error	Description	HTTPS Status Code	Action that Returns this Error
Application Name is null string	The required application name is set to null.	400	CreatePlatformApplication
Platform Name is null string	The required platform name is set to null.	400	CreatePlatformApplication
Platform Name is invalid	An invalid or out-of-range value was supplied for the platform name.	400	CreatePlatformApplication
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	CreatePlatformApplication
APNs — Principal is not provided	The APNs principal, which is the SSL certificate, was not provided.	400	CreatePlatformApplication
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see CreatePlatformApplication	400	CreatePlatformApplication

Error	Description	HTTPS Status Code	Action that Returns this Error
	in the Amazon Simple Notification Service API Reference.		
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	CreatePlatformApplication
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
FCM — serverAPIKey is empty	The FCM credential, which is the API key, is empty.	400	CreatePlatformApplication
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	CreatePlatformApplication
FCM — serverAPIKey is invalid	The FCM credential, which is the API key, is invalid.	400	CreatePlatformApplication
ADM — clientsecret is not provided	The required client secret is not provided.	400	CreatePlatformApplication
ADM — clientsecret is a null string	The required string for the client secret is null.	400	CreatePlatformApplication
ADM — client_secret is empty string	The required string for the client secret is empty.	400	CreatePlatformApplication
ADM — client_secret is not valid	The required string for the client secret is not valid.	400	CreatePlatformApplication
ADM — client_id is empty string	The required string for the client ID is empty.	400	CreatePlatformApplication
ADM — clientId is not provided	The required string for the client ID is not provided.	400	CreatePlatformApplication
ADM — clientid is a null string	The required string for the client ID is null.	400	CreatePlatformApplication
ADM — client_id is not valid	The required string for the client ID is not valid.	400	CreatePlatformApplication

Error	Description	HTTPS Status Code	Action that Returns this Error
EventEndpointCreated has invalid ARN format	EventEndpointCreated has invalid ARN format.	400	CreatePlatformApplication
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	CreatePlatformApplication
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	CreatePlatformApplication
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	CreatePlatformApplication
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	CreatePlatformApplication
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	CreatePlatformApplication
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure is not an existing topic.	400	CreatePlatformApplication
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	CreatePlatformApplication
Platform ARN is invalid	Platform ARN is invalid.	400	SetPlatformAttributes
Platform ARN is valid but does not belong to the user	Platform ARN is valid but does not belong to the user.	400	SetPlatformAttributes
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	SetPlatformAttributes
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes

Error	Description	HTTPS Status Code	Action that Returns this Error
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	SetPlatformAttributes
APNs — Principal is not provided	The APNs principal, which is the SSL certificate, was not provided.	400	SetPlatformAttributes
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	SetPlatformAttributes
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	SetPlatformAttributes
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	SetPlatformAttributes
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	SetPlatformAttributes
ADM — clientId is not provided	The required string for the client ID is not provided.	400	SetPlatformAttributes
ADM — clientId is a null string	The required string for the client ID is null.	400	SetPlatformAttributes
ADM — clientsecret is not provided	The required client secret is not provided.	400	SetPlatformAttributes
ADM — clientsecret is a null string	The required string for the client secret is null.	400	SetPlatformAttributes
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	SetPlatformAttributes
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	SetPlatformAttributes

Error	Description	HTTPS Status Code	Action that Returns this Error
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	SetPlatformAttributes
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure has invalid ARN format.	400	SetPlatformAttributes
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	SetPlatformAttributes
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	SetPlatformAttributes
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	SetPlatformAttributes
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	SetPlatformAttributes
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure is not an existing topic.	400	SetPlatformAttributes
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	SetPlatformAttributes
Platform ARN is invalid	The platform ARN is invalid.	400	GetPlatformApplicationAttributes
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	GetPlatformApplicationAttributes
Token specified is invalid	The specified token is invalid.	400	ListPlatformApplications
Platform ARN is invalid	The platform ARN is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	ListEndpointsByPlatformApplication
Token specified is invalid	The specified token is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	DeletePlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	DeletePlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	CreatePlatformEndpoint
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	CreatePlatformEndpoint

Error	Description	HTTPS Status Code	Action that Returns this Error
Token is not specified	The token is not specified.	400	CreatePlatformEndpoint
Token is not of correct length	The token is not the correct length.	400	CreatePlatformEndpoint
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	CreatePlatformEndpoint
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	DeleteEndpoint
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	DeleteEndpoint
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	SetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	SetEndpointAttributes
Token is not specified	The token is not specified.	400	SetEndpointAttributes
Token is not of correct length	The token is not the correct length.	400	SetEndpointAttributes
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	SetEndpointAttributes
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	GetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	GetEndpointAttributes
Target ARN is invalid	The target ARN is invalid.	400	Publish
Target ARN is valid but does not belong to the user	The target ARN is valid, but does not belong to the user.	403	Publish
Message format is invalid	The message format is invalid.	400	Publish
Message size is larger than supported by protocol/end-service	The message size is larger than supported by the protocol/end-service.	400	Publish

Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber (Send 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 \(p. 151\)](#), or you can [send a message to multiple phone numbers \(p. 155\)](#) at once by subscribing those phone numbers to a topic and sending your message to the topic.

You can [set SMS preferences \(p. 148\)](#) 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 \(p. 166\)](#), 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 \(p. 171\)](#).

Topics

- [Setting SMS Messaging Preferences \(p. 148\)](#)
- [Sending an SMS Message \(p. 151\)](#)
- [Sending an SMS Message to Multiple Phone Numbers \(p. 155\)](#)
- [Monitoring SMS Activity \(p. 161\)](#)
- [Managing Phone Numbers and SMS Subscriptions \(p. 166\)](#)
- [Reserving a Dedicated Short Code for SMS Messaging \(p. 170\)](#)
- [Supported Regions and Countries \(p. 171\)](#)

Setting SMS Messaging Preferences

Use Amazon SNS to specify preferences for SMS messaging, such as how your deliveries are optimized (for cost or for reliable delivery), your monthly spending limit, how message 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 [Sending an SMS Message \(p. 151\)](#).

Topics

- [Setting SMS Messaging Preferences Using the AWS Management Console \(p. 148\)](#)
- [Setting Preferences \(AWS SDKs\) \(p. 149\)](#)

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 \(p. 171\)](#).
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 AWS SDKs, use the action in that SDK that corresponds to the `SetSMSAttributes` request in the Amazon SNS API. With this request, you assign values to the different SMS attributes, such as your monthly spend quota and your default SMS type (promotional or transactional). For all SMS attributes, see [SetSMSAttributes](#) in the *Amazon Simple Notification Service API Reference*.

The following examples show how to set SMS preferences using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following example uses the `setSMSAttributes` method of the `AmazonSNSClient` class in the AWS SDK for Java. This examples sets values for the different attribute names:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    setDefaultSmsAttributes(snsClient);
}

public static void setDefaultSmsAttributes(AmazonSNSClient snsClient) {
```

```
SetSMSAttributesRequest setRequest = new SetSMSAttributesRequest()
    .addAttributesEntry("DefaultSenderId", "mySenderId")
    .addAttributesEntry("MonthlySpendLimit", "1")
    .addAttributesEntry("DeliveryStatusIAMRole",
        "arn:aws:iam::123456789012:role/mySnsRole")
    .addAttributesEntry("DeliveryStatusSuccessSamplingRate", "10")
    .addAttributesEntry("DefaultSMSType", "Transactional")
    .addAttributesEntry("UsageReportS3Bucket", "sns-sms-daily-usage");
snsClient.setSMSAttributes(setRequest);
Map<String, String> myAttributes = snsClient.getSMSAttributes(new
GetSMSAttributesRequest())
    .getAttributes();
System.out.println("My SMS attributes:");
for (String key : myAttributes.keySet()) {
    System.out.println(key + " = " + myAttributes.get(key));
}
}
```

This example sets the value for the `MonthlySpendLimit` attribute to 1.00 USD. By default, this is the maximum amount allowed by Amazon SNS. If you want to raise the quota, submit [submit a request](#). For **New limit value**, enter your desired monthly spend quota. In the **Use Case Description** field, explain that you are requesting an SMS monthly spend quota increase. The AWS Support team provides an initial response to your request within 24 hours.

To verify that the attributes were set correctly, the example prints the result of the `getSMSAttributes` method. When you run this example, the attributes are displayed in the console output window of your IDE:

```
My SMS attributes:
DeliveryStatusSuccessSamplingRate = 10
UsageReportS3Bucket = sns-sms-daily-usage
DefaultSMSType = Transactional
DeliveryStatusIAMRole = arn:aws:iam::123456789012:role/mySnsRole
MonthlySpendLimit = 1
DefaultSenderId = mySenderId
```

AWS SDK for .NET

The following example uses the `SetSMSAttributes` method of the `AmazonSimpleNotificationServiceClient` class in the AWS SDK for .NET. This example sets values for the different attribute names:

```
static void Main(string[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USWest2);
    SetDefaultSmsAttributes(snsClient);
}

public static void SetDefaultSmsAttributes(AmazonSimpleNotificationServiceClient
snsClient)
{
    SetSMSAttributesRequest setRequest = new SetSMSAttributesRequest();
    setRequest.Attributes["DefaultSenderId"] = "mySenderId";
    setRequest.Attributes["MonthlySpendLimit"] = "1";
    setRequest.Attributes["DeliveryStatusIAMRole"] = "arn:aws:iam::123456789012:role/
mySnsRole";
    setRequest.Attributes["DeliveryStatusSuccessSamplingRate"] = "10";
    setRequest.Attributes["DefaultSMSType"] = "Transactional";
    setRequest.Attributes["UsageReportS3Bucket"] = "sns-sms-daily-usage";
    SetSMSAttributesResponse setResponse = snsClient.SetSMSAttributes(setRequest);
    GetSMSAttributesRequest getRequest = new GetSMSAttributesRequest();
```

```
GetSMSAttributesResponse getResponse = snsClient.GetSMSAttributes(getRequest);
Console.WriteLine("My SMS attributes:");
foreach (var item in getResponse.Attributes)
{
    Console.WriteLine(item.Key + " = " + item.Value);
}
}
```

This example sets the value for the `MonthlySpendLimit` attribute to 1.00 USD. By default, this is the maximum amount allowed by Amazon SNS. If you want to raise the quota, [submit a case](#). For **New limit value**, enter your desired monthly spend quota. In the **Use Case Description** field, explain that you are requesting an SMS monthly spend quota increase. The AWS Support team provides an initial response to your request within 24 hours.

To verify that the attributes were set correctly, the example prints the result of the `GetSMSAttributes` method. When you run this example, the attributes are displayed in the console output window of your IDE:

```
My SMS attributes:
DeliveryStatusSuccessSamplingRate = 10
UsageReportS3Bucket = sns-sms-daily-usage
DefaultSMSType = Transactional
DeliveryStatusIAMRole = arn:aws:iam::123456789012:role/mySnsRole
MonthlySpendLimit = 1
DefaultSenderId = mySenderId
```

Sending an SMS Message

You can use Amazon SNS to send SMS messages to SMS-enabled devices. You can publish messages directly to the phone numbers for these devices, and you do not need to subscribe the phone numbers to an Amazon SNS topic.

Subscribing phone numbers to a topic can be still useful if you want to publish each message to multiple phone numbers at once. For steps on how to publish an SMS message to a topic, see [Sending an SMS Message to Multiple Phone Numbers \(p. 155\)](#).

When you send a message, you can control whether the message is optimized for cost or reliable delivery, and you can specify a sender ID. If you send the message programmatically using the Amazon SNS API or 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 on whole-word boundaries. The total size quota for a single SMS publish action is 1600 bytes.

When you send an SMS message, specify the phone number using the E.164 format. E.164 is a standard for the phone number structure used for international telecommunication. Phone numbers that follow this format can have a maximum of 15 digits, and they are prefixed with the plus character (+) and the country code. For example, a U.S. phone number in E.164 format would appear as +1XXX5550100.

Topics

- [Sending a Message \(Console\)](#) (p. 152)
- [Sending a Message \(AWS SDKs\)](#) (p. 152)

Sending a Message (Console)

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. 171).
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Text messaging (SMS)** page, choose **Send a text message (SMS)**. The **Send text message (SMS)** window opens.
5. For **Message type**, choose one of the following:
 - **Promotional** – Noncritical messages, such as marketing messages. Amazon SNS optimizes the 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 the message delivery to achieve the highest reliability.

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.
8. (Optional) For **Sender ID**, type a custom ID that contains up to 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. 171).

If you do not specify a sender ID, the message will display a long code as the sender ID in supported countries or regions. For countries and regions that require an alphabetic sender ID, the message displays *NOTICE* as the sender ID.

This message-level sender ID overrides your default sender ID, which you set on the **Text messaging preferences** page.

9. Choose **Send text message**.

Sending a Message (AWS SDKs)

To send an SMS message using one of the AWS SDKs, use the action 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 up to 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 \(p. 171\)](#).

If you do not specify a sender ID, the message will display a long code as the sender ID in supported countries and regions. For countries or regions that require an alphabetic sender ID, the message displays *NOTICE* as the sender ID.

This message-level attribute overrides the account-level attribute `DefaultSenderId`, which you set using the `SetSMSAttributes` request.

`AWS.SNS.SMS.MaxPrice`

The maximum amount in USD that you are willing to spend to send the SMS message. Amazon SNS will not send the message if it determines that doing so would incur a cost that exceeds the maximum price.

This attribute has no effect if your month-to-date SMS costs have already exceeded the quota set for the `MonthlySpendLimit` attribute, which you set using the `SetSMSAttributes` request.

If you are 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 are sending:

- **Promotional** (default) – Noncritical messages, such as marketing messages. Amazon SNS optimizes the 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 the message delivery to achieve the highest reliability.

This message-level attribute overrides the account-level attribute `DefaultSMSType`, which you set using the `SetSMSAttributes` request.

(Optional) Setting Message Attributes

The following examples show how to set message attributes using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

With the AWS SDK for Java, you set message attribute values by constructing a map that associates the attribute keys with `MessageAttributeValue` objects. Each `MessageAttributeValue` object is initialized with an attribute value, and each object declares the data type for the value. The following example sets the sender ID to "mySenderId", maximum price to 0.50 USD, and SMS type to promotional:

```
Map<String, MessageAttributeValue> smsAttributes =
    new HashMap<String, MessageAttributeValue>();
smsAttributes.put("AWS.SNS.SMS.SenderID", new MessageAttributeValue()
    .withStringValue("mySenderId") //The sender ID shown on the device.
    .withDataType("String"));
smsAttributes.put("AWS.SNS.SMS.MaxPrice", new MessageAttributeValue()
    .withStringValue("0.50") //Sets the max price to 0.50 USD.
    .withDataType("Number"));
smsAttributes.put("AWS.SNS.SMS.SMSType", new MessageAttributeValue()
    .withStringValue("Promotional") //Sets the type to promotional.
    .withDataType("String"));
```

When you send an SMS message, you will apply your attributes to the `PublishRequest` object.

AWS SDK for .NET

With the AWS SDK for .NET, you set message attribute values by constructing a map that associates the attribute keys with `MessageAttributeValue` objects. Each `MessageAttributeValue` object is initialized with an attribute value, and each object declares the data type for the value. The following example sets the sender ID to "mySenderId", maximum price to 0.50 USD, and SMS type to promotional:

```
PublishRequest pubRequest = new PublishRequest();
// add optional MessageAttributes...
pubRequest.MessageAttributes["AWS.SNS.SMS.SenderID"] =
    new MessageAttributeValue{ StringValue = "mySenderId", DataType = "String"};
pubRequest.MessageAttributes["AWS.SNS.SMS.MaxPrice"] =
    new MessageAttributeValue { StringValue = "0.50", DataType = "Number" };
pubRequest.MessageAttributes["AWS.SNS.SMS.SMSType"] =
    new MessageAttributeValue { StringValue = "Promotional", DataType = "String" };
```

Sending a Message

The following examples show how to send a message using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following example uses the `publish` method of the `AmazonSNSClient` class in the AWS SDK for Java. This example sends a message directly to a phone number:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    String message = "My SMS message";
    String phoneNumber = "+1XXX5550100";
    Map<String, MessageAttributeValue> smsAttributes =
        new HashMap<String, MessageAttributeValue>();
    //<set SMS attributes>
    sendSMSMessage(snsClient, message, phoneNumber, smsAttributes);
}

public static void sendSMSMessage(AmazonSNSClient snsClient, String message,
    String phoneNumber, Map<String, MessageAttributeValue> smsAttributes) {
    PublishResult result = snsClient.publish(new PublishRequest()
        .withMessage(message)
        .withPhoneNumber(phoneNumber)
        .withMessageAttributes(smsAttributes));
    System.out.println(result); // Prints the message ID.
}
```

When you run this example, the message ID is displayed in the console output window of your IDE:

```
{MessageId: 9b888f80-15f7-5c30-81a2-c4511a3f5229}
```

AWS SDK for .NET

The following example uses the `Publish` method of the `AmazonSimpleNotificationServiceClient` class in the AWS SDK for .NET. This example sends a message directly to a phone number:

```
static void Main(string[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USWest2);
```

```
PublishRequest pubRequest = new PublishRequest();
pubRequest.Message = "My SMS message";
pubRequest.PhoneNumber = "+1XXX5550100";
// add optional MessageAttributes, for example:
// pubRequest.MessageAttributes.Add("AWS.SNS.SMS.SenderID", new
MessageAttributeValue
// { StringValue = "SenderId", DataType = "String" });
PublishResponse pubResponse = snsClient.Publish(pubRequest);
Console.WriteLine(pubResponse.MessageId);
}
```

When you run this example, the message ID is displayed in the console output window of your IDE:

```
9b888f80-15f7-5c30-81a2-c4511a3f5229
```

Sending an SMS Message to Multiple Phone Numbers

You can publish a single SMS message to many phone numbers at once by subscribing those phone numbers to a topic. A topic is a communication channel to which you can add subscribers and then publish messages to all of those subscribers. A subscriber will receive all messages published to the topic until you cancel the subscription or the subscriber opts out of receiving SMS messages from your account.

