
Amazon Simple Notification Service

Developer Guide

API Version 2010-03-31



Amazon Simple Notification Service: Developer Guide

Copyright © 2016 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
Are You a First-Time Amazon Simple Notification Service User?	2
Beyond the Getting Started Section	2
Accessing Amazon SNS	2
Common Scenarios	3
Fanout	3
Application and System Alerts	4
Push Email and Text Messaging	4
Mobile Push Notifications	4
Getting Started	5
Before You Begin	5
Create a Topic	6
Subscribe to a Topic	6
Publish to a Topic	7
Create Different Messages for Each Protocol	8
Clean Up	9
Using the SDK for Java	9
Managing Access	12
Overview	12
When to Use Access Control	13
Key Concepts	13
Architectural Overview	15
Using the Access Policy Language	17
Evaluation Logic	18
Example Cases for Amazon SNS Access Control	21
Special Information for Amazon SNS Policies	26
Amazon SNS Policy Limits	26
Valid Amazon SNS Policy Actions	26
Amazon SNS Keys	26
Controlling User Access to Your AWS Account	27
IAM and Amazon SNS Policies Together	27
Amazon SNS ARNs	30
Amazon SNS Actions	31
Amazon SNS Keys	31
Example Policies for Amazon SNS	32
Using Temporary Security Credentials	34
Amazon SNS Mobile Push	36
Overview	36
Prerequisites	37
Mobile Push High Level Steps	38
Step 1: Request Credentials from Mobile Platforms	38
Step 2: Request Token from Mobile Platforms	38
Step 3: Create Platform Application Object	39
Step 4: Create Platform Endpoint Object	39
Step 5: Publish Message to Mobile Endpoint	39
Getting Started with ADM	39
ADM Prerequisites	40
Step 1: Create a Kindle Fire App with the ADM Service Enabled	40
Step 2: Obtain a Client ID and Client Secret	40
Step 3: Obtain an API Key	41
Step 4: Obtain a Registration ID	41
Step 5: Sending a Message to a Kindle Fire app using Amazon SNS and ADM	42
Getting Started with APNS	44
APNS Prerequisites	44
Step 1: Create an iOS App	44

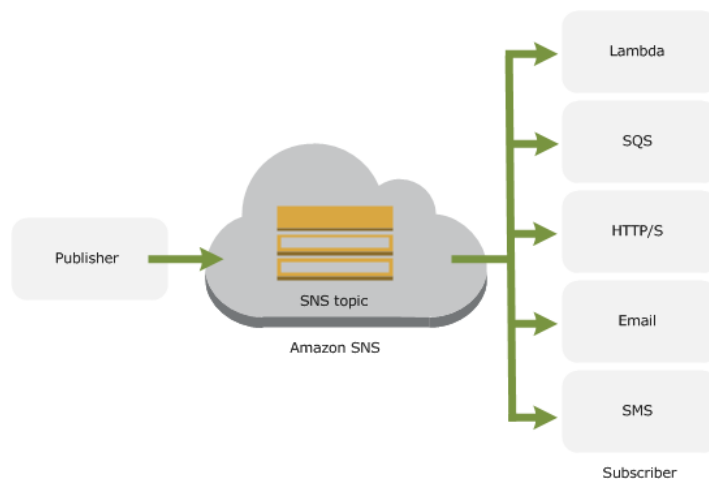
Step 2: Obtain an APNS SSL Certificate	45
Step 3: Obtain the App Private Key	45
Step 4: Verify the Certificate and App Private Key	46
Step 5: Obtain a Device Token	46
Next Steps	47
Send a message to an iOS app	47
Send a message to a VoIP app	49
Send a message to a Mac OS X app	49
Getting Started with Baidu	50
Baidu Prerequisites	50
Step 1: Create a Baidu Account	51
Step 2: Register as a Baidu Developer	52
Step 3: Create a Baidu Cloud Push Project	56
Step 4: Download and Install the Android Demo App	59
Step 5: Obtain a user Id and channel Id	63
Step 6: Send a Push Notification Message to a Mobile Endpoint using Amazon SNS and Baidu	63
Getting Started with GCM	67
GCM Prerequisites	68
Step 1: Create a Google API Project and Enable the GCM Service	68
Step 2: Obtain the Server API Key	68
Step 3: Obtain a Registration ID from GCM	69
Step 4: Send a Message to a Mobile Endpoint using GCM	70
Getting Started with MPNS	72
MPNS Prerequisites	72
Step 1: Set Up Your Windows Phone App to Receive Push Notifications Messages	73
Step 2: Get a Push Notification URI from MPNS	73
Step 3: Create a Windows Developer Account	73
Step 4: Upload TLS Certificate	73
Step 5: Send a Push Notification Message to a Windows Phone app using Amazon SNS and MPNS	73
Getting Started with WNS	75
WNS Prerequisites	76
Step 1: Set Up Your App to Receive Push Notifications Messages	76
Step 2: Get a Push Notification URI from WNS	76
Step 3: Get a Package Security Identifier from WNS	76
Step 4: Get a Secret Key from WNS	76
Step 5: Send a Push Notification Message to an App using Amazon SNS and WNS	77
Using Amazon SNS Mobile Push	78
Register Your Mobile App with AWS	78
Add Device Tokens or Registration IDs	80
Create a Platform Endpoint and Manage Device Tokens	83
Send a Direct Message to a Mobile Device	88
Send Messages to Mobile Devices Subscribed to a Topic	88
Send Custom Platform-Specific Payloads to Mobile Devices	88
Application Attributes for Message Delivery Status	90
Configuring Message Delivery Status Attributes with the AWS Management Console	91
Amazon SNS Message Delivery Status CloudWatch Log Examples	91
Configuring Message Delivery Status Attributes with the AWS SDKs	92
Platform Response Codes	93
Application Event Notifications	93
Available Application Events	93
How to Set Application Event Notifications	94
Amazon SNS TTL	95
TTL Message Attributes for Push Notification Services	96
Precedence Order for Determining TTL	96
Specifying TTL with the AWS Management Console	97
Specifying TTL with the AWS SDKs	97

Amazon SNS Mobile Push APIs	97
API Errors	99
Sending Messages to Amazon SQS Queues	106
Step 1. Get the ARN of the queue and the topic.	107
Step 2. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue	108
Step 3. Subscribe the queue to the Amazon SNS topic	109
Step 4. Give users permissions to the appropriate topic and queue actions	109
Adding a policy to an IAM user or group	110
Adding a policy to a topic or queue	110
Step 5. Test it	111
Sending Messages to a Queue in a Different Account	112
Queue Owner Creates Subscription	112
User Who Does Not Own the Queue Creates Subscription	114
Using an AWS CloudFormation Template to Create a Topic that Sends Messages to Amazon SQS Queues	115
Using an AWS CloudFormation Template to Set Up Topics and Queues Within an AWS Account	116
Sending and Receiving SMS Notifications	121
Task 1: Assign a Topic Display Name	122
Task 2: Subscribe to a Topic Using the SMS Protocol	123
Task 3: Publish a Message	124
Task 4: Cancel SMS Subscriptions	125
Sending Messages to HTTP/HTTPS Endpoints	127
Step 1: Make sure your endpoint is ready to process Amazon SNS messages	128
Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic	131
Step 3: Confirm the subscription	132
Step 4: Set the delivery retry policy for the subscription (optional)	132
Step 5: Give users permissions to publish to the topic (optional)	132
Step 6: Send messages to the HTTP/HTTPS endpoint	133
Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints	134
Applying Delivery Policies to Topics and Subscriptions	136
Setting the Maximum Receive Rate	137
Immediate Retry Phase	137
Pre-Backoff Phase	138
Backoff Phase	138
Post-Backoff Phase	139
Certificate Authorities for HTTPS Endpoints	140
Verifying Message Signatures	153
Example Code for an Endpoint Java Servlet	155
Invoking Lambda functions	159
Prerequisites	159
Configuring Amazon SNS with Lambda Endpoints with the AWS Management Console	159
Using Amazon SNS Topic Attributes for Message Delivery Status	161
Configuring Message Delivery Status Attributes with the AWS Management Console	162
Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS Endpoints with the AWS SDKs	162
Topic Attributes	162
Java Example to Configure Topic Attributes	163
Message Attributes	164
Message Attribute Items and Validation	164
Data Types	165
Reserved Message Attributes	165
Using Message Attributes with the AWS SDKs	166
Monitoring Amazon SNS with CloudWatch	167
Access CloudWatch Metrics for Amazon SNS	167
Set CloudWatch Alarms for Amazon SNS Metrics	168
Amazon SNS Metrics	169
Dimensions for Amazon Simple Notification Service Metrics	170

Logging Amazon SNS API Calls By Using CloudTrail	171
Amazon SNS Information in CloudTrail	171
Understanding Amazon SNS Log File Entries	172
Appendix: Message and JSON Formats	175
HTTP/HTTPS Headers	175
HTTP/HTTPS Subscription Confirmation JSON Format	176
HTTP/HTTPS Notification JSON Format	178
HTTP/HTTPS Unsubscribe Confirmation JSON Format	179
SetSubscriptionAttributes Delivery Policy JSON Format	181
SetTopicAttributes Delivery Policy JSON Format	181
Appendix: Large Payload and Raw Message Delivery	183
Enabling Raw Message Delivery with the AWS Management Console	183
Document History	185

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 (i.e., web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (i.e., 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.

Topics

- [Are You a First-Time Amazon Simple Notification Service User? \(p. 2\)](#)
- [Beyond the Getting Started Section \(p. 2\)](#)
- [Accessing Amazon SNS \(p. 2\)](#)
- [Common Amazon SNS Scenarios \(p. 3\)](#)

Are You a First-Time Amazon Simple Notification Service User?

If you are a first-time user of Amazon SNS, we recommend that you begin by reading the following sections:

- **What is Amazon SNS** – The rest of this section includes a video that introduces Amazon SNS and walks you through the example presented in [Getting Started with Amazon Simple Notification Service \(p. 5\)](#), and presents common use-case scenarios.
- **Getting Started** – The [Getting Started with Amazon Simple Notification Service \(p. 5\)](#) section walks you through creating a topic, subscribing to it, publishing a message to it, unsubscribing from it, and finally, deleting the topic.

Beyond the Getting Started Section

Beyond the getting started section, you'll probably want to learn more about Amazon SNS operations. The following sections provide detailed information about working with Amazon SNS:

- [Managing Access to Your Amazon SNS Topics \(p. 12\)](#)

You have detailed control over which endpoints a topic allows, who is able to publish to a topic, and under what conditions. This section shows you how to control access through the use of *access control policies*.

- [Monitoring Amazon SNS with CloudWatch \(p. 167\)](#)

Amazon SNS and CloudWatch are integrated so you can collect, view, and analyze metrics for every active Amazon SNS topic.

- [Sending Amazon SNS Messages to Amazon SQS Queues \(p. 106\)](#)

You can use Amazon SNS to send messages to one or more Amazon SQS queues.

- [Sending and Receiving SMS Notifications Using Amazon SNS \(p. 121\)](#)

You can use Amazon Simple Notification Service (Amazon SNS) to send SMS notifications to SMS-enabled mobile phones and smart phones.