Topics

- [Sending a Message to a Topic \(Console\) \(p. 155\)](#)
- [Sending a Message to a Topic \(AWS SDKs\) \(p. 156\)](#)

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, set the region selector to a [region that supports SMS messaging \(p. 171\)](#).
3. On the navigation panel, choose **Topics**.
4. On the **Topics** page, choose **Create new topic**. The **Create new topic** window opens.
5. For **Topic name**, type a name.
6. (Optional) For **Display name**, type 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 would appear as:

```
MYTOPIC> Hello World!
```

7. Choose **Create topic**. The topic name and Amazon Resource Name (ARN) are added to the table on the **Topics** page.

To add SMS subscriptions

Subscriptions enable you to send an SMS message to multiple recipients by publishing the message just once to your topic.

1. On the **Topics** page, choose the topic ARN.
2. On the topic details page, choose **Create Subscription**.
3. For **Protocol**, select **SMS**.
4. For **Endpoint**, type the phone number to which you want to send messages.
5. Choose **Create Subscription**. The subscription information is added to the **Subscriptions** table.

You can repeat these steps to add more phone numbers, and you can add other types of subscriptions, such as email.

To send the 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. On the topic details page, choose **Publish to topic**.
2. On the **Publish a message** page, for **Subject**, leave the field blank unless your topic contains email subscriptions and you want to publish to both email and SMS subscriptions. The text that you enter for **Subject** is used as the email subject line.
3. For **Message**, type a message.

For information about the size quotas for SMS messages, see [Sending an SMS Message \(p. 151\)](#).

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 space that Amazon SNS adds.

4. Choose **Publish message**. Amazon SNS sends the SMS message and displays a success message.

Sending a Message to a Topic (AWS SDKs)

To send an SMS message to a topic using one of AWS SDKs, use the actions in that SDK that correspond to the following requests in the Amazon SNS API:

CreateTopic

Creates a topic to which you can subscribe phone numbers and then publish messages to all of those phone numbers at once by publishing to the topic.

Subscribe

Subscribes a phone number to a topic.

Publish

Sends a message to each phone number subscribed to a topic.

You can use the `MessageAttributes` parameter to set several attributes for the message (for example, the maximum price). For more information, see [Sending a Message \(AWS SDKs\) \(p. 152\)](#).

Creating a Topic

The following examples show how to create a topic using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following example uses the `createTopic` method of the `AmazonSNSClient` class in the AWS SDK for Java:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    String topicArn = createSNSTopic(snsClient);
}

public static String createSNSTopic(AmazonSNSClient snsClient) {
    CreateTopicRequest createTopic = new CreateTopicRequest("mySNSTopic");
    CreateTopicResult result = snsClient.createTopic(createTopic);
    System.out.println("Create topic request: " +
        snsClient.getCachedResponseMetadata(createTopic));
    System.out.println("Create topic result: " + result);
    return result.getTopicArn();
}
```

The example uses the `getCachedResponseMetadata` method to get the request ID.

When you run this example, the following is displayed in the console output window of your IDE:

```
{TopicArn: arn:aws:sns:us-east-1:123456789012:mySNSTopic}
CreateTopicRequest - {AWS_REQUEST_ID=93f7fc90-f131-5ca3-ab18-b741fef918b5}
```

AWS SDK for .NET

The following example uses the `createTopic` method of the `AmazonSimpleNotificationServiceClient` class in the AWS SDK for .NET:

```
static void Main(string[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USSouth2);
    String topicArn = CreateSNSTopic(snsClient);
}

public static String CreateSNSTopic(AmazonSimpleNotificationServiceClient snsClient)
{
    //create a new SNS topic
    CreateTopicRequest createTopicRequest = new CreateTopicRequest("MyNewTopic200");
    CreateTopicResponse createTopicResponse =
    snsClient.CreateTopic(createTopicRequest);
    //get request id for CreateTopicRequest from SNS metadata
    Console.WriteLine("CreateTopicRequest - " +
    createTopicResponse.ResponseMetadata.RequestId);
    return createTopicResponse.TopicArn;
}
```

When you run this example, the following is displayed in the console output window of your IDE:

```
{TopicArn: arn:aws:sns:us-east-1:123456789012:mySNSTopic}
CreateTopicRequest - 93f7fc90-f131-5ca3-ab18-b741fef918b5
```

Adding an SMS Subscription to Your Topic

The following examples show how to add an SMS subscription to a topic using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following example uses the `subscribe` method of the `AmazonSNSClient` class in the AWS SDK for Java:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    String phoneNumber = "+1XXX5550100";
    String topicArn = createSNSTopic(snsClient);
    subscribeToTopic(snsClient, topicArn, "sms", phoneNumber);
}

//<create SNS topic>

public static void subscribeToTopic(AmazonSNSClient snsClient, String topicArn,
    String protocol, String endpoint) {
    SubscribeRequest subscribe = new SubscribeRequest(topicArn, protocol,
        endpoint);
    SubscribeResult subscribeResult = snsClient.subscribe(subscribe);
    System.out.println("Subscribe request: " +
        snsClient.getCacheResponseMetadata(subscribe));
    System.out.println("Subscribe result: " + subscribeResult);
}
```

This example constructs the `subscribeRequest` object and passes it the following arguments:

- `topicArn` - The Amazon Resource Name (ARN) of the topic to which you are adding a subscription.
- `"sms"` - The protocol option for an SMS subscription.
- `endpoint` - The phone number that you are subscribing to the topic.

The example uses the `getCacheResponseMetadata` method to get the request ID for the subscribe request.

When you run this example, the ID of the subscribe request is displayed in the console window of your IDE:

```
SubscribeRequest - {AWS_REQUEST_ID=f38fe925-8093-5bd4-9c19-a7c7625de38c}
```

AWS SDK for .NET

The following example uses the `subscribe` method of the `AmazonSimpleNotificationServiceClient` class in the AWS SDK for .NET:

```
static void Main(string[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USSouth1);
    String phoneNumber = "+1XXX5550100";
    String topicArn = CreateSNSTopic(snsClient);
    SubscribeToTopic(snsClient, topicArn, "sms", phoneNumber);
}

//<create SNS topic>

static public void SubscribeToTopic(AmazonSimpleNotificationServiceClient snsClient,
    String topicArn,
    String protocol, String endpoint)
{
    SubscribeRequest subscribeRequest = new SubscribeRequest(topicArn, protocol,
    endpoint);
    SubscribeResponse subscribeResponse = snsClient.Subscribe(subscribeRequest);
    Console.WriteLine("Subscribe request: " +
    subscribeResponse.ResponseMetadata.RequestId);
    Console.WriteLine("Subscribe result: " + subscribeResponse.);
}
```

```
}
```

This example constructs the `SubscribeRequest` object and passes it the following arguments:

- `topicArn` - The Amazon Resource Name (ARN) of the topic to which you are adding a subscription.
- `"sms"` - The protocol option for an SMS subscription.
- `endpoint` - The phone number that you are subscribing to the topic.

When you run this example, the ID of the subscribe request is displayed in the console window of your IDE:

```
SubscribeRequest - f38fe925-8093-5bd4-9c19-a7c7625de38c
```

(Optional) Setting Message Attributes

The following examples show how to set message attributes using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

With the AWS SDK for Java, you set message attribute values by constructing a map that associates the attribute keys with `MessageAttributeValue` objects. Each `MessageAttributeValue` object is initialized with an attribute value, and each object declares the data type for the value. The following example sets the sender ID to "mySenderId", maximum price to 0.50 USD, and SMS type to promotional:

```
Map<String, MessageAttributeValue> smsAttributes =
    new HashMap<String, MessageAttributeValue>();
smsAttributes.put("AWS.SNS.SMS.SenderID", new MessageAttributeValue()
    .withStringValue("mySenderId") //The sender ID shown on the device.
    .withDataType("String"));
smsAttributes.put("AWS.SNS.SMS.MaxPrice", new MessageAttributeValue()
    .withStringValue("0.50") //Sets the max price to 0.50 USD.
    .withDataType("Number"));
smsAttributes.put("AWS.SNS.SMS.SMSType", new MessageAttributeValue()
    .withStringValue("Promotional") //Sets the type to promotional.
    .withDataType("String"));
```

For more information about message attributes, see [Sending a Message \(AWS SDKs\) \(p. 152\)](#)

When you send an SMS message, you will apply your attributes to the `PublishRequest` object.

AWS SDK for .NET

With the AWS SDK for .NET, you set message attribute values by adding attribute keys with `MessageAttributeValue` objects to the `MessageAttributes` field of the `PublishRequest` object. Each `MessageAttributeValue` object is initialized with an attribute value, and each object declares the data type for the value. The following example sets the sender ID to "mySenderId", maximum price to 0.50 USD, and SMS type to promotional:

```
PublishRequest pubRequest = new PublishRequest();
pubRequest.MessageAttributes["AWS.SNS.SMS.SenderID"] =
    new MessageAttributeValue { StringValue = "mySenderId", DataType = "String" };
pubRequest.MessageAttributes["AWS.SNS.SMS.MaxPrice"] =
    new MessageAttributeValue { StringValue = "0.50", DataType = "Number" };
pubRequest.MessageAttributes["AWS.SNS.SMS.SMSType"] =
```

```
new MessageAttributeValue { StringValue = "Promotional", DataType = "String" };
```

For more information about message attributes, see [Sending a Message \(AWS SDKs\) \(p. 152\)](#)

When you send an SMS message, you will apply your attributes to the `PublishRequest` object.

Publishing a Message to Your Topic

The following examples show how to publish a message to a topic using the Amazon SNS clients that are provided by the AWS SDKs.

AWS SDK for Java

The following example uses the `publish` method of the `AmazonSNSClient` class in the AWS SDK for Java:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    String message = "My SMS message";
    Map<String, MessageAttributeValue> smsAttributes =
        new HashMap<String, MessageAttributeValue>();
    //<set SMS attributes>
    String topicArn = createSNSTopic(snsClient);
    //<subscribe to topic>
    sendSMSMessageToTopic(snsClient, topicArn, message, smsAttributes);
}

//<create topic method>

//<subscribe to topic method>

public static void sendSMSMessageToTopic(AmazonSNSClient snsClient, String topicArn,
    String message, Map<String, MessageAttributeValue> smsAttributes) {
    PublishResult result = snsClient.publish(new PublishRequest()
        .withTopicArn(topicArn)
        .withMessage(message)
        .withMessageAttributes(smsAttributes));
    System.out.println(result);
}
```

Amazon SNS will attempt to deliver that message to every phone number that is subscribed to the topic.

This example constructs the `publishRequest` object while passing the topic Amazon Resource Name (ARN) and the message as arguments. The `publishResult` object captures the message ID returned by Amazon SNS.

When you run this example, the message ID is displayed in the console output window of your IDE:

```
{MessageId: 9b888f80-15f7-5c30-81a2-c4511a3f5229}
```

AWS SDK for .NET

The following example uses the `Publish` method of the `AmazonSimpleNotificationServiceClient` class in the AWS SDK for .NET:

```
public static void main(string[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USSouth2);
```

```
String topicArn = createSNSTopic(snsClient);
PublishRequest pubRequest = new PublishRequest();
pubRequest.Message = "My SMS message";
pubRequest.TopicArn = topicArn;
// add optional MessageAttributes...
// pubRequest.MessageAttributes["AWS.SNS.SMS.SenderID"] =
//     new MessageAttributeValue{ StringValue = "mySenderId", DataType =
"String"};
PublishResponse pubResponse = snsClient.Publish(pubRequest);
Console.WriteLine(pubResponse.MessageId);
}

//<create topic method>

//<subscribe to topic method>
```

Amazon SNS will attempt to deliver that message to every phone number that is subscribed to the topic.

This example constructs the `publishRequest` object and assigns the topic Amazon Resource Name (ARN) and the message. The `publishResponse` object captures the message ID returned by Amazon SNS.

When you run this example, the message ID is displayed in the console output window of your IDE:

```
9b888f80-15f7-5c30-81a2-c4511a3f5229
```

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 \(p. 161\)](#)
- [Viewing Amazon CloudWatch Metrics and Logs for SMS Deliveries \(p. 162\)](#)
- [Viewing Daily SMS Usage Reports \(p. 164\)](#)

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 \(p. 171\)](#).
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 \(p. 164\)](#).

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 \(p. 162\)](#)
- [Viewing CloudWatch Logs \(p. 162\)](#)
- [Example Log for Successful SMS Delivery \(p. 163\)](#)
- [Example Log for Failed SMS Delivery \(p. 163\)](#)
- [SMS Delivery Failure Reasons \(p. 163\)](#)

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 \(p. 213\)](#).

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 will write 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 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 \(p. 171\)](#).
 3. On the navigation panel, choose **Text messaging (SMS)**.
 4. On the **Text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
 5. On the **Edit text messaging preferences** page, in the **Delivery status logging** section, do the following:
 - a. 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.
- b. Choose **Create new service role**.
 - c. Choose **Create new roles**.
 - d. On the **SNS is requesting permission to use resources in your account** page, choose **Allow**.
6. Choose **Save changes**.

Example Log for Successful SMS Delivery

The delivery status log for a successful SMS delivery will resemble the following example:

```
{
  "notification": {
    "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
    "timestamp": "2016-06-28 00:40:34.558"
  },
  "delivery": {
    "phoneCarrier": "My Phone Carrier",
    "mnc": 270,
    "destination": "+1XXX5550100",
    "priceInUSD": 0.00645,
    "smsType": "Transactional",
    "mcc": 310,
    "providerResponse": "Message has been accepted by phone carrier",
    "dwellTimeMs": 599,
    "dwellTimeMsUntilDeviceAck": 1344
  },
  "status": "SUCCESS"
}
```

Example Log for Failed SMS Delivery

The delivery status log for a failed SMS delivery will resemble the following example:

```
{
  "notification": {
    "messageId": "1077257a-92f3-5ca3-bc97-6a915b310625",
    "timestamp": "2016-06-28 00:40:34.559"
  },
  "delivery": {
    "mnc": 0,
    "destination": "+1XXX5550100",
    "priceInUSD": 0.00645,
    "smsType": "Transactional",
    "mcc": 0,
    "providerResponse": "Unknown error attempting to reach phone",
    "dwellTimeMs": 1420,
    "dwellTimeMsUntilDeviceAck": 1692
  },
  "status": "FAILURE"
}
```

SMS Delivery Failure Reasons

The reason for a failure is provided with the `providerResponse` attribute. SMS messages might fail to deliver for the following reasons:

- Blocked as spam by phone carrier
- Destination is blacklisted
- 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 blacklisted
- 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 \(p. 164\)](#)
- [Subscribing to Daily Usage Reports \(p. 164\)](#)

Daily Usage Report Information

The usage report includes the following information for each SMS message that you send from your account.

Note

The report does not include messages that are sent to recipients who have opted out.

- Time of publication for message (in UTC)
- Message ID
- Destination phone number
- Message type
- Delivery status
- Message price (in USD)
- Part number (a message is split into multiple parts if it is too long for a single message)
- Total number of parts

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. Sign in to the [Amazon S3 console](#).
2. Choose **Create Bucket**.
3. For **Bucket Name**, type a name, such as `sns-sms-daily-usage`. For information about conventions and restrictions for bucket names, see [Rules for Bucket Naming](#) in the *Amazon Simple Storage Service Developer Guide*.
4. Choose **Create**.
5. In the **All Buckets** table, choose the bucket and choose **Properties**.
6. In the **Permissions** section, choose **Add 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 \(p. 165\)](#).

If you use the example policy, remember to replace `my-s3-bucket` with the name of your bucket.

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` and `s3:GetBucketLocation` actions. You can use this example when you create an Amazon S3 bucket to receive daily SMS usage reports from Amazon SNS.

```
{
  "Statement": [{
    "Sid": "AllowPutObject",
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3::my-s3-bucket/*"
  }, {
    "Sid": "AllowGetBucketLocation",
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": "s3:GetBucketLocation",
    "Resource": "arn:aws:s3::my-s3-bucket"
  }]
}
```

Example Daily Usage Report

After you subscribe to daily usage reports, each day, Amazon SNS puts a CSV file with usage data in the following location:

```
<my-s3-bucket>/SMSUsageReports/<region>/YYYY/MM/DD/00x.csv.gz
```

Each file can contain up to 50,000 records. If the records for a day exceed this quota, Amazon SNS will add multiple files.

The following shows an example report:

```
PublishTimeUTC,MessageId,DestinationPhoneNumber,MessageType,DeliveryStatus,PriceInUSD,PartNumber,TotalL
2016-05-10T03:00:29.476Z,96a298ac-1458-4825-
a7eb-7330e0720b72,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.90084,1,1
2016-05-10T03:00:29.561Z,1e29d394-
d7f4-4dc9-996e-26412032c344,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.34322,1,1
2016-05-10T03:00:30.769Z,98ba941c-afc7-4c51-
ba2c-56c6570a6c08,1XXX5550100,Transactional,Message has been accepted by phone
carrier,0.27815,1,1
```

Managing Phone Numbers and SMS Subscriptions

Amazon SNS provides several options for managing who receives SMS messages from your account. With a limited frequency, you can opt in phone numbers that have opted out of receiving SMS messages from your account. To stop sending messages to SMS subscriptions, you can remove subscriptions or the topics that publish to them.

Topics

- [Opting Out of Receiving SMS Messages](#) (p. 166)
- [Managing Phone Numbers and Subscriptions \(Console\)](#) (p. 166)
- [Managing Phone Numbers and Subscriptions \(AWS SDKs\)](#) (p. 167)

Opting Out of Receiving SMS Messages

Where required by local laws and regulations (such as the US and Canada), SMS recipients can use their devices to opt out by replying to the message with any of the following:

- ARRET (French)
- CANCEL
- END
- OPT-OUT
- OPTOUT
- QUIT
- REMOVE
- STOP
- TD
- UNSUBSCRIBE

To opt out, the recipient must reply to the same long code or short code 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. 171).
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.

Viewing All Opted Out Phone Numbers

To view all opted out phone numbers, submit a `ListPhoneNumbersOptedOut` request with the Amazon SNS API.

Or, you can use the Amazon SNS clients in the AWS SDKs, as shown by the following examples:

AWS SDK for Java

With the AWS SDK for Java, you can use the `listPhoneNumbersOptedOut` method of the `AmazonSNSClient` class:

```
public static void main(String[] args) {
    AmazonSNSClient snsClient = new AmazonSNSClient();
    listOptOut(snsClient);
}

public static void listOptOut(AmazonSNSClient snsClient) {
    String nextToken = null;
    do {
        ListPhoneNumbersOptedOutResult result = snsClient
            .listPhoneNumbersOptedOut(new ListPhoneNumbersOptedOutRequest()
                .withNextToken(nextToken));
        nextToken = result.getNextToken();
        for (String phoneNum : result.getPhoneNumbers()) {
            System.out.println(phoneNum);
        }
    } while (nextToken != null);
}
```

AWS SDK for .NET

With the AWS SDK for .NET, you can use the `ListPhoneNumbersOptedOut` method of the `AmazonSimpleNotificationServiceClient` class:

```
public static void Main(String[] args)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient(Amazon.RegionEndpoint.USWest2);
    ListOptOut(snsClient);
}

public static void ListOptOut(AmazonSimpleNotificationServiceClient snsClient)
{
    String nextToken = null;
    do
    {
        ListPhoneNumbersOptedOutRequest listRequest = new
        ListPhoneNumbersOptedOutRequest { NextToken = nextToken };
        ListPhoneNumbersOptedOutResponse listResponse =
        snsClient.ListPhoneNumbersOptedOut(listRequest);
        nextToken = listResponse.NextToken;
        foreach (String phoneNum in listResponse.PhoneNumbers)
            Console.WriteLine(phoneNum);
    } while (nextToken != null);
}
```

Amazon SNS returns a paginated response, so this example repeats the request each time Amazon SNS returns a next token. When you run this example, it displays a list of all opted out phone numbers in the console output window of your IDE.

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.

Or, you can use the Amazon SNS clients in the AWS SDKs, as shown by the following examples:

AWS SDK for Java

With the AWS SDK for Java, you can use the `checkIfPhoneNumberIsOptedOut` method of the `AmazonSNSClient` class:

```
CheckIfPhoneNumberIsOptedOutRequest request = new
    CheckIfPhoneNumberIsOptedOutRequest().withPhoneNumber(phoneNumber);
System.out.println(snsClient.checkIfPhoneNumberIsOptedOut(request));
```

When you run this example, a true or false result is displayed in the console output window of your IDE:

```
{IsOptedOut: false}
```

AWS SDK for .NET

Using the AWS SDK for .NET, you can use the `CheckIfPhoneNumberIsOptedOut` method of the `AmazonSimpleNotificationServiceClient` class:

```
CheckIfPhoneNumberIsOptedOutRequest request = new CheckIfPhoneNumberIsOptedOutRequest
{ PhoneNumber = phoneNumber };
```

```
Console.WriteLine(snsClient.CheckIfPhoneNumberIsOptedOut(request).IsOptedOut);
```

When you run this example, a true or false result is displayed in the console output window of your IDE:

```
false
```

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.

Or, you can use the Amazon SNS clients in the AWS SDKs, as shown by the following examples:

AWS SDK for Java

With the AWS SDK for Java, you can use the `optInPhoneNumber` method of the `AmazonSNSClient` class:

```
snsClient.optInPhoneNumber(new OptInPhoneNumberRequest().withPhoneNumber(phoneNumber));
```

AWS SDK for .NET

With the AWS SDK for .NET, you can use the `OptInPhoneNumber` method of the `AmazonSimpleNotificationServiceClient` class:

```
snsClient.OptInPhoneNumber(new OptInPhoneNumberRequest { PhoneNumber = phoneNumber});
```

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.

Or, you can use the Amazon SNS clients in the AWS SDKs, as shown by the following examples:

AWS SDK for Java

With the AWS SDK for Java, you can get your subscription ARNs using the `listSubscriptions` method of the `AmazonSNSClient` class:

```
ListSubscriptionsResult result = snsClient.listSubscriptions();  
for (Subscription sub : result.getSubscriptions()) {  
    System.out.println(sub);  
}
```

You can delete a subscription by passing its ARN as a string argument to the `unsubscribe` method:

```
snsClient.unsubscribe(subscriptionArn);
```

AWS SDK for .NET

With the AWS SDK for .NET, use code like the following:

```
ListSubscriptionsResponse response = snsClient.ListSubscriptions();
```

```
// find the subscriptionArn you want
foreach (Subscription sub in response.Subscriptions)
    Console.WriteLine(sub.SubscriptionArn);
// unsubscribe
snsClient.Unsubscribe(subscriptionArn);
```

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.

Or, you can use the Amazon SNS clients in the AWS SDKs, as shown by the following examples:

AWS SDK for Java

With the AWS SDK for Java, you can get your topic ARNs using the `listTopics` method of the `AmazonSNSClient` class:

```
ListTopicsResult result = snsClient.listTopics();
for (Topic t : result.getTopics()) {
    System.out.println(t);
}
```

You can delete a topic by passing its ARN as a string argument to the `deleteTopic` method:

```
snsClient.deleteTopic(topicArn);
```

AWS SDK for .NET

Using the AWS SDK for .NET, use code like the following:

```
ListTopicsResponse restopics= snsClient.ListTopics();
// find the topicArn you want
foreach (Topic t in restopics.Topics)
    Console.WriteLine(t.TopicArn);
// delete
snsClient.DeleteTopic(topicArn);
```

Reserving a Dedicated Short Code for SMS Messaging

To send SMS messages using a persistent short code, you can reserve a dedicated short code that is assigned to your account and available exclusively to you.

A short code is a 5 or 6 digit number that you can use to send SMS messages to certain destinations. Short codes are often used for application-to-person (A2P) messaging, two-factor authentication (2FA), and marketing. Unless you reserve a short code, Amazon SNS will assign a short code to your messages. This short code is shared with other Amazon SNS users, and it varies based upon destination and message type (transactional or promotional). By reserving a short code, you make it easier for your audience to recognize that your organization is the source of your messages.

Your dedicated short code is available exclusively to you, so others are unable to message your audience using the same short code. Consequently, your short code has some protection from malicious activity that might threaten your brand reputation or prompt wireless carriers to block your messages.

Amazon SNS can use your short code to message telephone numbers in the United States. For other destinations, Amazon SNS assigns a long code or alphanumeric code as required.

A dedicated short code supports a higher delivery rate, which enables you send up to 100 SMS messages per second. After you reserve a short code, you can request to increase this quota by submitting an [submitting a request](#).

For pricing information, see [Worldwide SMS Pricing](#).

To reserve 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 for US destinations**.
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.
9. Select your preferred language and contact method, and choose **Submit**.

A customer service associate will contact you for additional information about your use case, including the customized messages your audience members will see when they reply to your short code with HELP or STOP. AWS will work with wireless carriers on your behalf to provision your short code. It typically takes 6 - 12 weeks for all carriers to approve your use case and provision the short code so that you can send messages to the subscribers in their networks. AWS will notify you when the short code provisioning is complete.

To discontinue your short code reservation, send a request to AWS Support.

Supported Regions and Countries

Currently, Amazon SNS supports SMS messaging in the following AWS Regions:

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	sns.us-east-1.amazonaws.com	HTTP and HTTPS
US West (Oregon)	us-west-2	sns.us-west-2.amazonaws.com	HTTP and HTTPS
Europe (Ireland)	eu-west-1	sns.eu-west-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	sns.ap-northeast-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Singapore)	ap-southeast-1	sns.ap-southeast-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Sydney)	ap-southeast-2	sns.ap-southeast-2.amazonaws.com	HTTP and HTTPS

You can use Amazon SNS to send SMS messages to the following countries and regions:

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Afghanistan	AF		
Albania	AL	Yes	
Algeria	DZ		
Andorra	AD	Yes	
Angola	AO	Yes	
Anguilla	AI	Yes	
Antigua and Barbuda	AG	Yes	
Argentina	AR		Yes
Armenia	AM	Yes	
Aruba	AW	Yes	
Australia	AU	Yes	Yes
Austria	AT	Yes	Yes
Azerbaijan	AZ		
Bahamas	BS	Yes	
Bahrain	BH	Yes	
Bangladesh	BD		
Barbados	BB	Yes	
Belarus	BY	Yes ¹ (p. 178)	
Belgium	BE		Yes
Belize	BZ	Yes	
Benin	BJ	Yes	
Bermuda	BM	Yes	
Bhutan	BT	Yes	
Bolivia	BO	Yes	
Bosnia and Herzegovina	BA	Yes	
Botswana	BW	Yes	
Brazil	BR		Yes
Brunei	BN	Yes	
Bulgaria	BG	Yes	
Burkina Faso	BF	Yes	
Burundi	BI	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Cambodia	KH	Yes	
Cameroon	CM	Yes	
Canada	CA		Yes
Cape Verde	CV	Yes	
Cayman Islands	KY		
Central African Republic	CF	Yes	
Chad	TD	Yes	
Chile	CL		Yes
China	CN		For support, contact sales .
Colombia	CO		
Comoros	KM	Yes	
Cook Islands	CK	Yes	
Costa Rica	CR		
Croatia	HR		Yes
Cyprus	CY	Yes	
Czech Republic	CZ		Yes
Democratic Republic of the Congo	CD		
Denmark	DK	Yes	Yes
Djibouti	DJ	Yes	
Dominica	DM	Yes	
Dominican Republic	DO		
East Timor	TL		
Ecuador	EC		
Egypt	EG	Yes ¹ (p. 178)	
El Salvador	SV		
Equatorial Guinea	GQ	Yes	
Estonia	EE	Yes	Yes
Ethiopia	ET		
Faroe Islands	FO	Yes	
Fiji	FJ	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Finland	FI	Yes	Yes
France	FR	Yes	Yes
French Guiana	GF		
Gabon	GA	Yes	
Gambia	GM	Yes	
Georgia	GE	Yes	
Germany	DE	Yes	Yes
Ghana	GH		
Gibraltar	GI	Yes	
Greece	GR	Yes	
Greenland	GL	Yes	
Grenada	GD	Yes	
Guadeloupe	GP	Yes	
Guam	GU		
Guatemala	GT		Yes
Guinea	GN	Yes	
Guinea-Bissau	GW	Yes	
Guyana	GY	Yes	
Haiti	HT	Yes	
Honduras	HN		Yes
Hong Kong	HK	Yes	Yes
Hungary	HU		Yes
Iceland	IS	Yes	
India	IN	Yes ¹ (p. 178)	Yes
Indonesia	ID		Yes
Iraq	IQ		
Ireland	IE	Yes	Yes
Israel	IL	Yes	Yes
Italy	IT	Yes	Yes
Ivory Coast	CI		
Jamaica	JM	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Japan	JP		Yes
Jordan	JO	Yes ¹ (p. 178)	
Kazakhstan	KZ		
Kenya	KE		
Kiribati	KI		
Kuwait	KW	Yes ¹ (p. 178)	
Kyrgyzstan	KG		
Laos	LA		
Latvia	LV	Yes	Yes
Lebanon	LB	Yes	
Lesotho	LS	Yes	
Liberia	LR	Yes	
Libya	LY	Yes	
Liechtenstein	LI	Yes	
Lithuania	LT	Yes	Yes
Luxembourg	LU	Yes	
Macau	MO	Yes	
Former Yugoslav Republic of Macedonia	MK	Yes	
Madagascar	MG	Yes	
Malawi	MW	Yes	
Malaysia	MY		Yes
Maldives	MV	Yes	
Mali	ML		
Malta	MT	Yes	
Martinique	MQ	Yes	
Mauritania	MR	Yes	
Mauritius	MU	Yes	
Mexico	MX		Yes
Moldova	MD	Yes	
Monaco	MC		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Mongolia	MO	Yes	
Montenegro	ME	Yes	
Montserrat	MS	Yes	
Morocco	MA		
Mozambique	MZ		
Myanmar	MM		
Namibia	NA		
Nepal	NP		
Netherlands	NL	Yes	Yes
Netherlands Antilles	AN	Yes	
New Caledonia	NC	Yes	
New Zealand	NZ		Yes
Nicaragua	NI		
Niger	NE	Yes	
Nigeria	NG		
Norway	NO	Yes	Yes
Oman	OM	Yes	
Pakistan	PK		
Palau	PW		
Palestinian Territories	PS		
Panama	PA		
Papua New Guinea	PG	Yes	
Paraguay	PY	Yes	
Peru	PE		
Philippines	PH		Yes
Poland	PL	Yes	Yes
Portugal	PT	Yes	Yes
Puerto Rico	PR		Yes
Qatar	QA	Yes ¹ (p. 178)	
Republic of the Congo	CG		
Reunion Island	RE	Yes	

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Romania	RO		Yes
Russia	RU	Yes ¹ (p. 178)	Yes
Rwanda	RW	Yes	
Saint Kitts and Nevis	KN		
Saint Lucia	LC		
Saint Vincent and the Grenadines	VC		
Samoa	WS	Yes	
Sao Tome and Principe	ST	Yes	
Saudi Arabia	SA	Yes ¹ (p. 178)	
Senegal	SN	Yes	
Serbia	RS	Yes	
Seychelles	SC	Yes	
Sierra Leone	SL	Yes	
Singapore	SG	Yes	Yes
Slovakia	SK	Yes	Yes
Slovenia	SI	Yes	Yes
Solomon Islands	SB	Yes	
Somalia	SO	Yes	
South Africa	ZA		Yes
South Korea	KR		Yes
South Sudan	SS	Yes	
Spain	ES	Yes	Yes
Sri Lanka	LK		
Suriname	SR	Yes	
Swaziland	SZ	Yes	
Sweden	SE	Yes	Yes
Switzerland	CH	Yes	Yes
Taiwan	TW		Yes
Tajikistan	TJ	Yes	
Tanzania	TZ		

Country or region	ISO code	Supports sender IDs	Supports two-way SMS
Thailand	TH	Yes ¹ (p. 178)	Yes
Togo	TG	Yes	
Tonga	TO	Yes	
Trinidad and Tobago	TT	Yes	
Tunisia	TN	Yes	
Turkey	TR		Yes
Turkmenistan	TM	Yes	
Turks and Caicos Islands	TC	Yes	
Uganda	UG	Yes	
Ukraine	UA	Yes	Yes
United Arab Emirates	AE	Yes ¹ (p. 178)	
United Kingdom	GB	Yes	Yes
United States	US		Yes
Uruguay	UY		
Uzbekistan	UZ	Yes	
Vanuatu	VU	Yes	
Venezuela	VE		
Vietnam	VN	Yes ¹ (p. 178)	
Virgin Islands, British	VG	Yes	
Virgin Islands, US	VI	Yes	
Yemen	YE	Yes	
Zambia	ZM	Yes	
Zimbabwe	ZW	Yes	

Notes

1. Senders are required to use a pre-registered alphabetic sender ID. To request a sender ID from AWS Support, file a support request. Some countries require senders to meet specific requirements or abide by certain restrictions in order to obtain approval. In these cases, AWS Support might contact you for additional information after you submit your sender ID request.

Troubleshooting Amazon SNS Topics

This section provides information about troubleshooting Amazon SNS topics.

Troubleshooting Amazon Simple Notification Service 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*.

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 in Amazon SNS \(p. 180\)](#)
- [Identity and Access Management in Amazon SNS \(p. 188\)](#)
- [Logging and Monitoring in Amazon SNS \(p. 209\)](#)
- [Compliance Validation for Amazon SNS \(p. 216\)](#)
- [Resilience in Amazon SNS \(p. 217\)](#)
- [Infrastructure Security in Amazon SNS \(p. 217\)](#)
- [Amazon SNS Security Best Practices \(p. 218\)](#)

Data Protection in Amazon SNS

The following sections provide information about data protection in Amazon SNS.

Topics

- [Data Encryption \(p. 180\)](#)
- [Internetwork Traffic Privacy \(p. 185\)](#)

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 \(p. 180\)](#)
- [Key Management \(p. 182\)](#)

Encryption at Rest

Server-side encryption (SSE) lets you transmit 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 \(p. 27\)](#). 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 \(p. 181\)](#) that encrypt your message data are also encrypted and stored with the data they protect.

The following are benefits of using AWS KMS:

- You can create and manage [customer master keys \(CMKs\) \(p. 181\)](#) yourself.
- You can also use the AWS managed CMK for Amazon SNS, which is 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* and the [AWS Key Management Service Cryptographic Details](#) whitepaper.

Topics

- [Encryption Scope \(p. 181\)](#)
- [Key Terms \(p. 181\)](#)

Encryption Scope

SSE encrypts the body of a message in an Amazon SNS topic.

SSE doesn't encrypt the following:

- Topic metadata (topic name and attributes)
- Message metadata (subject, message ID, timestamp, and attributes)
- Per-topic metrics

Note

- A message is encrypted only if it is sent after the encryption of a topic is enabled. Amazon SNS doesn't encrypt backlogged messages.
- Any encrypted message remains encrypted even if the encryption of its topic is disabled.

Key Terms

The following key terms can help you better understand the functionality of SSE. For detailed descriptions, see the [Amazon Simple Notification Service API Reference](#).

Data Key

The data encryption key (DEK) responsible for encrypting the contents of Amazon SNS messages.

For more information, see [Data Keys](#) in the *AWS Key Management Service Developer Guide* and [Envelope Encryption](#) in the *AWS Encryption SDK Developer Guide*.

Customer Master Key ID

The alias, alias ARN, key ID, or key ARN of an AWS managed customer master key (CMK) or a custom CMK—in your account or in another account. While the alias of the AWS managed CMK for Amazon

SNS is always `alias/aws/sns`, the alias of a custom CMK can, for example, be `alias/MyAlias`. You can use these CMKs to protect the messages in Amazon SNS topics.

Note

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SNS for a topic, AWS KMS creates the AWS managed CMK for Amazon SNS.
- Alternatively, the first time you use the `Publish` action on a topic with SSE enabled, AWS KMS creates the AWS managed CMK for Amazon SNS.

You can create CMKs, define the policies that control how CMKs can be used, and audit CMK usage using the **Customer managed keys** section of the AWS KMS console or the `CreateKey` AWS KMS action. For more information, see [Customer Master Keys \(CMKs\)](#) and [Creating Keys](#) in the *AWS Key Management Service Developer Guide*. For more examples of CMK identifiers, see `KeyId` in the *AWS Key Management Service API Reference*. For information about finding CMK identifiers, see [Find the Key ID and ARN](#) in the *AWS Key Management Service Developer Guide*.

Important

There are additional charges for using AWS KMS. For more information, see [Estimating AWS KMS Costs \(p. 182\)](#) 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 \(p. 182\)](#)
- [Configuring AWS KMS Permissions \(p. 183\)](#)
- [AWS KMS Errors \(p. 185\)](#)

Estimating AWS KMS Costs

To predict costs and better understand your AWS bill, you might want to know how often Amazon SNS uses your customer master key (CMK).

Note

Although the following formula can give you a very good idea of expected costs, actual costs might be higher because of the distributed nature of Amazon SNS.

To calculate the number of API requests (*R*) *per topic*, use the following formula:

$$R = B / D * (2 * P)$$

B is the billing period (in seconds).

D is the data key reuse period (in seconds—Amazon SNS reuses a data key for up to 5 minutes).

P is the number of publishing [principals](#) that send to the Amazon SNS topic.

The following are example calculations. For exact pricing information, see [AWS Key Management Service Pricing](#).

Example 1: Calculating the Number of AWS KMS API Calls for 1 Publisher and 1 Topic

This example assumes the following:

- The billing period is January 1-31 (2,678,400 seconds).

- The data key reuse period is 5 minutes (300 seconds).
- There is 1 topic.
- There is 1 publishing principal.

$$2,678,400 / 300 * (2 * 1) = 17,856$$

Example 2: Calculating the Number of AWS KMS API Calls for Multiple Publishers and 2 Topics

This example assumes the following:

- The billing period is February 1-28 (2,419,200 seconds).
- The data key reuse period is 5 minutes (300 seconds).
- There are 2 topics.
- The first topic has 3 publishing principals.
- The second topic has 5 publishing principals.

$$(2,419,200 / 300 * (2 * 3)) + (2,419,200 / 300 * (2 * 5)) = 129,024$$

Configuring AWS KMS Permissions

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of topics and encryption and decryption of messages. For examples and more information about AWS KMS permissions, see [AWS KMS API Permissions: Actions and Resources Reference](#) in the *AWS Key Management Service Developer Guide*.

Note

You can also manage permissions for KMS keys using IAM policies. For more information, see [Using IAM Policies with AWS KMS](#).

While you can configure global permissions to send to and receive from Amazon SNS, AWS KMS requires explicitly naming the full ARN of CMKs in specific regions in the **Resource** section of an IAM policy.

You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. To do this, name the principals that produce and consume encrypted messages in Amazon SNS as users in the CMK key policy.

Alternatively, you can specify the required AWS KMS actions and CMK ARN in an IAM policy assigned to the principals that publish and subscribe to receive encrypted messages in Amazon SNS. For more information, see [Managing Access to AWS KMS CMKs](#) in the *AWS Key Management Service Developer Guide*.

Allow a User to Send Messages to a Topic with SSE

The publisher must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions for the customer master key (CMK).

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:default-regionus-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }], {
```

```

    "Effect": "Allow",
    "Action": [
        "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:123456789012:MyTopic"
  }
}