- [Sending Amazon SNS Messages to HTTP/HTTPS Endpoints \(p. 127\)](#)

You can use Amazon SNS to send notification messages to one or more HTTP or HTTPS endpoints.

Accessing Amazon SNS

If you have an AWS account, you can access Amazon SNS in any of the following ways.

AWS Management Console

The AWS Management Console provides a web interface where you can manage your compute, storage, and other cloud resources. Within the AWS Management Console, individual services have their own console. To open the Amazon SNS console, log in to <https://console.aws.amazon.com/> and choose **SNS** from the console home page, or use the SNS console direct URL: <https://console.aws.amazon.com/sns/>. For a tutorial that helps you complete common SNS tasks in the console, see [Getting Started with Amazon Simple Notification Service \(p. 5\)](#).

AWS Command Line Interface (CLI)

Provides commands for a broad set of AWS products, and is supported on Windows, Mac, and Linux. To get started, see [AWS Command Line Interface User Guide](#). For more information about the commands for Amazon SNS, see [sns](#) in the *AWS Command Line Interface Reference*.

AWS Tools for Windows PowerShell

Provides commands for a broad set of AWS products for those who script in the PowerShell environment. To get started, see the [AWS Tools for Windows PowerShell User Guide](#). For more information about the cmdlets for Amazon SNS, see [Amazon Simple Notification Service](#) in the *AWS Tools for Windows PowerShell Reference*.

AWS SDKs

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to Amazon SNS and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the [Tools for Amazon Web Services](#) page.

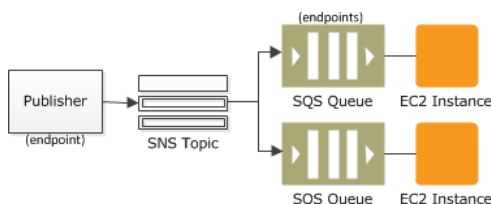
Amazon SNS Query API

You can access Amazon SNS and AWS programmatically by using the Amazon SNS Query API, which lets you issue requests directly to the service. For more information, see the [Amazon Simple Notification Service API Reference](#).

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 [Sending Amazon SNS Messages to Amazon SQS Queues \(p. 106\)](#). For more information about sending Amazon

SNS messages to HTTP/S endpoints, see [Sending Amazon SNS Messages to HTTP/HTTPS Endpoints](#) (p. 127).

Application and System Alerts

Application and system alerts are notifications, triggered by predefined thresholds, 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 AWS 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 [Sending and Receiving SMS Notifications Using Amazon SNS](#) (p. 121).

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 [Amazon SNS Mobile Push Notifications](#) (p. 36)

Getting Started with Amazon Simple Notification Service

This section contains information for you to understand [Amazon SNS](#) concepts and quickly set up and use available tools and interfaces for creating and publishing to topics. To get started with push notification messages, see [Amazon SNS Mobile Push Notifications](#) (p. 36).

Topics

- [Before You Begin](#) (p. 5)
- [Create a Topic](#) (p. 6)
- [Subscribe to a Topic](#) (p. 6)
- [Publish to a Topic](#) (p. 7)
- [Clean Up](#) (p. 9)
- [Using the AWS SDK for Java with Amazon SNS](#) (p. 9)

Before You Begin

To use Amazon SNS, you need an AWS account. If you don't already have one, use the following procedure.

To sign up for AWS account

1. Open <http://aws.amazon.com/>, and then choose **Create an AWS Account**.
2. Follow the online instructions.

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

To get started with Amazon SNS

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Click the **Get Started** button.

You should now be on the SNS Home page.

Create a Topic

Now that you're signed up for Amazon SNS, you're ready to create a topic. A topic is a communication channel to send messages and subscribe to notifications. It provides an access point for publishers and subscribers to communicate with each other. In this section you create a topic named *MyTopic*.

To create a topic

1. In the [Amazon SNS console](#), click **Create topic**.

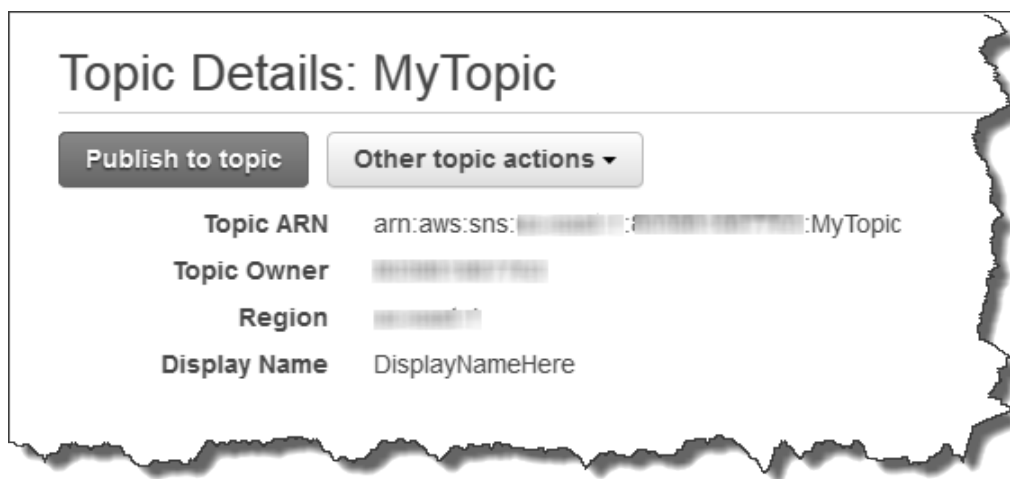
The **Create topic** dialog box appears.

2. In the **Topic name** box, type a topic name.
3. Click **Create topic**.

The new topic appears in the **Topics** page.

4. Select the new topic and then click the topic ARN.

The **Topic Details** page appears.



5. Copy the topic **ARN** for the next task.

```
arn:aws:sns:us-west-2:111122223333:MyTopic
```

Subscribe to a Topic

To receive messages published to a topic, you have to subscribe an endpoint to that topic. An endpoint is a mobile app, web server, email address, or an Amazon SQS queue that can receive notification messages from Amazon SNS. Once you subscribe an endpoint to a topic and the subscription is confirmed, the endpoint will receive all messages published to that topic.

In this section you subscribe an endpoint to the topic you just created in the previous section. You configure the subscription to send the topic messages to your email account.

To subscribe to a topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Click **Create subscription**.

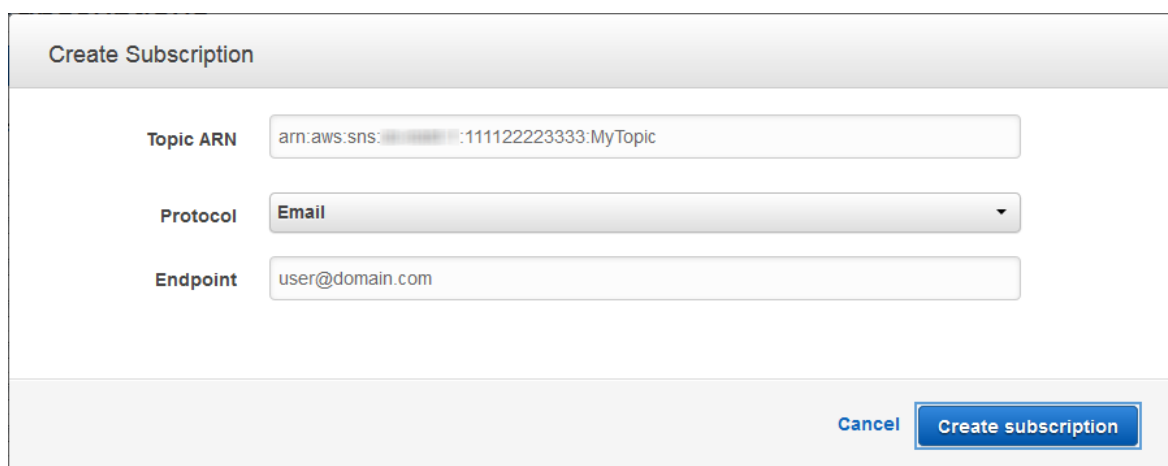
The **Create Subscription** dialog box appears.

3. In the **Topic ARN** field, paste the topic ARN you created in the previous task, for example:
`arn:aws:sns:us-west-2:111122223333:MyTopic`.
4. In the **Protocol** drop-down box, select **Email**.
5. In the **Endpoint** box, type an email address you can use to receive the notification.

Important

Entourage Users: Entourage strips out the confirmation URL. Type an email address you can access in a different email application.

6. Click **Create subscription**.



7. Go to your email application and open the message from AWS Notifications, and then click the link to confirm your subscription.

Your web browser displays a confirmation response from Amazon SNS.

Publish to a Topic

Publishers send messages to topics. Once a new message is published, Amazon SNS attempts to deliver that message to every endpoint that is subscribed to the topic. In this section you publish a message to the email address you defined in the previous task.

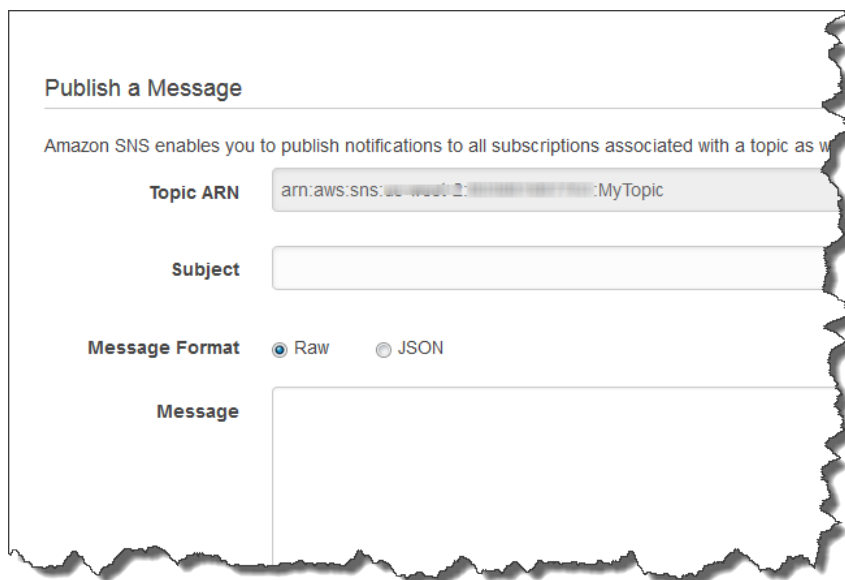
To publish to a topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.

In the left navigation pane, click **Topics** and then select the topic you want to publish to.

2. Click the **Publish to topic** button.

The **Publish a Message** page appears.



The screenshot shows the 'Publish a Message' interface. At the top, it says 'Publish a Message' and 'Amazon SNS enables you to publish notifications to all subscriptions associated with a topic as well as to the topic itself.' Below this are four main sections: 'Topic ARN' with a text input containing 'arn:aws:sns:us-west-2:123456789012:MyTopic'; 'Subject' with an empty text input; 'Message Format' with two radio buttons, 'Raw' (selected) and 'JSON'; and 'Message' with a large empty text area.

3. In the **Subject** box, type a subject line for your message.
4. In the **Message** box, type a brief message.
5. Click **Publish Message**.