```

Enable Compatibility between Event Sources from AWS Services and Encrypted Topics

Several AWS services publish events to Amazon SNS topics. To allow these event sources to work with encrypted topics, you must perform the following steps.

1. Use a customer managed CMK. For more information, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*.
2. To allow the AWS service to have the `kms:GenerateDataKey*` and `kms:Decrypt` permissions, add the following statement to the CMK policy.

```

{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey*",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }]
}

```

Event Source	Service Principal
Amazon CloudWatch	cloudwatch.amazonaws.com
Amazon CloudWatch Events	events.amazonaws.com
Amazon DynamoDB	dynamodb.amazonaws.com
Amazon S3 Glacier	glacier.amazonaws.com
Amazon Redshift	redshift.amazonaws.com
Amazon Simple Email Service	ses.amazonaws.com
Amazon Simple Storage Service	s3.amazonaws.com
AWS CodeCommit	codecommit.amazonaws.com
AWS Database Migration Service	dms.amazonaws.com
AWS Directory Service	ds.amazonaws.com
AWS Snowball	importexport.amazonaws.com

Note

Some Amazon SNS event sources require you to provide an IAM role (rather than the service principal) in the AWS KMS key policy:

- [Amazon EC2 Auto Scaling](#)
 - [Amazon Elastic Transcoder](#)
 - [AWS CodePipeline](#)
 - [AWS Config](#)
 - [AWS Elastic Beanstalk](#)
 - [AWS IoT](#)
3. [Enable SSE for your topic \(p. 27\)](#) using your CMK.
 4. Provide the ARN of the encrypted topic to the event source.

AWS KMS Errors

When you work with Amazon SNS and AWS KMS, you might encounter errors. The following list describes the errors and possible troubleshooting solutions.

KMSAccessDeniedException

The ciphertext references a key that doesn't exist or that you don't have access to.

HTTP Status Code: 400

KMSDisabledException

The request was rejected because the specified CMK isn't enabled.

HTTP Status Code: 400

KMSInvalidStateException

The request was rejected because the state of the specified resource isn't valid for this request. For more information, see [How Key State Affects Use of a Customer Master Key](#) in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

KMSNotFoundException

The request was rejected because the specified entity or resource can't be found.

HTTP Status Code: 400

KMSOptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

KMSThrottlingException

The request was denied due to request throttling. For more information about throttling, see [Limits](#) in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

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 \(p. 186\)](#)
- [Creating an Amazon VPC Endpoint Policy for Amazon SNS \(p. 187\)](#)

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 [Tutorial: Publishing Amazon SNS Messages Privately from Amazon VPC \(p. 32\)](#).

Creating the Endpoint

You can create an Amazon SNS endpoint in your VPC using the AWS Management Console, the AWS CLI, an AWS SDK, the Amazon SNS API, or AWS CloudFormation.

For information about creating and configuring an endpoint using the Amazon VPC console or the AWS CLI, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

Note

When you create an endpoint, specify Amazon SNS as the service that you want your VPC to connect to. In the Amazon VPC console, service names vary based on the region. For example, if you choose US East (N. Virginia), the service name is **com.amazonaws.us-east-1.sns**.

For information about creating and configuring an endpoint using AWS CloudFormation, see the [AWS::EC2::VPCEndpoint](#) resource in the *AWS CloudFormation User Guide*.

Testing the Connection Between Your VPC and Amazon SNS

After you create an endpoint for Amazon SNS, you can publish messages from your VPC to your Amazon SNS topics. To test this connection, do the following:

1. Connect to an Amazon EC2 instance that resides in your VPC. For information about connecting, see [Connect to Your Linux Instance](#) or [Connecting to Your Windows Instance](#) in the Amazon EC2 documentation.

For example, to connect to a Linux instance using an SSH client, run the following command from a terminal:

```
$ ssh -i ec2-key-pair.pem ec2-user@instance-hostname
```

Where:

- *ec2-key-pair.pem* is the file that contains the key pair that Amazon EC2 provided when you created the instance.
 - *instance-hostname* is the public hostname of the instance. To get the hostname in the [Amazon EC2 console](#): Choose **Instances**, choose your instance, and find the value for **Public DNS (IPv4)**.
2. From your instance, use the Amazon SNS [publish](#) command with the AWS CLI. You can send a simple message to a topic with the following command:

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
```

Where:

- *aws-region* is the AWS Region that the topic is located in.
- *sns-topic-arn* is the Amazon Resource Name (ARN) of the topic. To get the ARN from the [Amazon SNS console](#): Choose **Topics**, find your topic, and find the value in the **ARN** column.

If the message is successfully received by Amazon SNS, the terminal prints a message ID, like the following:

```
{
  "MessageId": "6c96dfff-0fdf-5b37-88d7-8cba910a8b64"
}
```

Creating an Amazon VPC Endpoint Policy for Amazon SNS

You can create a policy for Amazon VPC endpoints for Amazon SNS in which you specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following example VPC endpoint policy specifies that the IAM user `MyUser` is allowed to publish to the Amazon SNS topic `MyTopic`.

```
{
  "Statement": [{
    "Action": ["sns:Publish"],
    "Effect": "Allow",
    "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```



```
}
```

The following are denied:

- Other Amazon SNS API actions, such as `sns:Subscribe` and `sns:Unsubscribe`.
- Other IAM users and rules which attempt to use this VPC endpoint.
- `MyUser` publishing to a different Amazon SNS topic.

Note

The IAM user can still use other Amazon SNS API actions from *outside* the VPC.

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. 188)
- [Access Control](#) (p. 189)
- [Overview of Managing Access in Amazon SNS](#) (p. 189)
- [Using Identity-Based Policies with Amazon SQS](#) (p. 202)
- [Using Temporary Security Credentials with Amazon SNS](#) (p. 208)
- [Amazon SNS API Permissions: Actions and Resources Reference](#) (p. 208)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An [IAM user](#) is an identity within your AWS account that has specific custom permissions (for example, permissions to create a topic in Amazon SNS). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon SNS supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

- **IAM role** – An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:
 - **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the *IAM User Guide*.
 - **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. 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 \(p. 190\)](#)
- [Key Concepts \(p. 190\)](#)
- [Architectural Overview \(p. 193\)](#)
- [Using the Access Policy Language \(p. 194\)](#)
- [Evaluation Logic \(p. 194\)](#)
- [Example Cases for Amazon SNS Access Control \(p. 198\)](#)

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 \(p. 199\)](#).
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see [Limit Subscriptions to HTTPS \(p. 199\)](#).
- 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 \(p. 200\)](#).

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 \(p. 190\)](#)
- [Statement \(p. 191\)](#)
- [Policy \(p. 191\)](#)
- [Issuer \(p. 191\)](#)
- [Principal \(p. 191\)](#)
- [Action \(p. 191\)](#)
- [Resource \(p. 191\)](#)
- [Conditions and Keys \(p. 192\)](#)
- [Requester \(p. 192\)](#)
- [Evaluation \(p. 192\)](#)
- [Effect \(p. 192\)](#)
- [Default Deny \(p. 192\)](#)
- [Allow \(p. 192\)](#)
- [Explicit Deny \(p. 193\)](#)

Permission

A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *publish* (B) to *TopicA* (C) as long as *she uses the HTTP protocol* (D). Whenever

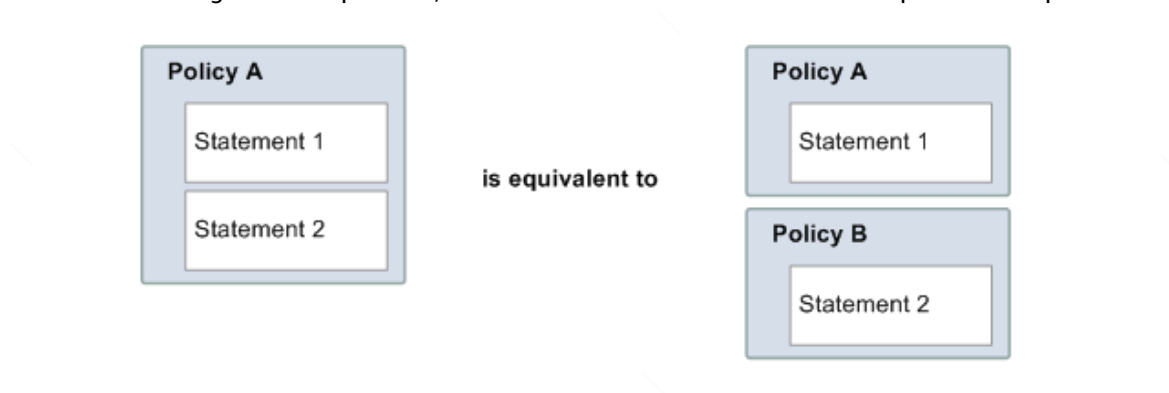
Jane publishes to TopicA, the service checks to see if she has permission and if the request satisfies the conditions set forth in the permission.

Statement

A *statement* is the formal description of a single permission, written in the access policy language. You always write a statement as part of a broader container document known as a *policy* (see the next concept).

Policy

A *policy* is a document (written in the access policy language) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can subscribe using the email protocol, and another that states that Bob cannot publish to TopicA. As shown in the following figure, an equivalent scenario would be to have two policies, one that states that Jane can subscribe using the email protocol, and another that states that Bob cannot publish to TopicA.



Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (that is, you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address.

Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. For example, Jane sends a request to Amazon SNS with `Action=Subscribe`. You can specify one or multiple actions in a policy.

Resource

The *resource* is the object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies."

Conditions and Keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (for example, the request must arrive before a specific day)
- IP address (for example, the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called `aws:CurrentTime` and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The AWS service itself (for example, Amazon SQS or Amazon SNS) might also define service-specific keys. For more information, see [Amazon SNS API Permissions: Actions and Resources Reference \(p. 208\)](#).

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 \(p. 194\)](#).

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, grant 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, grant 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 \(p. 194\)](#).

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 \(p. 194\)](#).

Default Deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

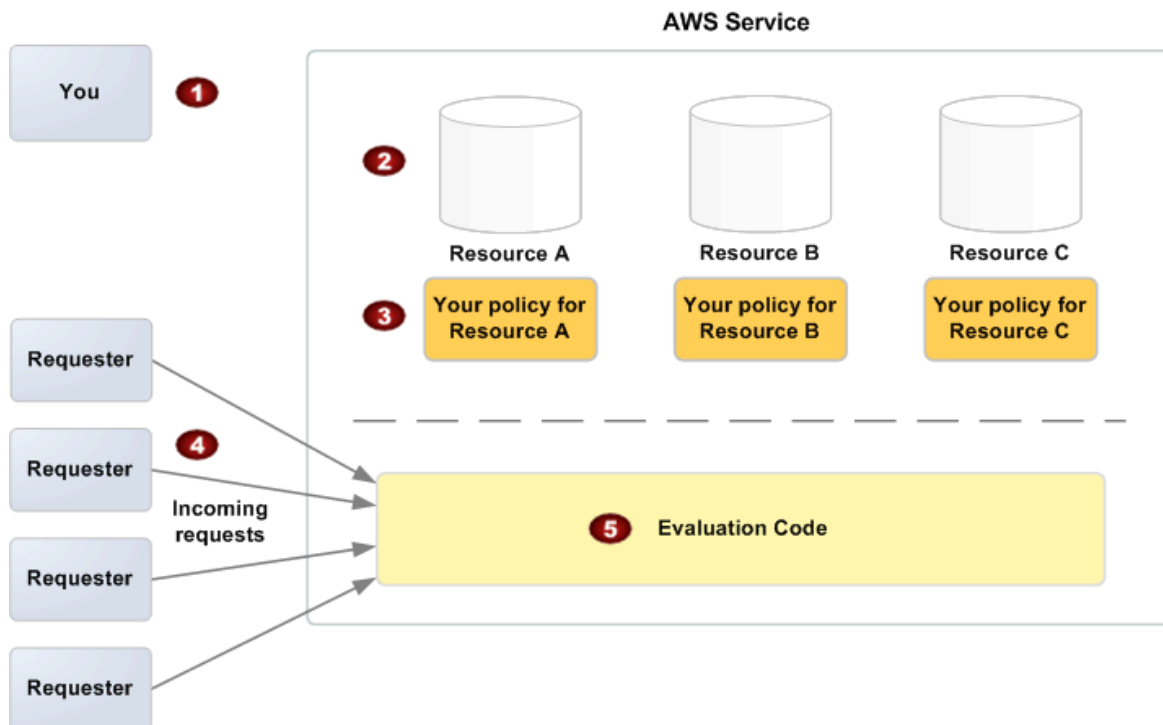
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

Explicit Deny

An *explicit deny* results from a statement that has `effect=deny`, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

Architectural Overview

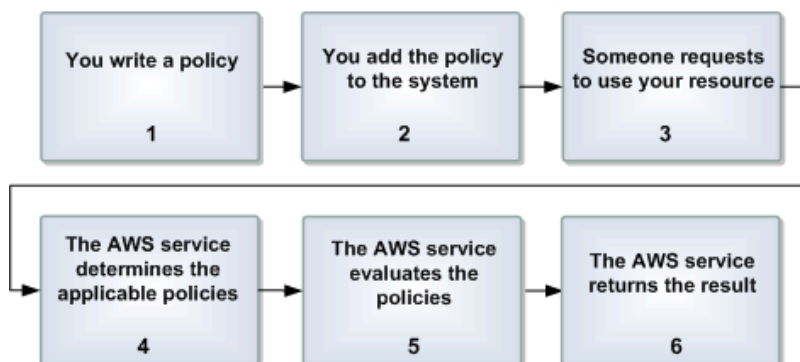
The following figure and table describe the main components that interact to provide access control for your resources.