A confirmation dialog box appears.

You can now use your email application to open the message from AWS Notifications and read the message.

Create Different Messages for Each Protocol

You can use message formatting support to customize the messages you send for each protocol. For example, a notification that goes to both email and SMS subscribers can be tailored to each type of client. SMS users can receive a version of the message formatted for the available 140 characters supported by the SMS standard, while email users can receive a longer, more detailed version of the same content.

To publish to a topic with message formatting

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic.
3. Click the **Publish to topic** button.

The **Publish a message** page appears.

4. Click the **JSON Message Generator** button.
5. In the **Message** box, type a brief message.
6. In this example for the **Target Platforms**, select **email** and **sms**.
7. Click the **Generate JSON** button.

In the following example, messages are specified for the default, email, and sms protocols. Do not delete any protocols from the list.

You can now modify the message text so that it is tailored to each type of client, such as in this example – up to 140 characters for sms and up to 256Kb for email.

```
{  
  "default": "Message body text here.",  
  "email": "Message body text here.",  
  "sms": "Message body text here."  
}
```

8. Click **Publish message**.

A confirmation dialog box appears.

Clean Up

You have created a topic, subscribed to it, and published a message to the topic. Now you clean up your environment by unsubscribing from the topic and then deleting the topic.

To unsubscribe from a topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.

In the left navigation pane, click **Subscriptions**.

The **Subscriptions** page opens.

2. Select your topic in the subscription list. This will be the only listing on the page, unless you created more than one subscription.
3. Click the **Other actions** drop-down list and then click **Delete subscription(s)**.

The **Delete** confirmation dialog box appears.

4. Click **Delete**.

To delete a topic

Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.

1. In the left navigation pane, click **Topics**, and then select the topic you want to delete.
2. Click the **Actions** drop-down list and select **Delete topics**.

The **Delete** confirmation dialog box appears.

3. Click **Delete**.

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

Using the AWS SDK for Java with Amazon SNS

The SDK for Java provides a class named `AmazonSNSClient` that you can use to interact with Amazon SNS. For information about downloading the AWS SDK for Java, go to [AWS SDK for Java](#).

The `AmazonSNSClient` class defines methods that map to underlying Amazon SNS Query API actions. (These actions are described in the [Amazon SNS API Reference](#)). When you call a method, you must create a corresponding request object and response object. The request object includes information that

you must pass with the actual request. The response object includes information returned from Amazon SNS in response to the request.

For example, the `AmazonSNSClient` class provides the `createTopic` method to create a topic to which notifications can be published. This method maps to the underlying [CreateTopic](#) API action. You create a `CreateTopicRequest` object to pass information with the `createTopic` method.

The following import statements are used with the provided java samples.

```
import com.amazonaws.services.sns.AmazonSNSClient;
import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sns.model.SubscribeRequest;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.PublishResult;
import com.amazonaws.services.sns.model.DeleteTopicRequest;
```

The following example shows how to create a new Amazon SNS client, set the Amazon SNS endpoint to use, and then create a new topic.

Note

In some of the following examples, the `getCachedResponseMetadata` method is used to show how to programmatically retrieve the request ID for a previously executed successful Amazon SNS request. This is typically used for debugging issues and is helpful when requesting assistance from Amazon Web Services.

Create a Topic

```
//create a new SNS client and set endpoint
AmazonSNSClient snsClient = new AmazonSNSClient(new ClasspathPropertiesFileCred
entialsProvider());
snsClient.setRegion(Region.getRegion(Regions.US_EAST_1));

//create a new SNS topic
CreateTopicRequest createTopicRequest = new CreateTopicRequest("MyNewTopic");
CreateTopicResult createTopicResult = snsClient.createTopic(createTopicRequest);
//print TopicArn
System.out.println(createTopicResult);
//get request id for CreateTopicRequest from SNS metadata
System.out.println("CreateTopicRequest - " + snsClient.getCachedResponse
Metadata(createTopicRequest));
```

When you run this example, the following is displayed in the console output window of your IDE, such as Eclipse:

```
{TopicArn: arn:aws:sns:us-east-1:123456789012:MyNewTopic}
CreateTopicRequest - {AWS_REQUEST_ID=93f7fc90-f131-5ca3-ab18-b741fef918b5}
```

The `TopicArn` is assigned to a string variable to use in additional operations.

```
String topicArn = "arn:aws:sns:us-east-1:123456789012:MyNewTopic";
```

The following examples show how to subscribe to, publish to, and delete a topic.

Subscribe to a Topic

```
//subscribe to an SNS topic
SubscribeRequest subRequest = new SubscribeRequest(topicArn, "email",
"example@example.com");
snsClient.subscribe(subRequest);
//get request id for SubscribeRequest from SNS metadata
System.out.println("SubscribeRequest - " + snsClient.getCachedResponse
Metadata(subRequest));
System.out.println("Check your email and confirm subscription.");
```

When you run this example, the following is displayed in the console output window of your IDE:

```
SubscribeRequest - {AWS_REQUEST_ID=9b7ff59a-f917-533a-a6bd-be4bf6df0acf}
Check your email and confirm subscription.
```

Publish to a Topic

```
//publish to an SNS topic
String msg = "My text published to SNS topic with email endpoint";
PublishRequest publishRequest = new PublishRequest(topicArn, msg);
PublishResult publishResult = snsClient.publish(publishRequest);
//print messageId of message published to SNS topic
System.out.println("MessageId - " + publishResult.getMessageId());
```

When you run this example, the following is displayed in the console output window of your IDE:

```
MessageId - 9b888f80-15f7-5c30-81a2-c4511a3f5229
```

Delete a Topic

```
//delete an SNS topic
DeleteTopicRequest deleteTopicRequest = new DeleteTopicRequest(topicArn);
snsClient.deleteTopic(deleteTopicRequest);
//get request id for DeleteTopicRequest from SNS metadata
System.out.println("DeleteTopicRequest - " + snsClient.getCachedResponse
Metadata(deleteTopicRequest));
```

When you run this example, the following is displayed in the console output window of your IDE:

```
DeleteTopicRequest - {AWS_REQUEST_ID=067a4980-4e93-5bfc-b88c-0251415bc852}
```

Managing Access to Your Amazon SNS Topics

Topics

- [Overview](#) (p. 12)
- [Special Information for Amazon SNS Policies](#) (p. 26)
- [Controlling User Access to Your AWS Account](#) (p. 27)

Amazon SNS supports other protocols beside email. You can use HTTP, HTTPS, and Amazon SQS queues. You have detailed control over which endpoints a topic allows, who is able to publish to a topic, and under what conditions. This appendix shows you how to control through the use of *access control policies*.

The main portion of this section includes basic concepts you need to understand, how to write a policy, and the logic Amazon Web Services (AWS) uses to evaluate policies and decide whether to give the requester access to the resource. Although most of the information in this section is service-agnostic, there are some Amazon SNS-specific details you need to know. For more information, see [Special Information for Amazon SNS Policies](#) (p. 26).

Overview

Topics

- [When to Use Access Control](#) (p. 13)
- [Key Concepts](#) (p. 13)
- [Architectural Overview](#) (p. 15)
- [Using the Access Policy Language](#) (p. 17)
- [Evaluation Logic](#) (p. 18)
- [Example Cases for Amazon SNS Access Control](#) (p. 21)

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.

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 (e.g., Publish). For more information, see [Allowing AWS account Access to a Topic \(p. 22\)](#).
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see [Limiting Subscriptions to HTTPS \(p. 22\)](#).
- You want to allow Amazon SNS to publish messages to your Amazon SQS queue. For more information, see [Publishing to an Amazon SQS Queue \(p. 23\)](#).

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.

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" (i.e., 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 (e.g., the request must arrive before a specific day)
- IP address (e.g., 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 (e.g., Amazon SQS or Amazon SNS) might also define service-specific keys. For more information, see [Special Information for Amazon SNS Policies \(p. 26\)](#).

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 \(p. 18\)](#).

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 given 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, given 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 \(p. 18\)](#).

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 \(p. 18\)](#).

Default Deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

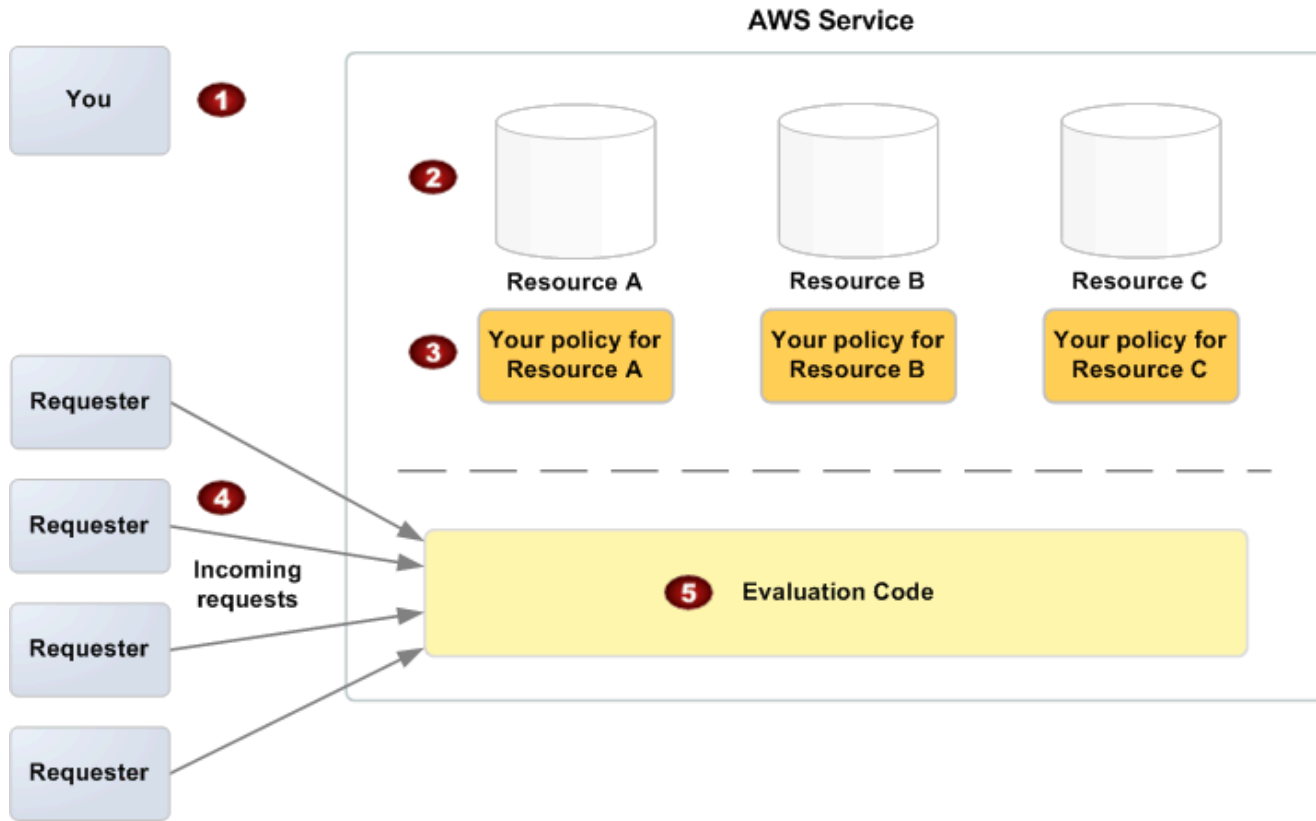
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

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

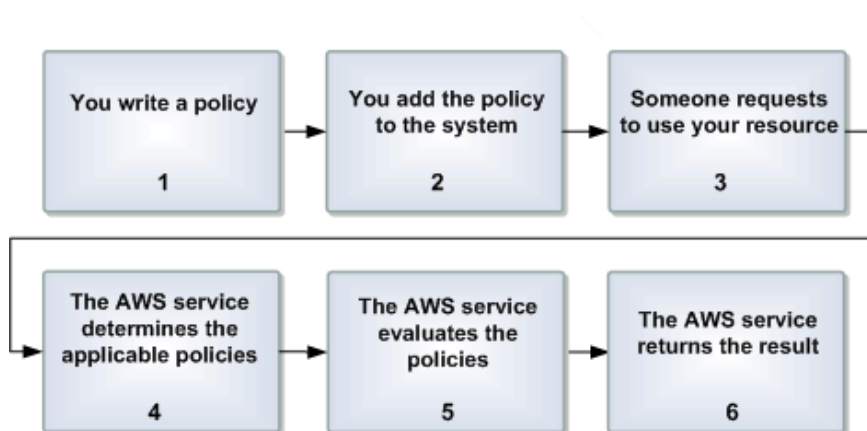
The following figure and table describe the main components that interact to provide access control for your resources.