1	You, the resource owner.
2	Your resources (contained within the AWS service; for example, Amazon SQS queues).
3	<p>Your policies.</p> <p>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.</p>
4	Requesters and their incoming requests to the AWS service.
5	<p>The access policy language evaluation code.</p> <p>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. 194).</p>

Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.



Process for Using Access Control with the Access Policy Language

1	<p>You write a policy for your resource.</p> <p>For example, you write a policy to specify permissions for your Amazon SNS topics.</p>
2	<p>You upload your policy to AWS.</p> <p>The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SNS <code>SetTopicAttributes</code> action to upload a policy for a particular Amazon SNS topic.</p>
3	<p>Someone sends a request to use your resource.</p> <p>For example, a user sends a request to Amazon SNS to use one of your topics.</p>
4	<p>The AWS service determines which policies are applicable to the request.</p> <p>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.).</p>
5	<p>The AWS service evaluates the policies.</p> <p>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. 194).</p>
6	<p>The AWS service either denies the request or continues to process it.</p> <p>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.</p>

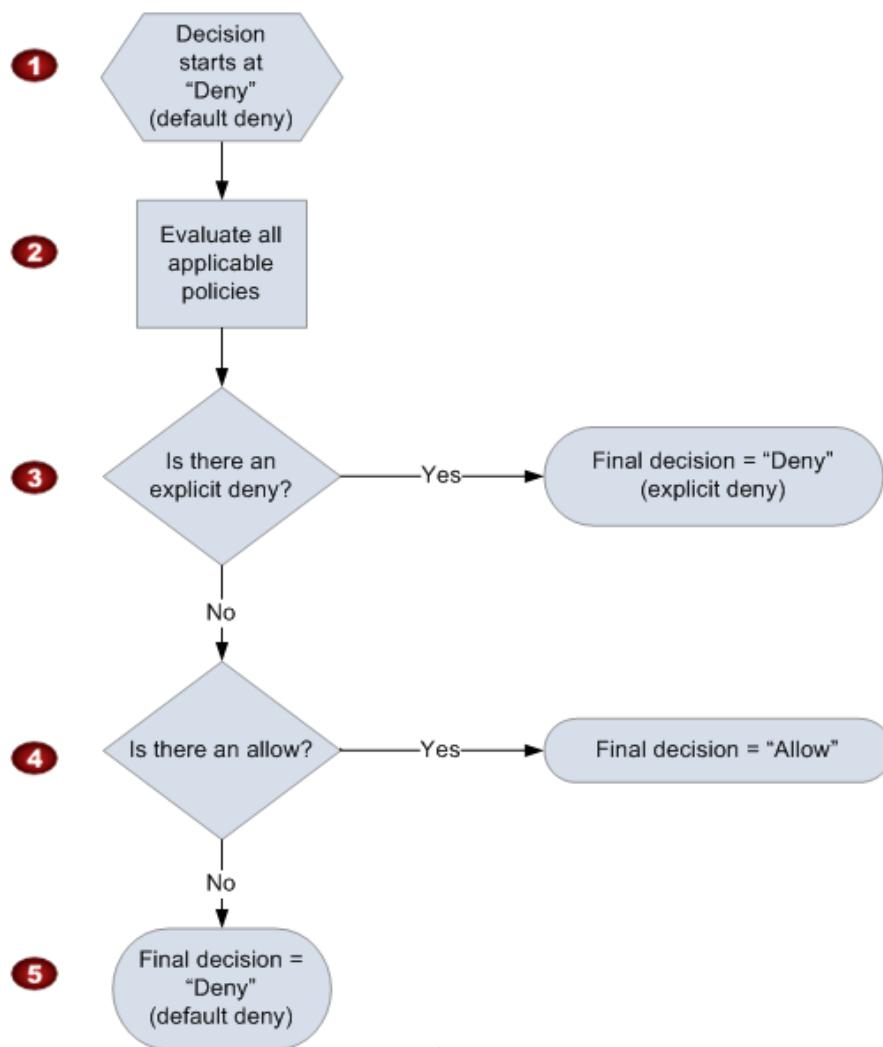
Evaluation Logic

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	<p>The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions).</p> <p>The order in which the enforcement code evaluates the policies is not important.</p>
3	<p>In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request.</p> <p>If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit Deny (p. 193)).</p>

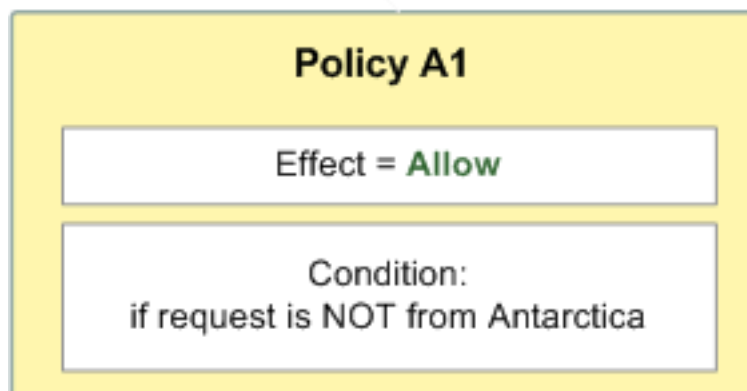
4	<p>If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request.</p> <p>If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).</p>
5	<p>If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a <i>default deny</i> (for more information, see Default Deny (p. 192))).</p>

The Interplay of Explicit and Default Denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if a user requests to use Amazon SNS, but the policy on the topic doesn't refer to the user's AWS account at all, then that policy results in a default deny.

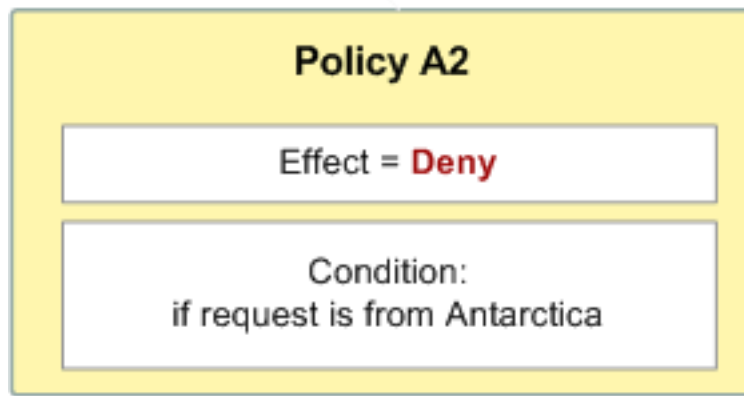
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met, and so the default result in that case is a default deny.

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



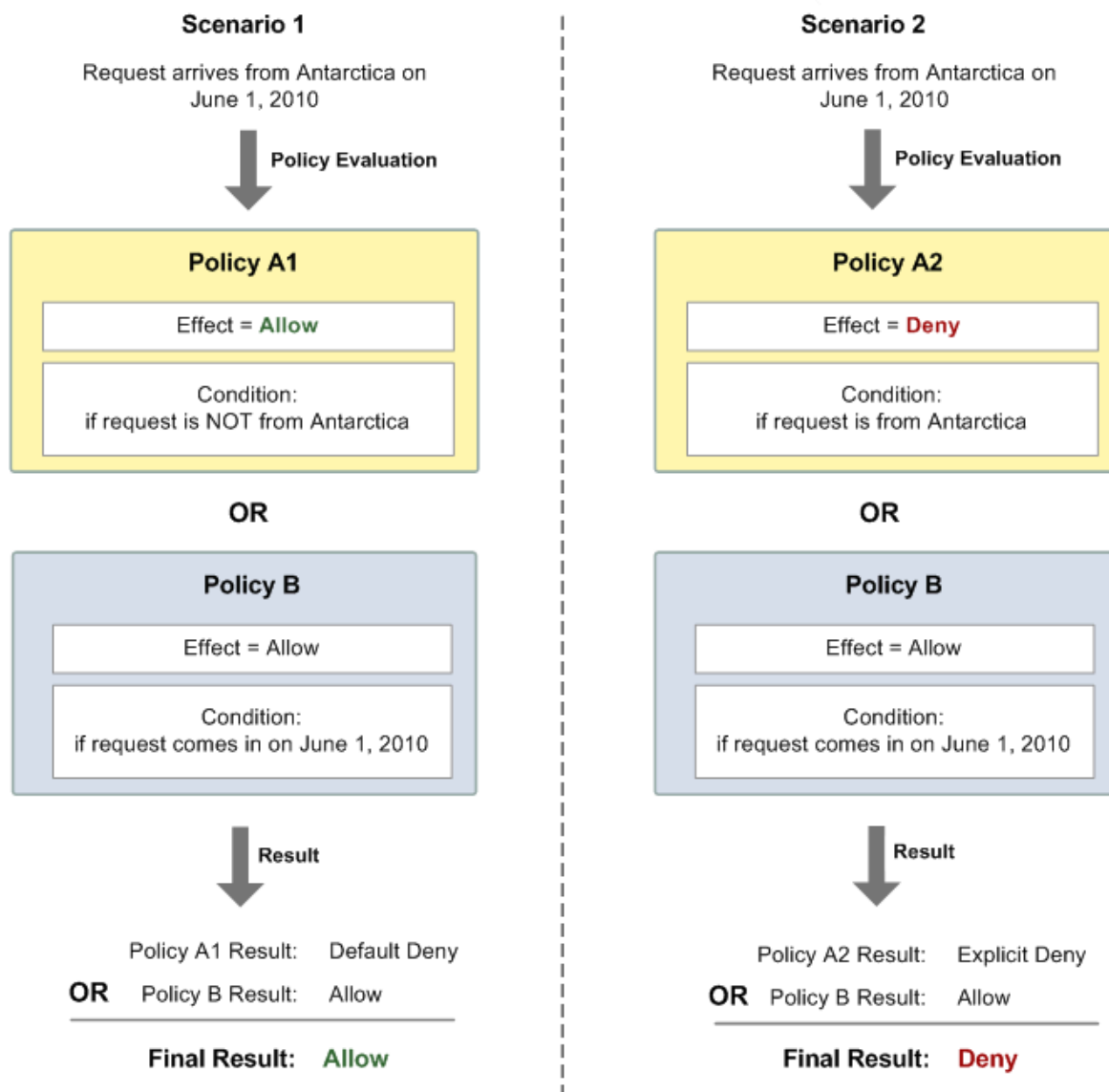
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.



In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy A2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

Example Cases for Amazon SNS Access Control

Topics

- [Grant AWS Account Access to a Topic \(p. 199\)](#)
- [Limit Subscriptions to HTTPS \(p. 199\)](#)
- [Publish Messages to an Amazon SQS Queue \(p. 200\)](#)

- [Allow Any AWS Resource to Publish to a Topic \(p. 200\)](#)
- [Allow an Amazon S3 Bucket to Publish to a Topic \(p. 201\)](#)
- [Allow Any CloudWatch Alarm in an AWS Account to Publish to an Amazon SNS Topic in a Different AWS Account \(p. 201\)](#)
- [Restrict Publication to an Amazon SNS Topic Only from a Specific VPC Endpoint \(p. 202\)](#)

This section grants a few examples of typical use cases for access control.

Grant AWS Account Access to a Topic

Let's say you have a topic in the Amazon SNS system. In the simplest case, you want to allow one or more AWS accounts access to a specific topic action (for example, Publish).

You can do this using the Amazon SNS API action `AddPermission`. It takes a topic, a list of AWS account IDs, a list of actions, and a label, and automatically creates a new statement in the topic's access control policy. In this case, you don't write a policy yourself, because Amazon SNS automatically generates the new policy statement for you. You can remove the policy statement later by calling `RemovePermission` with its label.

For example, if you called `AddPermission` on the topic `arn:aws:sns:us-east-2:444455556666:MyTopic`, with AWS account ID `1111-2222-3333`, the `Publish` action, and the label `grant-1234-publish`, Amazon SNS would generate and insert the following access control policy statement:

```
{
  "Statement": [{
    "Sid": "grant-1234-publish",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Publish"],
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic"
  }]
}
```

Once this statement is added, the user with AWS account `1111-2222-3333` can publish messages to the topic.

Limit Subscriptions to HTTPS

In the following example, you limit the notification delivery protocol to HTTPS.

You need to know how to write your own policy for the topic because the Amazon SNS `AddPermission` action doesn't let you specify a protocol restriction when granting someone access to your topic. In this case, you would write your own policy, and then use the `SetTopicAttributes` action to set the topic's `Policy` attribute to your new policy.

The following example of a full policy grants the AWS account ID `1111-2222-3333` the ability to subscribe to notifications from a topic.

```
{
  "Statement": [{
    "Sid": "Statement1",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
  },
```

```
"Action": ["sns:Subscribe"],
"Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
"Condition": {
  "StringEquals": {
    "sns:Protocol": "https"
  }
}
}]
}
```

Publish Messages to an Amazon SQS Queue

In this use case, you want to publish messages from your topic to your Amazon SQS queue. Like Amazon SNS, Amazon SQS uses Amazon's access control policy language. To allow Amazon SNS to send messages, you'll need to use the Amazon SQS action `SetQueueAttributes` to set a policy on the queue.

Again, you'll need to know how to write your own policy because the Amazon SQS `AddPermission` action doesn't create policy statements with conditions.

Note

The example presented below is an Amazon SQS policy (controlling access to your queue), not an Amazon SNS policy (controlling access to your topic). The actions are Amazon SQS actions, and the resource is the Amazon Resource Name (ARN) of the queue. You can determine the queue's ARN by retrieving the queue's `QueueArn` attribute with the `GetQueueAttributes` action.

```
{
  "Statement": [{
    "Sid": "Allow-SNS-SendMessage",
    "Effect": "Allow",
    "Principal": "*",
    "Action": ["sqs:SendMessage"],
    "Resource": "arn:aws:sqs:us-east-2:444455556666:MyQueue",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-2:444455556666:MyTopic"
      }
    }
  }]
}
```

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 Any AWS Resource to Publish to a Topic

In this case, you want to configure a topic's policy so that another AWS account's resource (for example, Amazon S3 bucket, Amazon EC2 instance, or Amazon SQS queue) can publish to your topic. 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.

In the following example statement, the topic owner in these policies is 1111-2222-3333 and the AWS resource owner is 4444-5555-6666. The example grants the AWS account ID 4444-5555-6666 the ability to publish to My-Topic from any AWS resource owned by the account.

Note

If you publish messages directly (rather than having an AWS resource publish messages on your behalf), a policy in which you specify an empty *Principal* and use `AWS:SourceAccount` as a condition will not work.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:111122223333:MyTopic",
    "Condition": {
      "StringEquals": {
        "AWS:SourceAccount": "444455556666"
      }
    }
  }]
}
```

Allow an Amazon S3 Bucket 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 `ArnLike` condition to make sure the ARN of the resource making the request (the `AWS:SourceARN`) is an Amazon S3 ARN. You could use a similar condition to restrict the permission to a set of Amazon S3 buckets, or even to a specific bucket. In this example, the topic owner is 1111-2222-3333 and the Amazon S3 owner is 4444-5555-6666. The example states that any Amazon S3 bucket owned by 4444-5555-6666 is allowed to publish to My-Topic.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:111122223333:MyTopic",
    "Condition": {
      "StringEquals": {
        "AWS:SourceAccount": "444455556666"
      },
      "ArnLike": {
        "AWS:SourceArn": "arn:aws:s3:*:*:*"
      }
    }
  }]
}
```

Allow Any CloudWatch Alarm in an AWS Account to Publish to an Amazon SNS Topic in a Different AWS Account

In this case, any CloudWatch alarms in account 111122223333 are allowed to publish to an Amazon SNS topic in account 444455556666.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": "*"
    },
    "Action": "SNS:Publish",
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
      }
    }
  }]
}
```

Restrict Publication to an Amazon SNS Topic Only from a Specific VPC Endpoint

In this case, the topic in account 444455556666 is allowed to publish only from the VPC endpoint with the ID `vpc-1ab2c34d`.