1	You, the resource owner.
2	Your resources (contained within the AWS service; e.g., Amazon SQS queues).
3	Your policies. Typically you have one policy per resource, although you could have multiple. The AWS service itself provides an API you use to upload and manage your policies.
4	Requesters and their incoming requests to the AWS service.
5	The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see Evaluation Logic (p. 18) .

Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.



Process for Using Access Control with the Access Policy Language

1	You write a policy for your resource. For example, you write a policy to specify permissions for your Amazon SNS topics.
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SNS <code>SetTopicAttributes</code> action to upload a policy for a particular Amazon SNS topic.
3	Someone sends a request to use your resource. For example, a user sends a request to Amazon SNS to use one of your topics.
4	The AWS service determines which policies are applicable to the request. For example, Amazon SNS looks at all the available Amazon SNS policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies. For example, Amazon SNS evaluates the policies and determines if the requester is allowed to use your topic or not. For information about the decision logic, see Evaluation Logic (p. 18) .
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

Related Topics

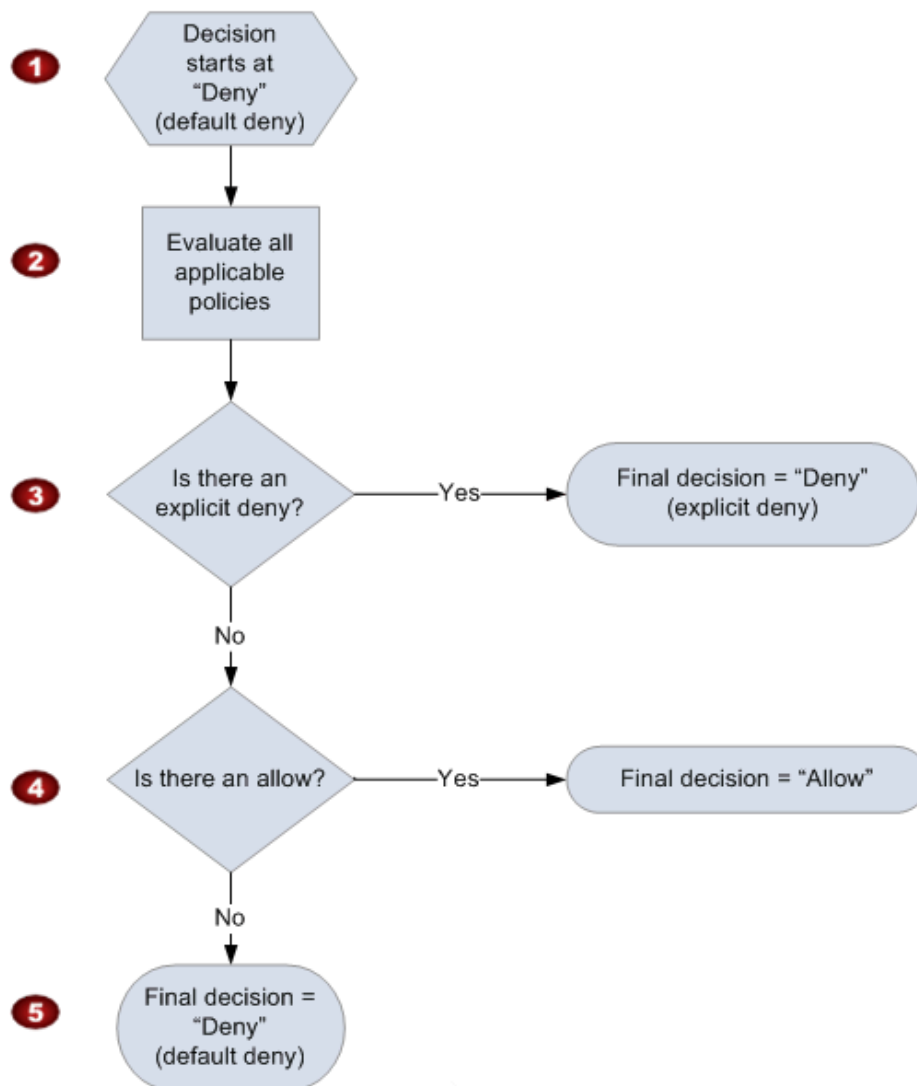
- [Architectural Overview \(p. 15\)](#)

Evaluation Logic

The goal at evaluation time is to decide whether a given request should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.



1	The decision starts with a default deny.
---	--

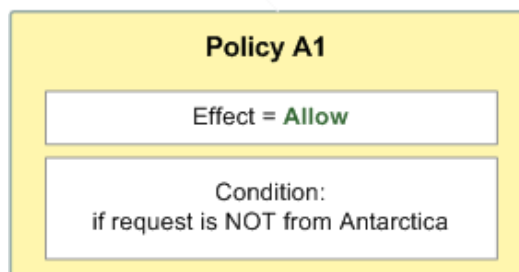
2	The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.
3	In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request. If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit Deny (p. 15)).
4	If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a <i>default deny</i> (for more information, see Default Deny (p. 15)).

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.

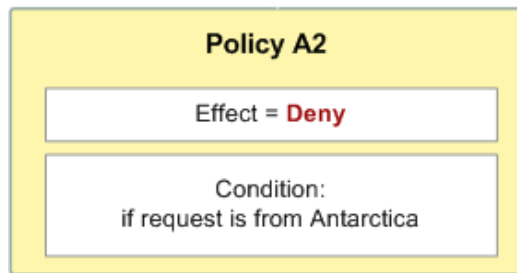
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met, 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.



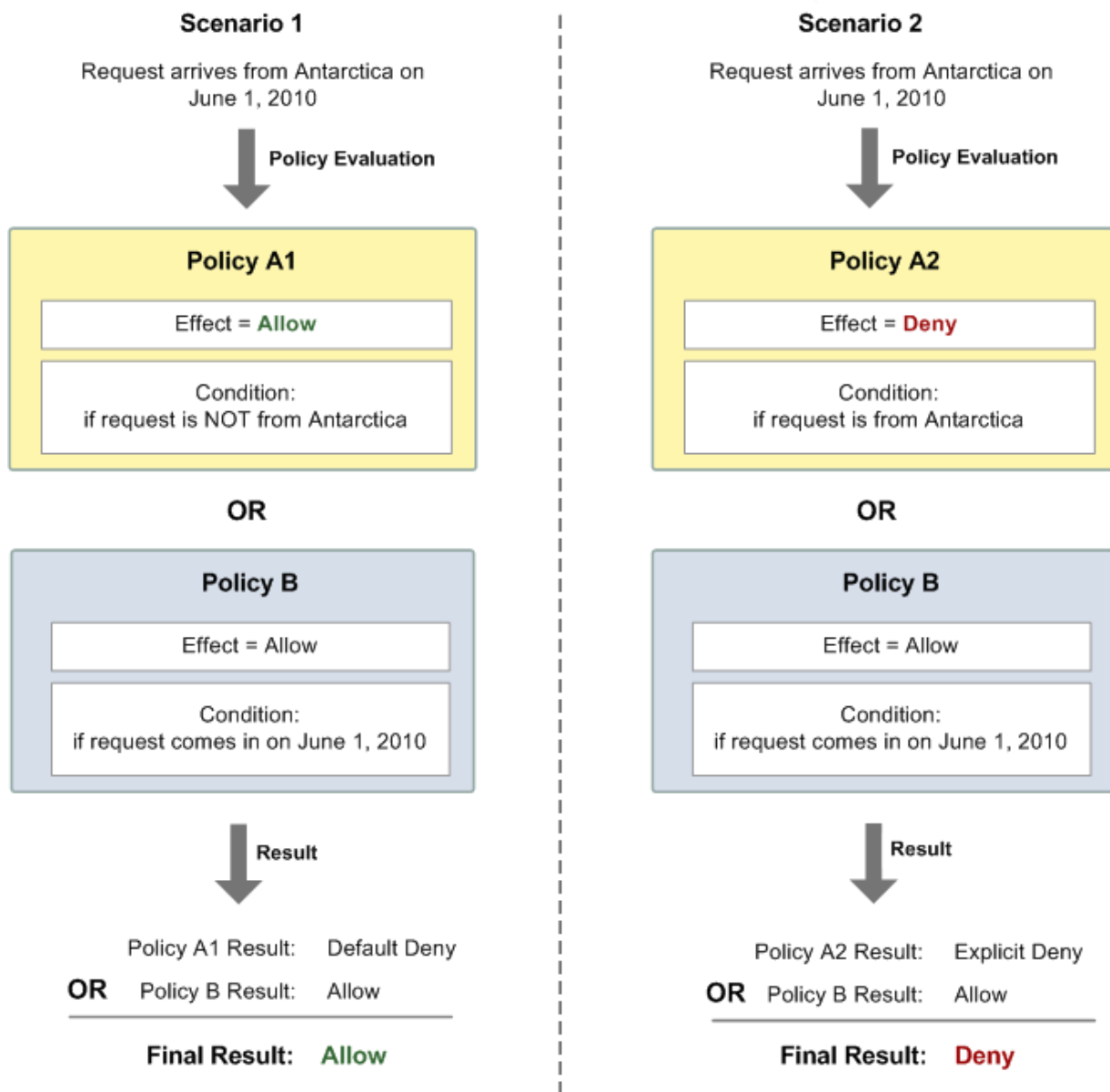
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

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 B2 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

- [Allowing AWS account Access to a Topic \(p. 22\)](#)
- [Limiting Subscriptions to HTTPS \(p. 22\)](#)

- [Publishing to an Amazon SQS Queue \(p. 23\)](#)
- [Allowing Any AWS Resource to Publish to a Topic \(p. 24\)](#)
- [Allowing an Amazon S3 Bucket to Publish to a Topic \(p. 24\)](#)

This section gives a few examples of typical use cases for access control.

Allowing 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 (e.g., Publish).

You can do this by 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-1:444455556666:MyTopic`, with AWS account ID `1111-2222-3333`, the `Publish` action, and the label `give-1234-publish`, Amazon SNS would generate and insert the following access control policy statement:

```
{
  "Version": "2012-10-17",
  "Id": "AWSAccountTopicAccess",
  "Statement": [
    {
      "Sid": "give-1234-publish",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": ["sns:Publish"],
      "Resource": "arn:aws:sns:us-east-1:444455556666:MyTopic"
    }
  ]
}
```

Once this statement is added, the user with AWS account `1111-2222-3333` can publish messages to the topic.

Limiting Subscriptions to HTTPS

In this use case, you want to allow subscription requests to your topic *only by HTTPS*, for security.

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 gives the AWS account ID `1111-2222-3333` the ability to subscribe to notifications from a topic.

Note

`Subscribe` and `Receive` are separate actions in the policy. You can apply different conditions to the subscriber and the message recipient.

```
{
  "Version": "2012-10-17",
  "Id": "SomePolicyId",
  "Statement" : [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal" : {
        "AWS": "111122223333"
      },
      "Action": [ "sns:Subscribe" ],
      "Resource": "arn:aws:sns:us-east-1:444455556666:MyTopic",
      "Condition" : {
        "StringEquals" : {
          "sns:Protocol": "https"
        }
      }
    }
  ]
}
```

Publishing 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 that 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.

```
{
  "Version": "2012-10-17",
  "Id": "MyQueuePolicy",
  "Statement" : [
    {
      "Sid": "Allow-SNS-SendMessage",
      "Effect": "Allow",
      "Principal" : "*",
      "Action": [ "sqs:SendMessage" ],
      "Resource": "arn:aws:sqs:us-east-1:444455556666:MyQueue",
      "Condition" : {
        "ArnEquals" : {
          "aws:SourceArn": "arn:aws:sns:us-east-1:444455556666: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-1:444455556666:MyTopic`.

The preceding policy is an example of the Amazon SQS policy you could write and add to a specific queue. It would grant Amazon SNS and other AWS products access. Amazon SNS gives a default policy to all newly created topics. The default policy gives all other AWS products access to your topic. This default policy uses an `aws:SourceArn` condition to ensure that AWS products access your topic only on behalf of AWS resources you own.

Allowing 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 (e.g., 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 gives the AWS account ID `4444-5555-6666` the ability to publish to `My-Topic` from any AWS resource owned by the account.

```
{
  "Version": "2012-10-17",
  "Id": "MyAWSPolicy",
  "Statement" : [
    {
      "Sid": "My-statement-id",
      "Effect": "Allow",
      "Principal" : "*",
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-1:111122223333:My-Topic",
      "Condition": {
        "StringEquals": {
          "AWS:SourceOwner": "444455556666"
        }
      }
    }
  ]
}
```

Allowing 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`.

```
{
  "Version": "2012-10-17",
  "Id": "MyAWSPolicy",
  "Statement" : [
    {
      "Sid": "My-statement-id",
      "Effect": "Allow",
      "Principal" : "*",
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-1:111122223333:My-Topic",
      "Condition": {
        "StringEquals": { "AWS:SourceOwner": "444455556666" } ,
        "ArnLike": { "AWS:SourceArn": "arn:aws:s3:*:*:*" }
      }
    }
  ]
}
```

Special Information for Amazon SNS Policies

The following list gives 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`

Amazon SNS Policy Limits

The following table lists the maximum limits for policy information.

Name	Maximum Limit
Bytes	30 kb
Statements	100
Principals	1 to 200 (0 is invalid.)
Resource	1 (0 is invalid. The value must match the ARN of the policy's topic.)

Valid Amazon SNS Policy Actions

Amazon SNS supports the actions shown in the following table.

Action	Description
sns:AddPermission	Grants permission to add permissions to the topic policy.
sns>DeleteTopic	Grants permission to delete a topic.
sns:GetTopicAttributes	Grants permission to receive all of the topic attributes.
sns:ListSubscriptionsByTopic	Grants permission to retrieve all the subscriptions to a specific topic.
sns:Publish	Grants permission to publish to a topic or endpoint. For more information, see Publish in the Amazon Simple Notification Service API Reference
sns:Receive	Grants permission to receive notifications from a topic.
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.

Amazon SNS Keys

Amazon SNS uses the following service-specific keys. You can use these in policies that restrict access to `Subscribe` requests and `Receive` 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 \(p. 32\)](#)) to restrict access to specific endpoints (e.g., `*@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 \(p. 32\)](#)) to restrict publication to specific delivery protocols (e.g., `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.

Controlling User Access to Your AWS Account

Topics

- [IAM and Amazon SNS Policies Together \(p. 27\)](#)
- [Amazon SNS ARNs \(p. 30\)](#)
- [Amazon SNS Actions \(p. 31\)](#)
- [Amazon SNS Keys \(p. 31\)](#)
- [Example Policies for Amazon SNS \(p. 32\)](#)
- [Using Temporary Security Credentials \(p. 34\)](#)

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 gives 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.

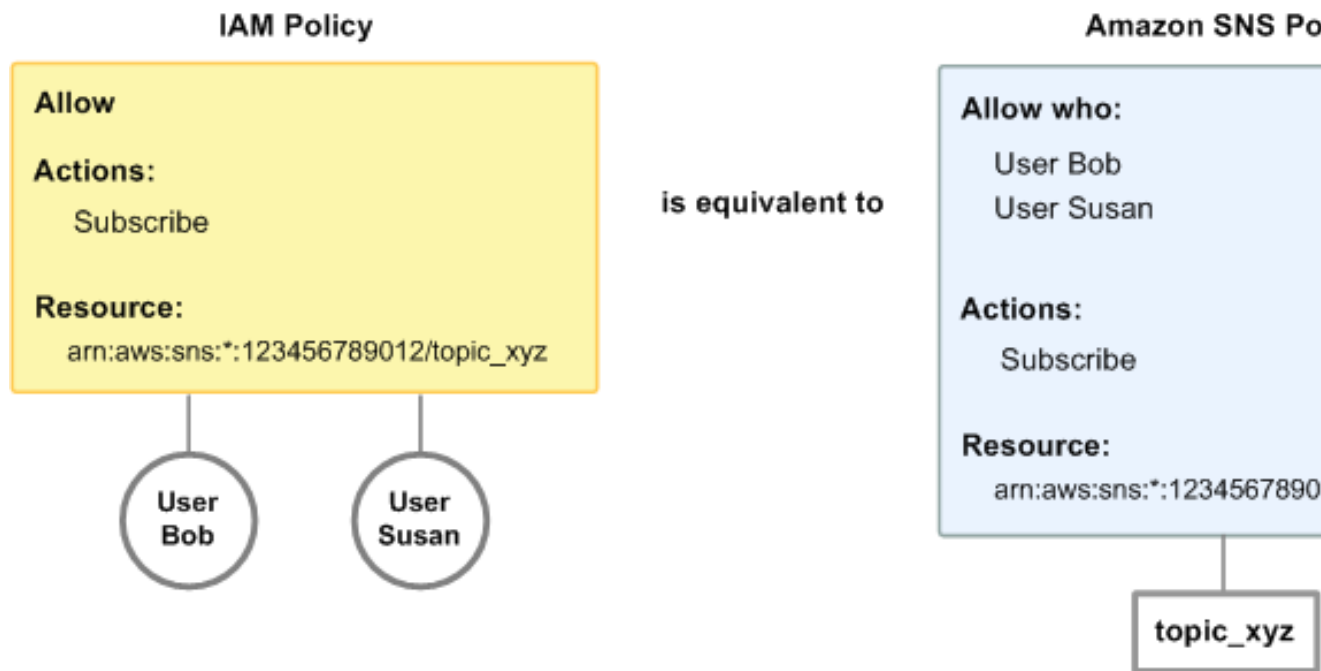
For examples of policies that cover Amazon SNS actions and resources, see [Example Policies for Amazon SNS \(p. 32\)](#).

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 (e.g., who can publish messages to it, who can subscribe to it, etc.). Amazon SNS policies can give access to other AWS accounts, or to users within your own AWS account.

To give 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 gives 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

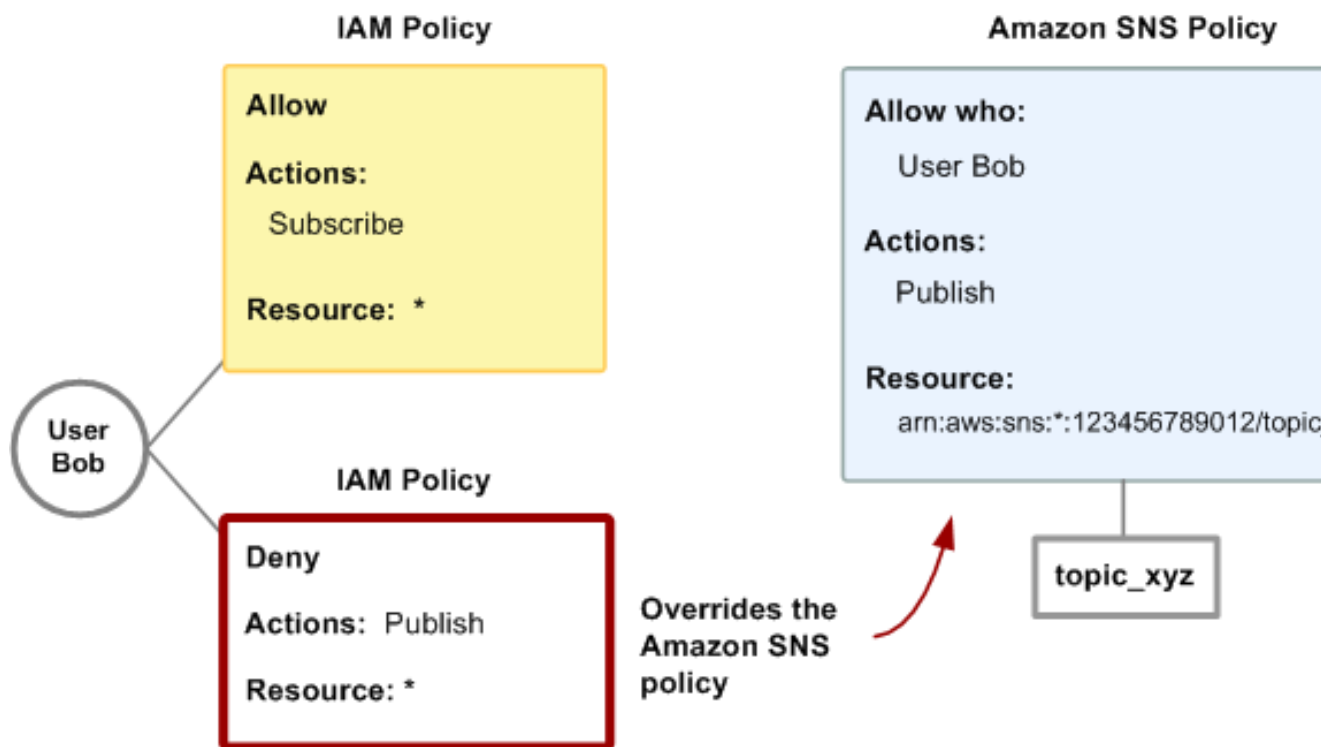
In this example, both an IAM policy and an Amazon SNS policy apply to Bob. The IAM policy gives him permission for `Subscribe` on any of the AWS account's topics, whereas the Amazon SNS policy gives him permission to use `Publish` on a specific topic (`topic_xyz`). The following diagram illustrates the 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 \(p. 18\)](#)). The following diagram illustrates the concept.



For examples of policies that cover Amazon SNS actions and resources, see [Example Policies for Amazon SNS \(p. 32\)](#). For more information about writing Amazon SNS policies, go to the [technical documentation for Amazon SNS](#).

Amazon SNS ARNs

For Amazon SNS, topics are the only resource type you can specify in a policy. 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

Following is an ARN for a topic named `my_topic` in the `us-east-1` region, belonging to AWS account `123456789012`.