```
{
  "Statement": [{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "SNS:Publish",
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpce": "vpc-1ab2c34d"
      }
    }
  }]
}
```

Using Identity-Based Policies with Amazon SQS

Topics

- [IAM and Amazon SNS Policies Together \(p. 203\)](#)
- [Amazon SNS Resource ARN Format \(p. 205\)](#)
- [Amazon SNS API Actions \(p. 206\)](#)
- [Amazon SNS Policy Keys \(p. 206\)](#)
- [Example Policies for Amazon SNS \(p. 206\)](#)

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.

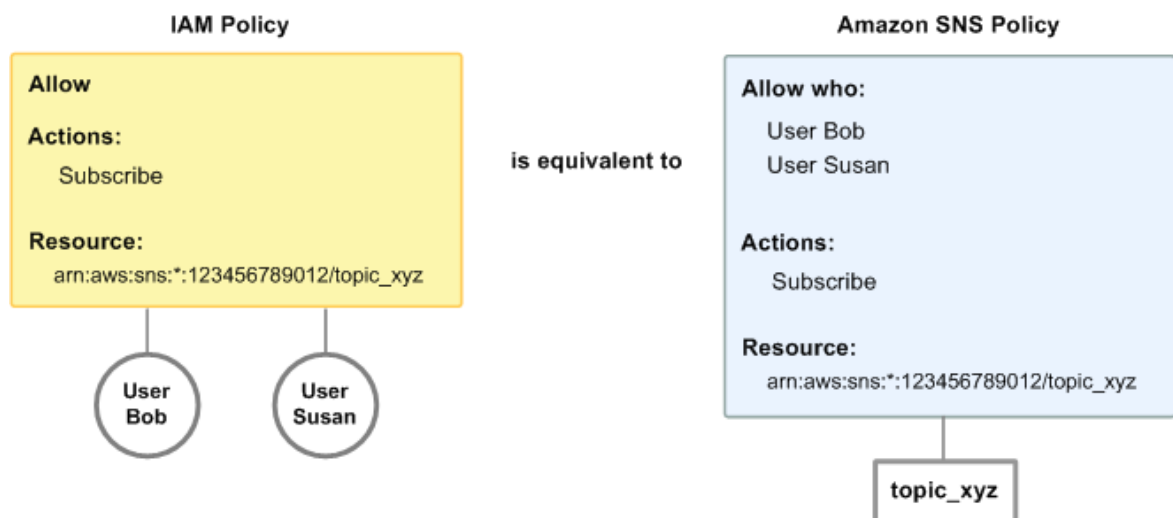
For examples of policies that cover Amazon SNS actions and resources, see [Example Policies for Amazon SNS \(p. 206\)](#).

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

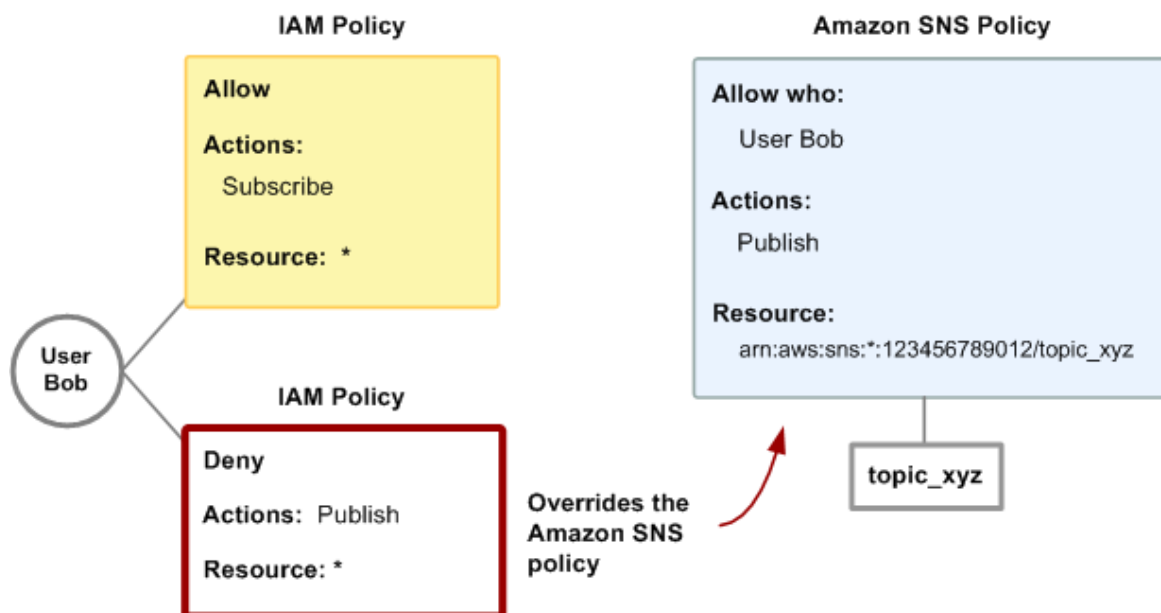
In this example, both an IAM policy and an Amazon SNS policy apply to Bob. The IAM policy grants him permission for `Subscribe` on any of the AWS account's topics, whereas the Amazon SNS policy grants him permission to use `Publish` on a specific topic (`topic_xyz`). The following diagram illustrates 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 \(p. 194\)](#)). The following diagram illustrates the concept.



For examples of policies that cover Amazon SNS actions and resources, see [Example Policies for Amazon SNS \(p. 206\)](#). For more information about writing Amazon SNS policies, go to the [technical documentation for Amazon SNS](#).

Amazon SNS Resource ARN Format

For Amazon SNS, topics are the only resource type you can specify in a policy. The following is the Amazon Resource Name (ARN) format for topics.

```
arn:aws:sns:region:account_ID:topic_name
```

For more information about ARNs, go to [ARNs](#) in *IAM User Guide*.

Example

The following is an ARN for a topic named `my_topic` in the `us-east-2` region, belonging to AWS account `123456789012`.

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

Example

If you had a topic named `my_topic` in each of the different Regions that Amazon SNS supports, you could specify the topics with the following ARN.

```
arn:aws:sns:*:123456789012:my_topic
```

You can use `*` and `?` wildcards in the topic name. For example, the following could refer to all the topics created by Bob that he has prefixed with `bob_`.

```
arn:aws:sns:*:123456789012:bob_*
```

As a convenience to you, when you create a topic, Amazon SNS returns the topic's ARN in the response.

Amazon SNS API Actions

In an IAM policy, you can specify any actions that Amazon SNS offers. However, the `ConfirmSubscription` and `Unsubscribe` actions do not require authentication, which means that even if you specify those actions in a policy, IAM won't restrict users' access to those actions.

Each action you specify in a policy must be prefixed with the lowercase string `sns:`. To specify all Amazon SNS actions, for example, you would use `sns:*`. For a list of the actions, go to the [Amazon Simple Notification Service API Reference](#).

Amazon SNS Policy Keys

Amazon SNS implements the following AWS-wide policy keys, plus some service-specific keys.

For a list of condition keys supported by each AWS service, see [Actions, Resources, and Condition Keys for AWS Services](#) in the *IAM User Guide*. For a list of condition keys that can be used in multiple AWS services, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

Amazon SNS uses the following service-specific keys. Use these keys in policies that restrict access to `Subscribe` requests.

- **sns:Endpoint**—The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example Policies for Amazon SNS \(p. 206\)](#)) 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 \(p. 206\)](#)) to restrict publication to specific delivery protocols (for example, `https`).

Example Policies for Amazon SNS

This section shows several simple policies for controlling user access to Amazon SNS.

Note

In the future, Amazon SNS might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

Example 1: Allow a group to create and manage topics

In this example, we create a policy that grants access to `CreateTopic`, `ListTopics`, `SetTopicAttributes`, and `DeleteTopic`.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:CreateTopic", "sns:ListTopics", "sns:SetTopicAttributes",
"sns:DeleteTopic"],
    "Resource": "*"
  }]
}
```

Example 2: Allow the IT group to publish messages to a particular topic

In this example, we create a group for IT, and assign a policy that grants access to `Publish` on the specific topic of interest.

```
{
```

```
"Statement": [{
  "Effect": "Allow",
  "Action": "sns:Publish",
  "Resource": "arn:aws:sns:*:123456789012:MyTopic"
}]
}
```

Example 3: Give users in the AWS account ability to subscribe to topics

In this example, we create a policy that grants access to the `Subscribe` action, with string matching conditions for the `sns:Protocol` and `sns:Endpoint` policy keys.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:Subscribe"],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "SNS:Endpoint": "*@example.com"
      },
      "StringEquals": {
        "sns:Protocol": "email"
      }
    }
  }]
}
```

Example 4: Allow a partner to publish messages to a particular topic

You can use an Amazon SNS policy or an IAM policy to allow a partner to publish to a specific topic. If your partner has an AWS account, it might be easier to use an Amazon SNS policy. However, anyone in the partner's company who possesses the AWS security credentials could publish messages to the topic. This example assumes that you want to limit access to a particular person (or application). To do this you need to treat the partner like a user within your own company, and use a IAM policy instead of an Amazon SNS policy.

For this example, we create a group called `WidgetCo` that represents the partner company; we create a user for the specific person (or application) at the partner company who needs access; and then we put the user in the group.

We then attach a policy that grants the group `Publish` access on the specific topic named *WidgetPartnerTopic*.

We also want to prevent the `WidgetCo` group from doing anything else with topics, so we add a statement that denies permission to any Amazon SNS actions other than `Publish` on any topics other than `WidgetPartnerTopic`. This is necessary only if there's a broad policy elsewhere in the system that grants users wide access to Amazon SNS.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
  },
  {
    "Effect": "Deny",
    "NotAction": "sns:Publish",
    "NotResource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
  }
]
```

```
}
```

Using Temporary Security Credentials with Amazon SNS

In addition to creating IAM users with their own security credentials, IAM also enables you to grant temporary security credentials to any user allowing this user to access your AWS services and resources. You can manage users who have AWS accounts; these users are IAM users. You can also manage users for your system who do not have AWS accounts; these users are called federated users. Additionally, "users" can also be applications that you create to access your AWS resources.

You can use these temporary security credentials in making requests to Amazon SNS. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials Amazon SNS denies the request.

For more information about IAM support for temporary security credentials, go to [Granting Temporary Access to Your AWS Resources](#) in *Using IAM*.

Example Using Temporary Security Credentials to Authenticate an Amazon SNS Request

The following example demonstrates how to obtain temporary security credentials to authenticate an Amazon SNS request.

```
http://sns.us-east-2.amazonaws.com/  
?Name=My-Topic  
&Action=CreateTopic  
&Signature=gfzIF53exFVdpSNb8AiWn3Lv%2FNYXh6S%2Br3yySK70oX4%3D  
&SignatureVersion=2  
&SignatureMethod=HmacSHA256  
&Timestamp=2010-03-31T12%3A00%3A00.000Z  
&SecurityToken=SecurityTokenValue  
&AWSSecretKeyId=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 policy information.

Name	Maximum Quota
Bytes	30 kb
Statements	100
Principals	1 to 200 (0 is invalid.)

Name	Maximum Quota
Resource	1 (0 is invalid. The value must match the ARN of the policy's topic.)

Valid Amazon SNS Policy Actions

Amazon SNS supports the actions shown in the following table.

Action	Description
sns:AddPermission	Grants permission to add permissions to the topic policy.
sns:DeleteTopic	Grants permission to delete a topic.
sns:GetTopicAttributes	Grants permission to receive all of the topic attributes.
sns:ListSubscriptionsByTopic	Grants permission to retrieve all the subscriptions to a specific topic.
sns:Publish	Grants permission to publish to a topic or endpoint. For more information, see Publish in the Amazon Simple Notification Service API Reference
sns:RemovePermission	Grants permission to remove any permissions in the topic policy.
sns:SetTopicAttributes	Grants permission to set a topic's attributes.
sns:Subscribe	Grants permission to subscribe to a topic.

Service-Specific Keys

Amazon SNS uses the following service-specific keys. You can use these in policies that restrict access to `Subscribe` requests.

- **sns:Endpoint**—The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [Example Policies for Amazon SNS \(p. 206\)](#)) 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 \(p. 206\)](#)) to restrict publication to specific delivery protocols (for example, `https`).

Important

When you use a policy to control access by `sns:Endpoint`, be aware that DNS issues might affect the endpoint's name resolution in the future.

Logging and Monitoring in Amazon SNS

This section provides information about logging and monitoring Amazon SNS topics.

Topics

- [Logging Amazon Simple Notification Service API Calls Using AWS CloudTrail \(p. 210\)](#)
- [Monitoring Amazon SNS Topics Using CloudWatch \(p. 213\)](#)

Logging Amazon Simple Notification Service API Calls Using AWS 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`
- `ConfirmSubscription`
- `CreatePlatformApplication`
- `CreatePlatformEndpoint`
- `CreateTopic`
- `DeleteEndpoint`
- `DeletePlatformApplication`
- `DeleteTopic`
- `GetEndpointAttributes`
- `GetPlatformApplicationAttributes`
- `GetSMSAttributes`
- `GetSubscriptionAttributes`
- `GetTopicAttributes`
- `ListEndpointsByPlatformApplication`

- [ListPhoneNumbersOptedOut](#)
- [ListPlatformApplications](#)
- [ListSubscriptions](#)
- [ListSubscriptionsByTopic](#)
- [ListTopics](#)
- [RemovePermission](#)
- [SetEndpointAttributes](#)
- [SetPlatformApplicationAttributes](#)
- [SetSubscriptionAttributes](#)
- [SetTopicAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Note

When you are not logged in to Amazon Web Services (unauthenticated mode) and either the [ConfirmSubscription](#) or [Unsubscribe](#) actions are invoked, then they will not be logged to CloudTrail. Such as, when you choose the provided link in an email notification to confirm a pending subscription to a topic, the [ConfirmSubscription](#) action is invoked in unauthenticated mode. In this example, the [ConfirmSubscription](#) action would not be logged to CloudTrail.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Example: Amazon SNS Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the [ListTopics](#), [CreateTopic](#), and [DeleteTopic](#) actions.

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
```



```

"eventTime": "2014-09-30T00:00:00Z",
"eventSource": "sns.amazonaws.com",
"eventName": "ListTopics",
"awsRegion": "us-west-2",
"sourceIPAddress": "127.0.0.1",
"userAgent": "aws-sdk-java/unknown-version",
"requestParameters": {
  "nextToken": "ABCDEF1234567890EXAMPLE=="
},
"responseElements": null,
"requestID": "example1-b9bb-50fa-abdb-80f274981d60",
"eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
},
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "userName": "Bob",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2014-09-30T00:00:00Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "CreateTopic",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version",
  "requestParameters": {
    "name": "hello"
  },
  "responseElements": {
    "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
  },
  "requestID": "example7-5cd3-5323-8a00-f1889011fee9",
  "eventID": "examplec-4f2f-4625-8378-130ac89660b1",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
},
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "userName": "Bob",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2014-09-30T00:00:00Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "DeleteTopic",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
  },
  "responseElements": null,
  "requestID": "example5-4faa-51d5-aab2-803a8294388d",
  "eventID": "example8-6443-4b4d-abfd-1b867280d964",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}

```

```
}  
  ],  
}
```

Monitoring Amazon SNS Topics Using CloudWatch

Amazon SNS and 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. 214). For more information about Amazon SNS push notifications, see [Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber \(Mobile Push\)](#) (p. 119).

The metrics you configure with CloudWatch for your Amazon SNS topics are automatically collected and pushed to CloudWatch every five minutes. 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.

Note

There is no charge for the Amazon SNS metrics reported in CloudWatch; they are provided as part of the Amazon SNS service.

View CloudWatch Metrics for Amazon SNS

You can monitor metrics for Amazon SNS using the CloudWatch console, CloudWatch's own command line interface (CLI), or programmatically using the CloudWatch API. The following procedures show you how to access the metrics using the AWS Management Console.

To view metrics using the CloudWatch console

1. Sign in to the [CloudWatch console](#).
2. On the navigation panel, choose **Metrics**.
3. On the **All metrics** tab, choose **SNS**, and then choose one of the following dimensions:
 - **Country, SMS Type**
 - **PhoneNumber**
 - **Topic Metrics**
 - **Metrics with no dimensions**
4. To view more detail, choose a specific item. For example, if you choose **Topic Metrics** and then choose **NumberOfMessagesPublished**, the average number of published Amazon SNS messages for a five-minute period throughout the time range of 6 hours is displayed.

Set CloudWatch Alarms for Amazon SNS Metrics

CloudWatch also allows you to set alarms when a threshold is met for a metric. For example, you could set an alarm for the metric, **NumberOfNotificationsFailed**, so that when your specified threshold number is met within the sampling period, then an email notification would be sent to inform you of the event.

To set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. Choose **Alarms**, and then choose the **Create Alarm** button. This launches the **Create Alarm** wizard.
3. Scroll through the Amazon SNS metrics to locate the metric you want to place an alarm on. Select the metric to create an alarm on and choose **Continue**.
4. Fill in the **Name**, **Description**, **Threshold**, and **Time** values for the metric, and then choose **Continue**.
5. Choose **Alarm** as the alarm state. If you want CloudWatch to send you an email when the alarm state is reached, choose either an existing Amazon SNS topic or choose **Create New Email Topic**. If you choose **Create New Email Topic**, you can set the name and email addresses for a new topic. This list will be saved and appear in the drop-down box for future alarms. Choose **Continue**.

Note

If you use **Create New Email Topic** to create a new Amazon SNS topic, the email addresses must be verified before they will receive notifications. Emails are sent only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they will not receive a notification.

6. At this point, the **Create Alarm** wizard gives you a chance to review the alarm you're about to create. If you need to make any changes, you can use the **Edit** links on the right. Once you are satisfied, choose **Create Alarm**.

For more information about using CloudWatch and alarms, see the [CloudWatch Documentation](#).

Amazon SNS Metrics

Amazon SNS sends the following metrics to CloudWatch.