```
arn:aws:sns:us-east-1:123456789012: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 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 Keys

Amazon SNS implements the following AWS-wide policy keys, plus some service-specific keys.

AWS-Wide Policy Keys

- `aws:CurrentTime`—To check for date/time conditions.
- `aws:EpochTime`—To check for date/time conditions using a date in epoch or UNIX time.
- `aws:MultiFactorAuthAge`—To check how long ago (in seconds) the MFA-validated security credentials making the request were issued using Multi-Factor Authentication (MFA). Unlike other keys, if MFA is not used, this key is not present.
- `aws:principaltype`—To check the type of principal (user, account, federated user, etc.) for the current request.
- `aws:SecureTransport`—To check whether the request was sent using SSL. For services that use only SSL, such as Amazon RDS and Amazon Route 53, the `aws:SecureTransport` key has no meaning.
- `aws:SourceArn`—To check the source of the request, using the Amazon Resource Name (ARN) of the source. (This value is available for only some services. For more information, see [Amazon Resource Name \(ARN\)](#) under "Element Descriptions" in the *Amazon Simple Queue Service Developer Guide*.)
- `aws:SourceIp`—To check the IP address of the requester. Note that if you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, the public IP address of the instance is evaluated.

- `aws:UserAgent`—To check the client application that made the request.
- `aws:userid`—To check the user ID of the requester.
- `aws:username`—To check the user name of the requester, if available.

Note

Key names are case sensitive.

Amazon SNS Keys

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 \(p. 32\)](#)) to restrict access to specific endpoints (e.g., `*@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 \(p. 32\)](#)) to restrict publication to specific delivery protocols (e.g., `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 gives access to `CreateTopic`, `ListTopics`, `SetTopicAttributes`, and `DeleteTopic`.

```
{
  "Version": "2012-10-17",
  "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 gives access to `Publish` on the specific topic of interest.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:*:123456789012:topic_xyz"
  }]
}
```

Example 3: Give users in the AWS account ability to subscribe to topics

In this example, we create a policy that gives access to the `Subscribe` action, with string matching conditions for the `sns:Protocol` and `sns:Endpoint` policy keys.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:Subscribe"],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "SNS:Endpoint": "*@yourcompany.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 gives 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 gives users wide access to Amazon SNS.

```
{
  "Version": "2012-10-17",
  "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

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.

Amazon Simple Notification Service Developer Guide Using Temporary Security Credentials

```
http://sns.us-east-1.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 Mobile Push Notifications

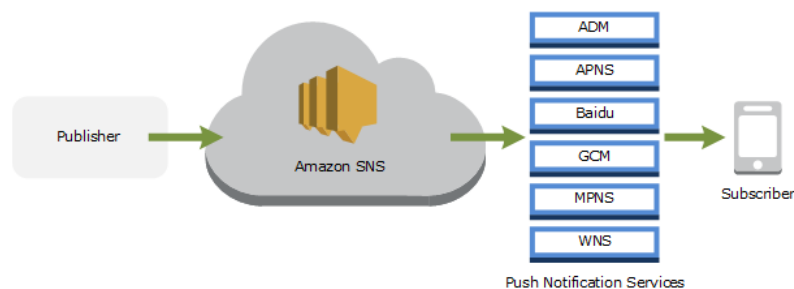
With [Amazon SNS](#), you have the ability to send push notification messages directly to apps on mobile devices. Push notification messages sent to a mobile endpoint can appear in the mobile app as message alerts, badge updates, or even sound alerts.

Overview

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)
- Google Cloud Messaging for Android (GCM)
- Microsoft Push Notification Service for Windows Phone (MPNS)
- Windows Push Notification Services (WNS)

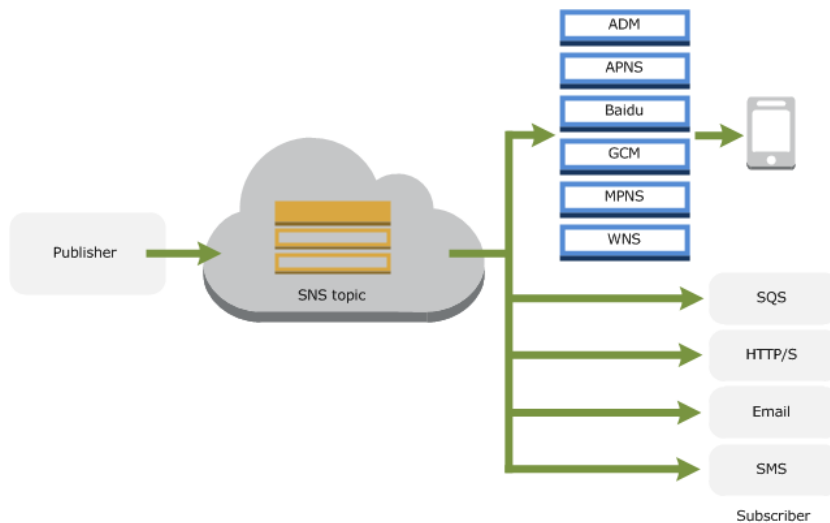
The following figure shows an overview of how Amazon SNS is used to send a direct push notification message to a mobile endpoint.



Push notification services, such as APNS and GCM, 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 [Amazon SNS Mobile Push High Level Steps \(p. 38\)](#)

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. The following figure shows a mobile endpoint as a subscriber to an Amazon SNS topic. The mobile endpoint communicates using push notification services where the other endpoints do not.



Prerequisites

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, GCM, 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

GCM, 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 by using the credentials that are stored in its corresponding platform application.

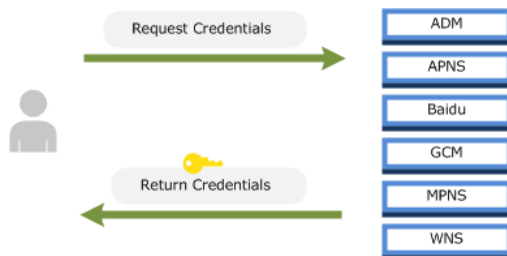
The following sections include the prerequisites for each of the supported push notification services. Once you've obtained the prerequisite information, you can send a push notification message using the AWS Management Console or the Amazon SNS mobile push APIs. For more information, see [Amazon SNS Mobile Push High Level Steps](#) (p. 38).

Amazon SNS Mobile Push High Level Steps

This section provides the high level steps you must perform to use Amazon SNS mobile push. First, for the mobile platforms you want to support you must complete the prerequisites, such as obtaining the required credentials and device token. For more information, see [Prerequisites](#) (p. 37) Then, you use the information you obtained from the mobile platforms with Amazon SNS to send a message to a mobile device. This information should help you gain a better understanding of the steps involved when using the Amazon SNS mobile push, as described in [Using Amazon SNS Mobile Push](#) (p. 78).

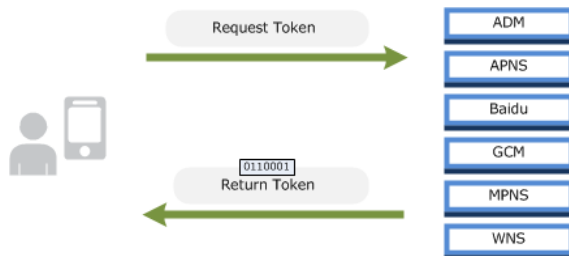
Step 1: Request Credentials from Mobile Platforms

To use Amazon SNS mobile push, you must first request the necessary credentials from the mobile platforms. For more information, see the getting started section for your platform later in this guide.



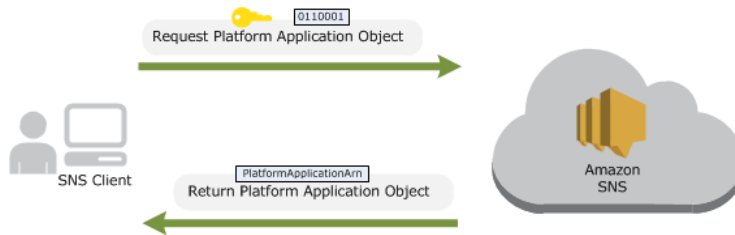
Step 2: Request Token from Mobile Platforms

You then use the returned credentials to request a token for your mobile app and device from the mobile platforms. The token you receive represents your mobile app and device. For more information, see the getting started section for you platform later in this guide.



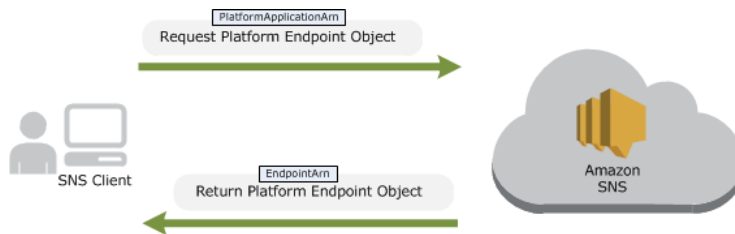
Step 3: Create Platform Application Object

The credentials and token are then used to create a platform application object (PlatformApplicationArn) from Amazon SNS. For more information, see [Create a Platform Endpoint and Manage Device Tokens](#) (p. 83).



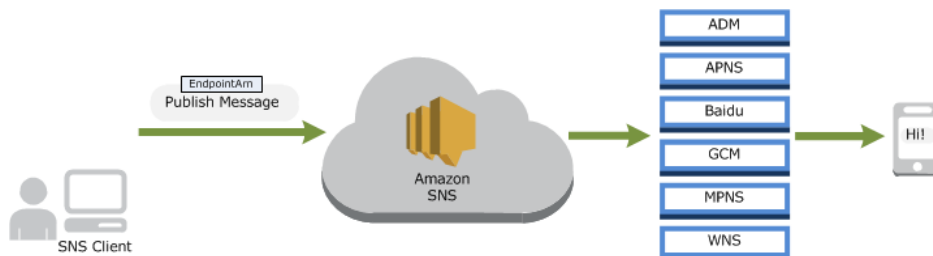
Step 4: Create Platform Endpoint Object

The PlatformApplicationArn is then used to create a platform endpoint object (EndpointArn) from Amazon SNS. For more information, see [Create a Platform Endpoint and Manage Device Tokens](#) (p. 83).



Step 5: Publish Message to Mobile Endpoint

The EndpointArn is then used to publish a message to an app on a mobile device. For more information, see [Send a Direct Message to a Mobile Device](#) (p. 88) and the [Publish API](#) in the Amazon Simple Notification Service API Reference.



Getting Started with Amazon Device Messaging

Amazon Device Messaging (ADM) is a service that enables you to send push notification messages to Kindle Fire apps. This section describes how to obtain the ADM prerequisites and send a push notification message using Amazon SNS and ADM.

Topics

- [ADM Prerequisites](#) (p. 40)

- [Step 1: Create a Kindle Fire App with the ADM Service Enabled](#) (p. 40)
- [Step 2: Obtain a Client ID and Client Secret](#) (p. 40)
- [Step 3: Obtain an API Key](#) (p. 41)
- [Step 4: Obtain a Registration ID](#) (p. 41)
- [Step 5: Sending a Push Notification Message to a Kindle Fire app using Amazon SNS and ADM](#) (p. 42)

ADM Prerequisites

To send push notifications to a Kindle Fire app using Amazon SNS and ADM, you need the following:

- Kindle Fire app with the ADM service enabled
- Client ID and client secret
- API key
- Registration ID

If you already have these prerequisites, then you can send a push notification message to a Kindle Fire app using either the Amazon SNS console or the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push](#) (p. 78). For more information about using the Amazon SNS API, see [Step 5: Sending a Push Notification Message to a Kindle Fire app using Amazon SNS and ADM](#) (p. 42)

Step 1: Create a Kindle Fire App with the ADM Service Enabled

To send a push notification message to a Kindle Fire app, you must have an Amazon developer account, set up your development environment, created a Kindle Fire app with ADM enabled, and registered the app with ADM. For more information, see [Integrating Your App with ADM](#).

To create a Kindle Fire app

1. Create an Amazon developer account by following the instructions at [Create an Account](#).
2. Set up your development environment for developing mobile apps for Kindle Fire tablets. For more information, see [Setting Up Your Development Environment](#).
3. Create a Kindle Fire app. For more information, see [Creating Your First Kindle Fire App](#).

Note

If you do not already have a Kindle Fire app registered with ADM, then you can use the sample Kindle Fire app provided by AWS as a template to get started. For more information, see [Step 4: Obtain a Registration ID](#) (p. 41).

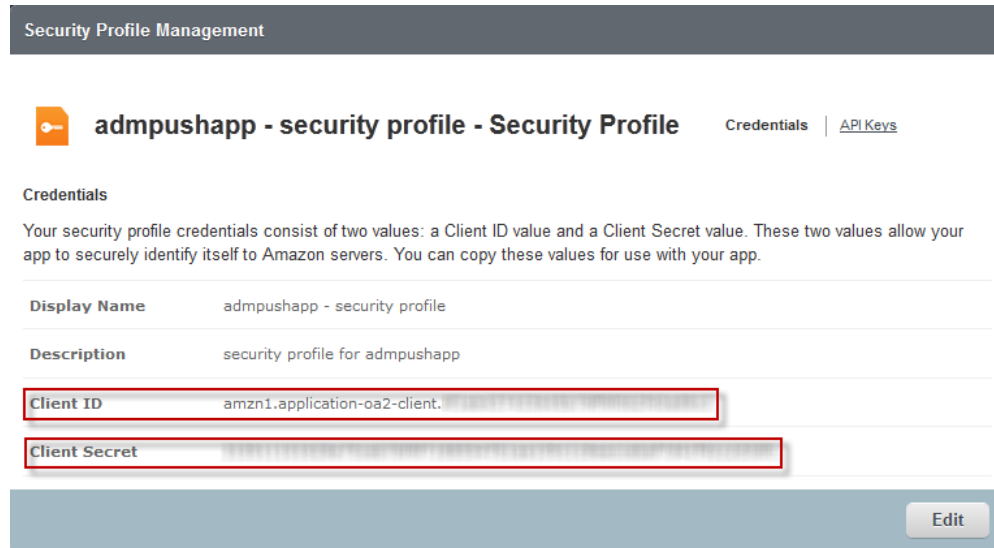
4. On the [Amazon App Distribution Portal](#), click **Apps and Services**, click the name of your Kindle Fire app, and then click **Device Messaging**.
5. Verify that ADM is enabled for the app. If your app is not listed on the Amazon App Distribution Portal, then add it and enable ADM.

Step 2: Obtain a Client ID and Client Secret

ADM uses a client ID and client secret to verify your server's identity. For more information, see [Obtaining ADM Credentials](#).

To obtain a client ID and client secret

1. On the [Amazon App Distribution Portal](#), click **Apps and Services**, click the name of your Kindle Fire app, and then click **Security Profile**. You should see a security profile associated with your app. If not, click **Security Profiles** to create a new security profile.
2. Click **View Security Profile**. Make note of the client ID and client secret.



Step 3: Obtain an API Key

ADM uses an API key to verify your app's identity.

Note

An **API key** is required to use ADM with pre-release or test apps. However, it is not required with a release or production version of your app when you allow Amazon to sign your app on your behalf.

To obtain an API key

- Obtain an API key by following instructions at [Getting Your OAuth Credentials and API Key](#).

Step 4: Obtain a Registration ID

The following steps show how to use the sample Kindle Fire app provided by AWS to obtain a registration ID from ADM. You can use this sample Kindle Fire app as an example to help you get started with Amazon SNS push notifications. The sample app requires that you have included the ADM JAR file, `amazon-device-messaging-1.0.1.jar` in your development environment. For more information, see [Setting Up ADM](#).

To obtain a registration ID from ADM for your app

1. Download and unzip the [snsmobilepush.zip](#) file.
2. Import the `KindleMobilePushApp` folder into your IDE. In Eclipse, click **File, Import**, expand the **Android** folder, click **Existing Android Code Into Workspace**, click **Next**, browse to the folder `KindleMobilePushApp`, click **OK**, and then click **Finish**.

Amazon Simple Notification Service Developer Guide

Step 5: Sending a Message to a Kindle Fire app using Amazon SNS and ADM

After the sample Kindle Fire app has been imported into your IDE, you need to add the API key for your Kindle Fire app to the `strings.xml` file, which is included in the sample Kindle Fire app.

3. Add the API key to the `strings.xml` file. In your IDE you will find the file included in the **values** folder, which is a subfolder of **res**. You add the string to the following:

```
<string name="api_key"></string>
```

4. Run the app to see the registration ID as output to the Android logging system. If you are using Eclipse with the Android ADT plug-in, you can see the registration ID in the **LogCat** display window. For example, the output containing the registration ID will look similar to the following:

```
amzn1.adm-registration.v2.Example...1cwWJUvgkcPPYcaXCpPwMg3Bqn-  
wiqIEzp5zz7y_jsM0PKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw
```

You should now have the necessary information from ADM (client ID, client secret, API key, and registration ID) to send push notification messages to your mobile endpoint. You can now send a push notification message to the Kindle Fire app on your device by either using the Amazon SNS console or the Amazon SNS API. To use the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#). To use the Amazon SNS API, see [Step 5: Sending a Push Notification Message to a Kindle Fire app using Amazon SNS and ADM \(p. 42\)](#).

Step 5: Sending a Push Notification Message to a Kindle Fire app using Amazon SNS and ADM

This section describes how to use the prerequisite information to send a push notification message to your Kindle Fire app using Amazon SNS and ADM. You add the gathered prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file.

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*.

To add the sample to Eclipse

1. Create a new Java project in Eclipse (**File | New | Java Project**).
2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library...**

Amazon Simple Notification Service Developer Guide
Step 5: Sending a Message to a Kindle Fire app using
Amazon SNS and ADM

3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open `SNSMobilePush.java` in Eclipse.
2. Uncomment `sample.demoKindleAppNotification()`. It should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);
// TODO: Uncomment the services you wish to use.
// sample.demoAndroidAppNotification();
sample.demoKindleAppNotification();
// sample.demoAppleAppNotification();
// sample.demoAppleSandboxAppNotification();
// sample.demoBaiduAppNotification();
// sample.demoWNSAppNotification();
// sample.demoMPNSAppNotification();
```

3. Locate the `demoKindleAppNotification` method and enter the registration ID you received from ADM for the value of the registration ID string. For example, it should look similar to the following:

```
String registrationId = "amzn1.adm-registration.v2.Example...1cwWJUvgk
cPPYcaXCpPWmG3Bqn-wiqIEzp5zZ7y_jsM0PKPxKhd
dCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw";
```

4. Enter the client ID for your app. For example, it should look similar to the following:

```
String clientId = "amzn1.application-oa2-client.EX
AMPLE7423654b79fc9f062fEXAMPLE";
```

5. Enter the client secret for your app. For example, it should look similar to the following:

```
String clientSecret = "EXAMPLE01658e75ceb7bf9f71939647b1aa105c1c8eac
cabaf7d41f68EXAMPLE";
```

6. Enter a name for 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. For example, it should look similar to the following:

```
String applicationName = "admpushapp";
```

7. Run the Java application. You should see output similar to the following in the output window of your IDE:

```
=====
Getting Started with Amazon SNS
=====

{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/ADM/mypush
```

```
appname}
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/ADM/mypushapp
name/97e9ced9-f136-3893-9d60-775467eafebb}
{"ADM": "{ \"data\": { \"message\": \"ENTER YOUR MESSAGE\" } }"}
Published. MessageId=b35fb4bz-b503-4e37-83d4-feu4218d6da6
```

On your Kindle Fire device, you should see a push notification message appear within the Kindle Fire app.

Getting Started with Apple Push Notification Service

Apple Push Notification Service (APNS) is a service that enables you to send push notification messages to iOS and OS X apps. This section describes how to obtain the APNS prerequisites and send a push notification message using Amazon SNS and APNS.

Topics

- [APNS Prerequisites \(p. 44\)](#)
- [Step 1: Create an iOS App \(p. 44\)](#)
- [Step 2: Obtain an APNS SSL Certificate \(p. 45\)](#)
- [Step 3: Obtain the App Private Key \(p. 45\)](#)
- [Step 4: Verify the Certificate and App Private Key \(p. 46\)](#)
- [Step 5: Obtain a Device Token \(p. 46\)](#)
- [Next Steps \(p. 47\)](#)
- [Send a push notification message to an iOS app using Amazon SNS and APNS \(p. 47\)](#)
- [Send a push notification message to a VoIP iOS app using Amazon SNS and APNS \(p. 49\)](#)
- [Send a push notification message to a Mac OS X app using Amazon SNS and APNS \(p. 49\)](#)

APNS Prerequisites

To send push notifications to mobile devices using Amazon SNS and APNS, you need to obtain the following:

- iOS app registered with APNS
- APNS SSL certificate
- App private key
- Device token

If you already have these prerequisites, you can send a push notification message to an iOS app using either the Amazon SNS console or you can use the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#). For more information about using the Amazon SNS API, see [Send a push notification message to an iOS app using Amazon SNS and APNS \(p. 47\)](#).

Step 1: Create an iOS App

To get started with sending a push notification message to an iOS app, you must have an Apple developer account, created an App ID (application identifier), registered your iOS device, and created an iOS

Provisioning Profile. For more information, see the [Local and Remote Notification Programming Guide](#) in the iOS Developer Library.

Note

If you do not already have an iOS app registered with APNS, then you can use the sample iOS app provided by AWS as a template to get started. For more information, see [Step 5: Obtain a Device Token](#) (p. 46).

Step 2: Obtain an APNS SSL Certificate

Amazon SNS requires the APNS SSL certificate of the app in the .pem format when using the Amazon SNS API. When using the Amazon SNS console you can upload the certificate in .p12 format and Amazon SNS will convert it to .pem and display it in the console. You use the Keychain Access application on your Mac computer to export the APNS SSL certificate. For more information about the SSL certificate, see [Provisioning and Development](#) in the Apple Local and Push Notification Programming Guide.

To download an APNS SSL certificate

1. On the [Apple Developer web site](#), click **Member Center**, click **Certificates, Identifiers and Profiles**, and then click **Certificates**.
2. Select the certificate you created for iOS APNS development, click **Download**, and then save the file, which will have the .cer extension type.

To convert the APNS SSL certificate from .cer format to .pem format

The following steps use the openssl utility.

- At a command prompt, type the following command. Replace `myapnsappcert.cer` with the name of the certificate you downloaded from the Apple Developer web site.