Metric	Description
NumberOfMessagesPublished	<p>The number of messages published to your Amazon SNS topics.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum</p>
NumberOfNotificationsDelivered	<p>The number of messages successfully delivered from your Amazon SNS topics to subscribing endpoints.</p> <p>For a delivery attempt to succeed, the endpoint's subscription must accept the message. A subscription accepts a message if a.) it lacks a filter policy or b.) its filter policy includes attributes that match those assigned to the message. If the subscription rejects the message, the delivery attempt isn't counted for this metric.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum</p>
NumberOfNotificationsFailed	<p>The number of messages that Amazon SNS failed to deliver.</p> <p>For Amazon SQS, email, SMS, or mobile push endpoints, the metric increments by 1 when Amazon SNS stops attempting message deliveries. For HTTP or HTTPS endpoints, the metric includes every failed delivery attempt, including retries that follow the initial attempt. For all other endpoints, the count increases by</p>

Metric	Description
	<p>1 when the message fails to deliver (regardless of the number of attempts).</p> <p>This metric does not include messages that were rejected by subscription filter policies.</p> <p>You can control the number of retries for HTTP endpoints. For more information, see Message Delivery Retries (p. 58).</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFilteredOut	<p>The number of messages that were rejected by subscription filter policies. A filter policy rejects a message when the message attributes don't match the policy attributes.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFilteredOut-InvalidAttributes	<p>The number of messages that were rejected by subscription filter policies because the messages' attributes are invalid – for example, because the attribute JSON is incorrectly formatted.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFilteredOut-NoMessageAttributes	<p>The number of messages that were rejected by subscription filter policies because the messages have no attributes.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsRedrivenToDlq	<p>The number of messages that have been moved to a dead-letter queue.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
NumberOfNotificationsFailedToRedriveToDlq	<p>The number of messages that couldn't be moved to a dead-letter queue.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average</p>
PublishSize	<p>The size of messages published.</p> <p>Units: <i>Bytes</i></p> <p>Valid Statistics: Minimum, Maximum, Average and Count</p>

Metric	Description
SMSThMonthToDateSpentUSD	<p>The charges you have accrued since the start of the current calendar month for sending SMS messages.</p> <p>You can set an alarm for this metric to know when your month-to-date charges are close to the monthly SMS spend quota for your account. When Amazon SNS determines that sending an SMS message would incur a cost that exceeds this quota, it stops publishing SMS messages within minutes.</p> <p>For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see Setting SMS Messaging Preferences (p. 148).</p> <p>Units: <i>USD</i></p> <p>Valid Statistics: Maximum</p>
SMSSuccessRate	<p>The rate of successful SMS message deliveries.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Sum, Average, Data Samples</p>

Dimensions for Amazon Simple Notification Service Metrics

Amazon Simple Notification Service sends the following dimensions to CloudWatch.

Dimension	Description
Application	Filters on application objects, which represent an app and device registered with one of the supported push notification services, such as APNs and FCM.
Application,Platform	Filters on application and platform objects, where the platform objects are for the supported push notification services, such as APNs and FCM.
Country	Filters on the destination country or region of an SMS message. The country or region is represented by its ISO 3166-1 alpha-2 code.
Platform	Filters on platform objects for the push notification services, such as APNs and FCM.
TopicName	Filters on Amazon SNS topic names.
SMSType	Filters on the message type of SMS message. Can be <i>promotional</i> or <i>transactional</i> .

Compliance Validation for Amazon SNS

Third-party auditors assess the security and compliance of Amazon SNS as part of multiple AWS compliance programs, including the Health Insurance Portability and Accountability Act (HIPAA).

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Amazon SNS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon SNS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures. For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Infrastructure Security in Amazon SNS

As a managed service, Amazon SNS is protected by the AWS global network security procedures described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

Use AWS API actions to access Amazon SNS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

You must sign requests using both an access key ID and a secret access key associated with an IAM principal. Alternatively, you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials for signing requests.

You can call these API actions from any network location, but Amazon SNS supports resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon SNS policies to control access from specific Amazon VPC endpoints or specific VPCs. This effectively isolates network access to a given Amazon SNS queue from only the specific VPC within the AWS network. For more information, see [Restrict Publication to an Amazon SNS Topic Only from a Specific VPC Endpoint](#) (p. 202).

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 \(p. 218\)](#)
- [Implement Least-Privilege Access \(p. 218\)](#)
- [Use IAM Roles for Applications and AWS Services Which Require Amazon SNS Access \(p. 219\)](#)
- [Implement Server-Side Encryption \(p. 219\)](#)
- [Enforce Encryption of Data in Transit \(p. 219\)](#)
- [Consider Using VPC Endpoints to Access Amazon SNS \(p. 219\)](#)

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 \(p. 188\)](#)

- [Amazon SNS API Permissions: Actions and Resources Reference \(p. 208\)](#)

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 \(p. 180\)](#) and [Key Management \(p. 182\)](#).

Enforce Encryption of Data in Transit

Without HTTPS (TLS), a network-based attacker can eavesdrop on network traffic or manipulate it using an attack such as man-in-the-middle. Allow only encrypted connections over HTTPS (TLS) using the `aws:SecureTransport` condition in the topic policy to force requests to use SSL.

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 an Amazon VPC Endpoint for Amazon SNS \(p. 186\)](#) and [Creating an Amazon VPC Endpoint Policy for Amazon SNS \(p. 187\)](#).

Amazon SNS Release Notes

The following table lists Amazon SNS feature releases and improvements. For changes to the *Amazon Simple Notification Service Developer Guide*, see [Amazon SNS Document History \(p. 223\)](#).

Date	Feature Release
November 14, 2019	<p>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.</p> <p>Note</p> <ul style="list-style-type: none"> The Amazon SNS subscription and Amazon SQS queue must be under the same AWS account and Region. Currently, you can't use an Amazon SQS FIFO queue as a dead-letter queue for an Amazon SNS subscription. To use an encrypted Amazon SQS queue as a dead-letter queue, you must use a custom CMK with a key policy that grants the Amazon SNS service principal access to AWS KMS API actions. For more information, see Encryption at Rest (p. 180) in this guide and Protecting Amazon SQS Data Using Server-Side Encryption (SSE) and AWS KMS in the <i>Amazon Simple Queue Service Developer Guide</i>. <p>For more information, see the following:</p> <ul style="list-style-type: none"> Amazon SNS Dead-Letter Queues (p. 62) Configuring an Amazon SNS Dead-Letter Queue for an Amazon SNS Subscription (p. 17) Message Delivery Retries (p. 58) The <code>RedrivePolicy</code> request parameter attribute of the following API actions: <ul style="list-style-type: none"> GetSubscriptionAttributes SetSubscriptionAttributes Subscribe The <code>NumberOfNotificationsRedrivenToDlq</code> and <code>NumberOfNotificationsFailedToRedriveToDlq</code> metrics in the Amazon SNS Metrics (p. 214) section.
October 18, 2019	You can specify a custom APNs header value. For more information, see Sending Messages to APNs as Alert or Background Notifications (p. 132) .
September 10, 2019	Amazon SNS supports the <code>apns-push-type</code> header field for mobile notifications sent through APNs. For more information, see Sending Messages to APNs as Alert or Background Notifications (p. 132) .
July 24, 2019	You can troubleshoot messages passing through Amazon SNS topics using AWS X-Ray. For more information, see Troubleshooting Amazon Simple Notification Service Topics Using AWS X-Ray (p. 179) section.
July 5, 2019	<ul style="list-style-type: none"> In addition to multiple strings, Amazon SNS allows blacklisting of multiple numeric values. For more information, see Anything-But Matching (Blacklisting) (p. 72).

Date	Feature Release
	<ul style="list-style-type: none"> You can use the <code>exists</code> operator to check whether an incoming message has an attribute whose key is listed in the filter policy. For more information, see Attribute Key Matching (p. 72).
May 16, 2019	<ul style="list-style-type: none"> You can track your Amazon SNS resources (for example, for cost allocation) by adding, removing, and listing metadata tags for Amazon SNS topics using the AWS Management Console. For more information, see Amazon SNS Tags (p. 87) and the Listing, Adding, and Removing Tags for an Amazon SNS Topic (p. 16) and Creating an Amazon SNS Topic (p. 8) tutorials. <p>Note Currently, tag-based access control isn't available.</p> <ul style="list-style-type: none"> You can use a single Amazon SNS API call, AWS SDK function, or AWS CLI command to simultaneously create a topic and specify its tags. For more information, see the <code>Tags</code> request parameter attribute of the <code>CreateTopic</code> API action.
April 26, 2019	<p>You can track your Amazon SNS resources (for example, for cost allocation) by adding, removing, and listing metadata tags for Amazon SNS topics using the TagResource, UntagResource, and ListTagsForResource API actions or AWS SDKs. For more information, see Amazon SNS Tags (p. 87) and the Listing, Adding, and Removing Tags for an Amazon SNS Topic (p. 16) tutorial.</p>
April 4, 2019	<p>You can create Amazon VPC endpoint policies for Amazon SNS. For more information, see Creating an Amazon VPC Endpoint Policy for Amazon SNS (p. 187).</p>
March 25, 2019	<p>You can use Amazon SNS for system-to-system messaging with AWS Event Fork Pipelines as a subscriber (p. 114). You can deploy any of the pipelines from the AWS Event Fork Pipelines suite in the AWS Serverless Application Repository independently, as your architecture requires. For more information, see the following:</p> <ul style="list-style-type: none"> Deploying and Testing the AWS Event Fork Pipelines Sample Application (p. 41) Subscribing AWS Event Fork Pipelines to an Amazon SNS Topic (p. 48) <ul style="list-style-type: none"> To Deploy and Subscribe the Event Storage and Backup Pipeline (p. 48) To Deploy and Subscribe the Event Search and Analytics Pipeline (p. 50) To Deploy and Subscribe the Event Replay Pipeline (p. 52)
March 21, 2019	<p>You can publish to your Amazon SNS topics privately from Amazon Virtual Private Cloud in the AWS GovCloud (US-West) Region. For more information, see Internetwork Traffic Privacy (p. 185).</p>

Date	Feature Release
March 6, 2019	<p>The Amazon SNS console user interface has been updated and improved. The following related changes are reflected in new and rewritten sections in this guide:</p> <ul style="list-style-type: none"> • Setting Up Access for Amazon SNS (p. 2) • Getting Started with Amazon SNS (p. 4) • Amazon SNS Tutorials (p. 8) <ul style="list-style-type: none"> • Creating an Amazon SNS Topic (p. 8) • Enabling Server-Side Encryption (SSE) for an Amazon SNS Topic (p. 27) • Enabling Server-Side Encryption (SSE) for an Amazon SNS Topic with an Encrypted Amazon SQS Queue Subscribed (p. 30) • Subscribing an Endpoint to an Amazon SNS Topic (p. 11) • Publishing a Message to an Amazon SNS Topic (p. 13) • Deleting an Amazon SNS Subscription or Topic (p. 20)
February 5, 2019	<p>Amazon SNS allows blacklisting multiple strings. For more information, see Anything-But Matching (Blacklisting) (p. 70).</p>
November 15, 2018	<p>Server-side encryption (SSE) for Amazon SNS lets you protect the contents of messages in Amazon SNS topics using keys managed in the AWS Key Management Service (AWS KMS). For more information about server-side encryption and how to get started using it, see Encryption at Rest (p. 180) and Enabling Server-Side Encryption (SSE) for an Amazon SNS Topic (p. 27).</p> <p>SSE adds the ability to set the <code>KmsMasterKeyId</code> attribute using the CreateTopic and SetTopicAttributes API actions.</p> <p>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.</p>
May 30, 2018	<p>Amazon SNS sends metrics for messages that are filtered by subscription filter policies to Amazon CloudWatch. For more information, see Amazon SNS Metrics (p. 214).</p>
April 10, 2018	<p>You can publish to your Amazon SNS topics privately from Amazon Virtual Private Cloud. For more information, see Internetwork Traffic Privacy (p. 185).</p>
April 10, 2018	<p>To stop filtering messages, you can remove a filter policy assigned to a subscription. For more information, see Removing a Subscription Filter Policy (p. 76).</p>
March 12, 2018	<p>In addition to exact matching with string values, filter policies support the following operations for matching message attributes: anything-but matching, prefix matching, exact numeric matching, and numeric range matching. For more information, see Message Filtering (p. 67).</p>
November 21, 2017	<p>Subscribers can apply filter policies to topic subscriptions to selectively receive messages published to Amazon SNS topics. For more information, see Message Filtering (p. 67).</p>

Date	Feature Release
March 31, 2017	Amazon SNS sends the <code>SMSMonthToDateSpentUSD</code> metric to CloudWatch. This metric shows the SMS message charges you accrued since the beginning of the current calendar month. For more information, see Amazon SNS Metrics (p. 214).
March 6, 2017	To send SMS messages using a persistent short code, you can reserve a dedicated short code assigned to your account and available exclusively to you. For more information, see Reserving a Dedicated Short Code for SMS Messaging (p. 170).
June 28, 2016	Amazon SNS supports SMS messaging to more than 200 countries and regions. You can send a message directly to a phone number, or you can send a message to multiple phone numbers simultaneously by subscribing those phone numbers to a topic and sending your message to the topic. For more information, see Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber (Send SMS) (p. 148).
June 15, 2015	Amazon SNS supports VoIP and Mac OS X push notifications.
February 5, 2015	Amazon SNS supports logging the delivery status of push notification messages.
October 9, 2014	Amazon SNS supports AWS CloudTrail.
August 19, 2014	Amazon SNS supports authenticated messages with Microsoft Push Notification Service for Windows Phone.
July 10, 2014	Amazon SNS supports setting a Time To Live (TTL) message attribute for mobile push notification messages.
June 12, 2014	Amazon SNS supports Baidu Cloud Push, Microsoft Push Notification Service for Windows Phone, and Windows Push Notification Services.
June 12, 2014	Message attributes let you provide structured metadata items about a message. For more information, see Amazon SNS Message Attributes (p. 64).
August 13, 2013	Amazon SNS supports sending notification messages directly to apps on mobile devices. For more information, see Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber (Mobile Push) (p. 119).

Amazon SNS Document History

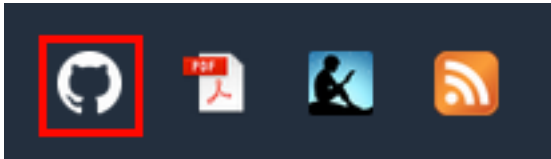
The following table lists changes to the *Amazon Simple Notification Service Developer Guide*. For Amazon SNS feature releases and improvements, see [Amazon SNS Release Notes](#) (p. 220).

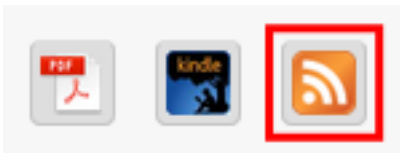
Date	Documentation Update
December 18, 2019	<ul style="list-style-type: none">Added the following sections:<ul style="list-style-type: none">Data Protection in Amazon SNS (p. 180)Data Encryption (p. 180)Logging and Monitoring in Amazon SNS (p. 209)Compliance Validation for Amazon SNS (p. 216)Resilience in Amazon SNS (p. 217)Infrastructure Security in Amazon SNS (p. 217)Amazon SNS Security Best Practices (p. 218)Renamed the following sections:

Date	Documentation Update
	<ul style="list-style-type: none"> • Encryption at Rest (p. 180) • Key Management (p. 182) • Internet Traffic Privacy (p. 185) • Identity and Access Management in Amazon SNS (p. 188) • Overview of Managing Access in Amazon SNS (p. 189) • Using Identity-Based Policies with Amazon SQS (p. 202) • Amazon SNS Troubleshooting (p. 179)
December 16, 2019	Rewrote the View CloudWatch Metrics for Amazon SNS (p. 213) section.
December 3, 2019	<ul style="list-style-type: none"> • Added the following statement to the To Create a Topic Using the AWS Management Console (p. 9) section: Important As a security precaution, Amazon SNS uses the <code>aws:sourceOwner</code> permission to limit access to the topic to the owner of the current AWS account. • Revised the information in the Sending JSON-Formatted Messages (p. 131) section. • Added the <code>verifyMessageSignatureURL()</code> method to the code example in the Example Code for an Amazon SNS Endpoint Java Servlet (p. 111) section.
December 2, 2019	<p>Rewrote the following sections:</p> <ul style="list-style-type: none"> • To Set the Topic Policy Using the AWS Management Console (p. 96) • To Add an Amazon SQS Queue Subscription to a Topic in Another AWS Account Using the AWS Management Console (p. 96) • To Confirm a Subscription Using the AWS Management Console (p. 97)
November 26, 2019	<ul style="list-style-type: none"> • Corrected the formatting for all JSON examples throughout this guide. • Revised the policy example in the Allow Any CloudWatch Alarm in an AWS Account to Publish to an Amazon SNS Topic in a Different AWS Account (p. 201) section.
November 25, 2019	<ul style="list-style-type: none"> • Revised the information in the following sections: <ul style="list-style-type: none"> • Amazon SNS Information in CloudTrail (p. 210) • User Notification Process Overview (p. 121) • Supported Regions and Countries (p. 171) • To Publish a Message to an Amazon SNS Topic Using the AWS Management Console (p. 13) • Added the Restrict Publication to an Amazon SNS Topic Only from a Specific VPC Endpoint (p. 202) section.
November 20, 2019	Revised the information in the Publish Messages to an Amazon SQS Queue (p. 200) section.
November 19, 2019	Updated the information for Japan in the Supported Regions and Countries (p. 171) section.
November 18, 2019	<ul style="list-style-type: none"> • Reorganized the subsections of the Amazon SNS Tutorials (p. 8) section. • Added the Prerequisites (p. 17) section.

Date	Documentation Update
November 14, 2019	<p>Added the following sections:</p> <ul style="list-style-type: none"> • Amazon SNS Dead-Letter Queues (p. 62) • Configuring an Amazon SNS Dead-Letter Queue for an Amazon SNS Subscription (p. 17) • Message Delivery Retries (p. 58)
November 12, 2019	<p>Revised the information in the following sections:</p> <ul style="list-style-type: none"> • Example Message with Attributes (p. 68)
November 8, 2019	<p>Revised the information in the Enable Compatibility between Event Sources from AWS Services and Encrypted Topics (p. 184) section.</p>
October 29, 2019	<p>Revised the information in the following sections:</p> <ul style="list-style-type: none"> • Verifying the Signatures of Amazon SNS Messages (p. 109) • How User Notifications Work (p. 120)
October 24, 2019	<ul style="list-style-type: none"> • Fixed the CLI example in the AWS CLI (p. 74) (Applying a Subscription Filter Policy) section. • Revised the information in the following sections: <ul style="list-style-type: none"> • Viewing Daily SMS Usage Reports (p. 164) • Sending Messages to an Application on Multiple Platforms (p. 132)
October 18, 2019	<ul style="list-style-type: none"> • Clarified the information regarding CloudWatch in the Enable Compatibility between Event Sources from AWS Services and Encrypted Topics (p. 184) section. • Updated the Reserved Message Attributes for Mobile Push Notifications (p. 65) page with new reserved message attributes for mobile push notifications. • Rewrote the Sending Messages to APNs as Alert or Background Notifications (p. 132) section. • Created the Specifying Custom APNs Header Values (p. 132) section.
October 16, 2019	<ul style="list-style-type: none"> • Revised the information in the following sections: <ul style="list-style-type: none"> • Amazon SNS Message Attributes (p. 64) • Amazon SNS Large Payload and Raw Message Delivery (p. 87) • Enable Compatibility between Event Sources from AWS Services and Encrypted Topics (p. 184) • Corrected the following C# code in the AWS SDK Examples (p. 126) section: <pre> if (m.Count > 0 && m[0].Groups.Count > 1) { // The platform endpoint already exists for this token, but with // additional custom data that createEndpoint doesn't want to // overwrite. // Just use the existing platform endpoint. endpointArn = m[0].Groups[1].Value; } </pre>
October 10, 2019	<p>Fixed the code excerpt that initializes and uses the <code>SNSMessageAttributeClass</code> in the To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for .NET (p. 25) section.</p>

Date	Documentation Update
October 8, 2019	<ul style="list-style-type: none"> Fixed the JSON example in the Creating an Amazon VPC Endpoint Policy for Amazon SNS (p. 187) section. Fixed the JSON example in the Allow Any AWS Resource to Publish to a Topic (p. 200) section and clarified an edge case for the policy.
October 3, 2019	Revised the information in the following sections: <ul style="list-style-type: none"> How User Notifications Work (p. 120) Prerequisites for Amazon SNS User Notifications (p. 120) User Notification Process Overview (p. 121)
September 23, 2019	Revised the information in the following sections: <ul style="list-style-type: none"> Sending Messages to APNs as Alert or Background Notifications (p. 132) Example Log for Successful SMS Delivery (p. 163) Example Log for Failed SMS Delivery (p. 163)
September 10, 2019	<ul style="list-style-type: none"> Revised the information in the following sections: <ul style="list-style-type: none"> Send Custom Platform-Specific Payloads to Mobile Devices (p. 131) Filter Policy Constraints (p. 69) Added the Sending Messages to APNs as Alert or Background Notifications (p. 132) section.
August 29, 2019	Revised the information in the following sections: <ul style="list-style-type: none"> Allow Any AWS Resource to Publish to a Topic (p. 200) Viewing CloudWatch Logs (p. 162) Subscribing to Daily Usage Reports (p. 164)
August 26, 2019	<ul style="list-style-type: none"> Added a note to the Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber (Mobile Push) (p. 119) to clarify which Regions allow the creation of mobile applications. Updated the countries and regions table in the Supported Regions and Countries (p. 171) section.
August 19, 2019	Added the Allow Any CloudWatch Alarm in an AWS Account to Publish to an Amazon SNS Topic in a Different AWS Account (p. 201) section.
August 16, 2019	Revised the information in the following sections: <ul style="list-style-type: none"> Example Policies for Amazon SNS (p. 206) Using Temporary Security Credentials with Amazon SNS (p. 208) Policy Quotas (p. 208) Example Cases for Amazon SNS Access Control (p. 198) <ul style="list-style-type: none"> Grant AWS Account Access to a Topic (p. 199) Limit Subscriptions to HTTPS (p. 199) Publish Messages to an Amazon SQS Queue (p. 200) Allow Any AWS Resource to Publish to a Topic (p. 200) Allow an Amazon S3 Bucket to Publish to a Topic (p. 201)
August 1, 2019	Corrected outdated links to the Amazon SNS console throughout this guide.

Date	Documentation Update
July 26, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• Example Message with Attributes (p. 68)• Viewing Daily SMS Usage Reports (p. 164)
July 24, 2019	<ul style="list-style-type: none">• Renamed the Amazon SNS Troubleshooting (p. 179) section.• Added the Troubleshooting Amazon Simple Notification Service Topics Using AWS X-Ray (p. 179) section.
July 19, 2019	Corrected the following statement throughout this guide: The AWS Support team provides an initial response to your request within 24 hours.
July 16, 2019	<p>In addition to HTML, PDF, and Kindle, the <i>Amazon Simple Notification Service Developer Guide</i> is available on GitHub. To leave feedback, choose the GitHub icon in the upper right-hand corner.</p> 
July 11, 2019	<ul style="list-style-type: none">• Revised the information in the following sections:<ul style="list-style-type: none">• Message Filtering (p. 67)• Attribute Key Matching (p. 72)• Corrected the C# code in the AWS SDK Examples (p. 126) section.
July 8, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• Anything-But Matching (Blacklisting) (p. 72)• Attribute Key Matching (p. 72)
July 5, 2019	Added the following sections: <ul style="list-style-type: none">• Anything-But Matching (Blacklisting) (p. 72)• Attribute Key Matching (p. 72)
June 23, 2019	Revised the information in the following section: <ul style="list-style-type: none">• TTL Message Attributes for Push Notification Services (p. 139)

Date	Documentation Update
June 22, 2019	<ul style="list-style-type: none">Restructured the following sections:<ul style="list-style-type: none">Amazon SNS Release Notes (p. 220)Amazon SNS Document History (p. 223)Revised the information in the following sections:<ul style="list-style-type: none">Send Custom Platform-Specific Payloads to Mobile Devices (p. 131)Sending JSON-Formatted Messages (p. 131)Sending Platform-Specific Messages (p. 132)Send a Direct Message to a Mobile Device (p. 131)Example Message with Attributes (p. 68) (code example)In addition to HTML, PDF, and Kindle, the <i>Amazon Simple Notification Service Developer Guide</i> release notes are available as an RSS feed. 
May 31, 2019	<p>Revised the information in the following sections:</p> <ul style="list-style-type: none">Topic Attributes (p. 56)Configuring Delivery Status Logging Using the AWS Management Console (p. 56)AWS Management Console (p. 73) (Applying a Subscription Filter Policy)AWS Management Console (p. 76) (Removing a Subscription Filter Policy)Enabling Raw Message Delivery Using the AWS Management Console (p. 87)Configuring Amazon SNS with Lambda Endpoints using the AWS Management Console (p. 89)Subscribe the Queue to the Topic (p. 92)Setting SMS Messaging Preferences Using the AWS Management Console (p. 148)Available Application Events (p. 136)AWS Management Console (p. 137) (Sending Mobile Push Notifications)TTL Message Attributes for Push Notification Services (p. 139)
May 30, 2019	<ul style="list-style-type: none">Corrected the syntax of C# code examples throughout this guide.Corrected <code>token.*</code> to <code>[Tt]oken.*</code> in the code examples in the AWS SDK Examples (p. 126) section.
May 28, 2019	<ul style="list-style-type: none">Revised the information in the following section:<ul style="list-style-type: none">Amazon SNS Message Attributes (p. 64)To Publish a Message to an Amazon SNS Topic Using the AWS Management Console (p. 13)Reserved Message Attributes for Mobile Push Notifications (p. 65)Added the Publishing a Message with Attributes to an Amazon SNS Topic (p. 22) section.

Date	Documentation Update
May 15, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• What is Amazon Simple Notification Service? (p. 1)• How User Notifications Work (p. 120)• Using Amazon SNS for User Notifications (p. 119)
April 3, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• Deploying AWS Event Fork Pipelines (p. 117)• Deploying and Testing the AWS Event Fork Pipelines Sample Application (p. 41)• Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber (Mobile Push) (p. 119)• Using Amazon SNS Mobile Push (p. 121)• Add Device Tokens or Registration IDs (p. 122)• Subscribe the HTTP/HTTPS Endpoint to the Amazon SNS Topic (p. 107)
March 28, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• How AWS Event Fork Pipelines Works (p. 115)• Example AWS Event Fork Pipelines Use Case (p. 42)• To Verify the Execution of the Sample Application and Its Pipelines (p. 45)• To Simulate an Issue and Replay Events for Recovery (p. 46)
March 26, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• With a Fork Pipeline as a Subscriber (p. 114)• To Deploy and Subscribe the Event Replay Pipeline (p. 52)• Getting Started with Amazon SNS (p. 4)• With an HTTP/S Endpoint as a Subscriber (p. 103)• Sending Amazon SNS Messages to an Amazon SQS Queue in a Different Account (p. 95)• Subscribing to Daily Usage Reports (p. 164)• Supported Regions and Countries (p. 171)• Limit Subscriptions to HTTPS (p. 199)
March 18, 2019	Revised the information in the following sections: <ul style="list-style-type: none">• Getting Started with Amazon SNS (p. 4)• Creating an Amazon SNS Topic (p. 8)• Enabling Server-Side Encryption (SSE) for an Amazon SNS Topic (p. 27)• Publishing a Message to an Amazon SNS Topic (p. 13)• With an Amazon SQS Queue as a Subscriber (p. 90)
March 15, 2019	Revised the information in the Key Terms (p. 181) (SSE) section.
March 1, 2019	Removed outdated and deprecated Getting Started guides for mobile push.
February 4, 2019	Made a minor correction in the Sending Amazon SNS Messages to an Amazon SQS Queue in a Different Account (p. 95) section.

Date	Documentation Update
January 31, 2019	<ul style="list-style-type: none">Removed the "Removing a Filter Policy Using AWS CloudFormation" section.Rewrote the following sections:<ul style="list-style-type: none">Message Filtering (p. 67)Amazon SNS Subscription Filter Policies (p. 67)Subscription Filter Policies as Java Collections (p. 77)
February 4, 2019	Made a minor correction in the Sending Amazon SNS Messages to an Amazon SQS Queue in a Different Account (p. 95) section.
January 30, 2019	Clarified the information in the following sections: <ul style="list-style-type: none">Amazon SNS Large Payload and Raw Message Delivery (p. 87)Application and System Alerts (p. 55)Internetwork Traffic Privacy (p. 185)
December 12, 2018	<ul style="list-style-type: none">Renamed the Amazon SNS Message Delivery Status (p. 55) section.Made minor formatting changes in the following sections:<ul style="list-style-type: none">With a Lambda Function as a Subscriber (p. 89)Internetwork Traffic Privacy (p. 185)Tutorial: Publishing Amazon SNS Messages Privately from Amazon VPC (p. 32)Creating an Amazon VPC Endpoint for Amazon SNS (p. 186)
November 20, 2018	Revised the Encryption at Rest (p. 180) section.

Date	Documentation Update
October 25, 2018	<ul style="list-style-type: none">Added the following sections:<ul style="list-style-type: none">Using Amazon SNS for System-to-System Messaging (p. 89)Using Amazon SNS for User Notifications (p. 119)How Amazon SNS Works (p. 54)Amazon SNS Troubleshooting (p. 179)Amazon SNS Security (p. 180)Reorganized the following sections:<ul style="list-style-type: none">Getting Started with Amazon SNS (p. 4)With a Lambda Function as a Subscriber (p. 89)With an Amazon SQS Queue as a Subscriber (p. 90)With a Lambda Function as a Subscriber (p. 89)With an Amazon SQS Queue as a Subscriber (p. 90)With an HTTP/S Endpoint as a Subscriber (p. 103)Using Amazon SNS for User Notifications with a Mobile Application as a Subscriber (Mobile Push) (p. 119)Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber (Send SMS) (p. 148)Common Amazon SNS Scenarios (p. 54)Amazon SNS Message Delivery Status (p. 55)Amazon SNS Message Attributes (p. 64)Amazon SNS Message and JSON Formats (p. 80)Amazon SNS Large Payload and Raw Message Delivery (p. 87)Identity and Access Management in Amazon SNS (p. 188)Overview of Managing Access in Amazon SNS (p. 189)Internetwork Traffic Privacy (p. 185)Monitoring Amazon SNS Topics Using CloudWatch (p. 213)Logging Amazon Simple Notification Service API Calls Using AWS CloudTrail (p. 210)
October 8, 2018	Corrected the example headers and requests in the Amazon SNS Message and JSON Formats (p. 80) section.
October 5, 2018	<ul style="list-style-type: none">Clarified the information in the With an Amazon SQS Queue as a Subscriber (p. 90) section.Corrected the example policies in the following sections:<ul style="list-style-type: none">Give Permission to the Topic to Send Messages to the Queue (p. 91)Adding a Policy to an IAM User or Group (p. 93)Queue Owner Creates Subscription (p. 96)
September 20, 2018	Made minor corrections throughout this guide.
September 19, 2018	Updated all the links in the Platform Response Codes (p. 136) section.

Date	Documentation Update
September 17, 2018	<ul style="list-style-type: none">Corrected HTTP request examples to use <code>application/json</code> in the following sections:<ul style="list-style-type: none">With an HTTP/S Endpoint as a Subscriber (p. 103)Make Sure Your Endpoint is Ready to Process Amazon SNS Messages (p. 104)Made other minor corrections throughout this guide.
September 13, 2018	<ul style="list-style-type: none">Corrected the information in the Reserving a Dedicated Short Code for SMS Messaging (p. 170) section.Corrected the code example to include <code>throw new SecurityException("Signature verification failed.");</code> in the correct place in the Example Code for an Amazon SNS Endpoint Java Servlet (p. 111) section.
September 12, 2018	<ul style="list-style-type: none">Corrected the information in the following section:<ul style="list-style-type: none">The Interplay of Explicit and Default Denials (p. 196)Configuring Amazon SNS with Lambda Endpoints using the AWS Management Console (p. 89)With an HTTP/S Endpoint as a Subscriber (p. 103)Added instructions for setting the policy to the Queue Owner Creates Subscription (p. 96) section.Fixed broken links in the following sections:<ul style="list-style-type: none">Platform Response Codes (p. 136)Sending Platform-Specific Messages (p. 132)With an HTTP/S Endpoint as a Subscriber (p. 103)
September 6, 2018	Corrected the information in the Creating the Endpoint (p. 186) section.
September 5, 2018	<ul style="list-style-type: none">Clarified the information in the Daily Usage Report Information (p. 164) section.Corrected the code example to include <code>.collect(Collectors.joining(", "));</code> in the Subscription Filter Policies as Java Collections (p. 77) section, to prevent <code>Attribute.List</code> from returning an incorrect policy format when <code>formatFilterPolicy()</code> is called.Corrected the code example to include <code>encryptedVal.add(false);</code> in the To Publish a Message with Attributes to an Amazon SNS Topic Using the AWS SDK for .NET (p. 25) section.Corrected the maximum push notification message size to 4,096 bytes.Corrected the ISO code for Mongolia to <code>MN</code> in the Supported Regions and Countries (p. 171) section.Made other minor corrections throughout this guide.
September 4, 2018	<ul style="list-style-type: none">Clarified that Amazon SNS isn't currently compatible with Amazon SQS queues in the With an Amazon SQS Queue as a Subscriber (p. 90) section.Clarified that message attributes (p. 64) are sent only when the message structure is String, not JSON.Updated the content in the Subscribe the Queue to the Topic (p. 92) section.Made a minor correction in the View CloudWatch Metrics for Amazon SNS (p. 213) section.Replaced all instances of <i>click</i> with <i>choose</i> for a device-agnostic user experience.Fixed broken links throughout this guide.

Date	Documentation Update
August 30, 2018	<ul style="list-style-type: none">• Rewrote the Logging Amazon Simple Notification Service API Calls Using AWS CloudTrail (p. 210) and Creating the Endpoint (p. 186) sections.• Made a minor correction in the Step 2: Create the AWS Resources (p. 34) section.• Added the AWS CloudFormation (p. 75) section.
August 22, 2018	Reconfigured and restructured parts of this guide.
August 7, 2018	Added code examples that show how to use Amazon SNS with the AWS SDK for .NET to various sections.
June 8, 2018	Added the Tutorial: Publishing Amazon SNS Messages Privately from Amazon VPC (p. 32) section.
September 23, 2015	Added the Create a Platform Endpoint and Manage Device Tokens (p. 124) topic.
September 23, 2015	Added the Application Event Notifications (p. 136) topic.
April 9, 2015	Added the With a Lambda Function as a Subscriber (p. 89) section.
April 9, 2015	Added the Amazon SNS Message Delivery Status (p. 55) section.
February 5, 2015	Added the Using Amazon SNS Application Attributes for Message Delivery Status (p. 133) section.
October 9, 2014	Added the Logging Amazon Simple Notification Service API Calls Using AWS CloudTrail (p. 210) section.
October 9, 2014	Added the User Notification Process Overview (p. 121) section.
July 10, 2014	Added the Using the Amazon SNS Time To Live (TTL) Message Attribute for Mobile Push Notifications (p. 138) section.
April 23, 2014	Added a section about using the AWS SDK for Java with Amazon SNS.
December 17, 2013	Added the Send Custom Platform-Specific Payloads to Mobile Devices (p. 131) section.
May 1, 2013	This is the first release of the <i>Amazon Simple Notification Service Developer Guide</i> .

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.