```
openssl x509 -in myapnsappcert.cer -inform DER -out myapnsappcert.pem
```

The newly created .pem file will be used to configure Amazon SNS for sending mobile push notification messages.

Step 3: Obtain the App Private Key

Amazon SNS requires an app private key in the .pem format. You use the **Keychain Access** application on your Mac computer to export the app private key.

To obtain the app private key

The private key associated with the SSL certificate can be exported from the Keychain Access application on your Mac computer. This is based on the assumption that you have already imported the .cer file you downloaded from the Apple Developer web site into Keychain Access. You can do this either by copying the .cer file into Keychain Access or double-clicking the .cer file.

1. Open Keychain Access, select **Keys**, and then highlight your app private key.
2. Click **File**, click **Export Items...**, and then enter a name in the **Save As:** field.
3. Accept the default .p12 file format and then click **Save**.

The .p12 file will then be converted to .pem file format.

To convert the app private key from .p12 format to .pem format

- At a command prompt, type the following command. Replace `myapnsappprivatekey.p12` with the name of the private key you exported from Keychain Access.

```
openssl pkcs12 -in myapnsappprivatekey.p12 -out myapnsappprivatekey.pem
-nodes -clcerts
```

The newly created .pem file will be used to configure Amazon SNS for sending mobile push notification messages.

Step 4: Verify the Certificate and App Private Key

You can verify the .pem certificate and private key files by using them to connect to APNS.

To verify the certificate and private key by connecting to APNS

- At a command prompt, type the following command. Replace `myapnsappcert.pem` and `myapnsappprivatekey.pem` with the name of your certificate and private key.

```
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert myapns
appcert.pem -key myapnsappprivatekey.pem
```

Step 5: Obtain a Device Token

When you register your app with APNS to receive push notification messages, a device token (64-byte hexadecimal value) is generated. The following steps describe how to use the sample iOS app provided by AWS to obtain a device token from APNS. You can use this sample iOS app to help you get started with Amazon SNS push notifications. For more information, see [Registering for Remote Notifications](#) in the Apple Local and Push Notification Programming Guide.

To obtain a device token from APNS for your app

- Download and unzip the [snsmobilepush.zip](#) file.
- Navigate to the `AppleMobilePushApp` folder and then open either the `iOS 7 and earlier` or `iOS 8` folder.
- In Xcode, open the `AmazonMobilePush.xcodeproj` project.
- Run the app in Xcode. In the output window, you should see the device token displayed, which is similar to the following:

```
Device Token = <example 29z6j5c4 df46f809 505189c4 c83fjcgf 7f6257e9 8542d2jt
3395kj73>
```

Note

Do not include spaces in the device token when submitting it to Amazon SNS.

Next Steps

You should now have the necessary information from APNS (SSL certificate, app private key, and device token) to send push notification messages to your mobile endpoint. You can now send a notification to the iOS app on your device by either using the Amazon SNS console or the Amazon SNS API.

- To send a notification to the iOS app on your device using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#).
- To use the Amazon SNS API, see [Send a push notification message to an iOS app using Amazon SNS and APNS \(p. 47\)](#).
- To send a push notification message to a VoIP app using Amazon SNS and APNS, see [Send a push notification message to a VoIP iOS app using Amazon SNS and APNS \(p. 49\)](#).
- To send a push notification message to a Mac OS X app using Amazon SNS and APNS, see [Send a push notification message to a Mac OS X app using Amazon SNS and APNS \(p. 49\)](#).

Send a push notification message to an iOS app using Amazon SNS and APNS

This section describes how to use the prerequisite information with the Amazon SNS API to send a push notification message to your iOS app using Amazon SNS and APNS. You add the prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file.

You can also use the Amazon SNS console. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#).

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*.

To add the sample to Eclipse

1. Create a new Java project in Eclipse (**File | New | Java Project**).
2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library...**
3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open `SNSMobilePush.java` in Eclipse.
2. Depending on which APNS you are using, uncomment either `sample.demoAppleAppNotification();` or `sample.demoAppleSandboxAppNotification();`

For example, if you're using `demoAppleSandboxAppNotification`, it should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);
// TODO: Uncomment the services you wish to use.
// sample.demoAndroidAppNotification();
// sample.demoKindleAppNotification();
// sample.demoAppleAppNotification();
sample.demoAppleSandboxAppNotification();
// sample.demoBaiduAppNotification();
// sample.demoWNSAppNotification();
// sample.demoMPNSAppNotification();
```

3. Locate the `demoAppleSandboxAppNotification` method and enter the device token you received from APNS for the value of the device token string. For example, it should look similar to the following:

```
String deviceToken = "example29z6j5c4df46f809505189c4c83fjcgf7f6257e98542d2jt3395kj73";
```

4. Enter the APNS SSL certificate for your app. At the beginning of each new line in your certificate, you must add `\n`. For example, it should look similar to the following:

```
String certificate = "-----BEGIN CERTIFICATE-----\nMIICiTCcAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBIDELMAkGALUEBhMC\nnVVMxCzAJBgNVBAGTAldBMRAdGyYDVQQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6\nnb24xFDASBgNVBASTC01BTSBDb25zb2x1MRIwEAYDVoQDEwLWZXR0Q2lsYW50Q2lsYWMxHZAAd\nBgcqhkiG9w0BCQEWEG5vb25lQGFtYXpvi5jb20wHhcNMTEwNDI0MjA0NTIxWjCB\niDELMAkGALUEBhMCVVMxCzAJBgNVBAGTAldBMRAdGyYD\nvVQHEwTZWFlwEAYDVoQDEwLWZXR0Q2lsYWMxHZAAdBgqhkiG9w0BCQEWEG5vb25l\nQGFtYXpvi5jb20wGZ8wDQYJKoZIhvcNAQEBBQADGyY0AMIGJAOGBAMAK0dn+a4Gm\nWIIWJ\nn21uUSwfEvySwTC2XADZ4nB+BLyGVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T\n\nnrDHu\n dUZg3qX4waLG5M43q7Wgc/MbQITxOUsQv7c7ugFFDzQGBzZswY6786m86gpE\n\nnIbb3OhjZnzcvQAaRHhdLQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4\n\nnnUhVVxYUnteD9+h8Mg9q6q+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb\n\nnFFBjvSfpJl1J00zbnYS5f6GuoEDmFJl0zxBHjJnyp378OD8uTs7fLv\n\njx79LjSTb\nnNYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=\n\n---END CERTIFICATE-----";
```

5. Enter the private key for your app. At the beginning of each new line in your certificate, you must add `\n`. For example, it should look similar to the following:

```
String privateKey = "-----BEGIN RSA PRIVATE KEY-----\nMJICiTcHAFIC\nCQD9m7oRw0uXOjANBgkqhkiG7w0BAQUFADCBIDELMAkGALUEBhMC\nnWVMxCzAJBgNVBAGTAldBMRAdGyYDVQQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6\n\nnVVMxCzAJBgNVBAGTAldBMRAdGyYDVQQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6\n\nn4MXNchZOFFreg4Rr3Xzhb9Rhv1IRgsr3wU4/FYai3z96EXAMPLE=\n\n---END RSA PRIVATE KEY-----";
```

6. Enter a name for 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. For example, it should look similar to the following:

```
String applicationName = "mypushappname";
```

7. Run the Java application. You should see output similar to the following in the output window of your IDE:

```
=====  
Getting Started with Amazon SNS  
=====
```

```
{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/APNS_SANDBOX/mypushappname}  
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/APNS_SANDBOX/pushapp/97e9ced9-f136-3893-9d60-775467eafebb}  
{"default":"This is the default Message","APNS_SANDBOX":{" \"aps\" : {  
  \"alert\" : \"You have got email.\", \"badge\" : 9,\"sound\" : \"default\"}}}"  
Published. MessageId=d65fb4bb-b903-5e37-83d4-feb4818d6da3
```

On your iOS device, you should see a message notification.

Send a push notification message to a VoIP iOS app using Amazon SNS and APNS

To send a push notification message to a VoIP app using Amazon SNS and APNS, you must first complete the prerequisites in [APNS Prerequisites](#) (p. 44).

Note

If you do not already have an iOS app registered with APNS, you can download and use the [snsmobilepush.zip](#) sample file provided by AWS as a template to get started. For more information, see [Step 5: Obtain a Device Token](#) (p. 46).

To register your mobile app with AWS

1. Go to <https://console.aws.amazon.com/sns/> and click **Create platform application**.
2. 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.

3. In the **Push Notification Platform** field, select **Apple Development** or **Apple Production**.
4. In the **Push Certification Type** field, select **VoIP Push Certificate**.
5. Select the password encrypted certificate and private key, as exported from Keychain Access on your Mac computer in the .p12 file format.
6. Enter your password, and then click **Create Platform Application**.

Send a push notification message to a Mac OS X app using Amazon SNS and APNS

To send a push notification message to a Mac OS X app using Amazon SNS and APNS, you must first complete the prerequisites in [APNS Prerequisites](#) (p. 44).

Note

If you do not already have a Mac OS X app registered with APNS, you can download and use a sample application such as [PushyMac](#), which is available from the Apple Developer web site.

To register your mobile app with AWS

1. Go to <https://console.aws.amazon.com/sns/> and click **Create platform application**.
2. 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.

3. In the **Push Notification Platform** field, select **Apple Development** or **Apple Production**.
4. In the **Push Certification Type** field, select **MacOS Push Certificate**.
5. Select the password encrypted certificate and private key, as exported from Keychain Access on your Mac computer in the .p12 file format.
6. Enter your password, and then click **Create Platform Application**.

Getting Started with Baidu Cloud Push

Baidu Cloud Push is a Chinese cloud service. Using Baidu, you can send push notification messages to mobile devices. This section describes how to obtain the Baidu prerequisites and send a push notification message using Amazon SNS and Baidu.

Topics

- [Baidu Prerequisites](#) (p. 50)
- [Step 1: Create a Baidu Account](#) (p. 51)
- [Step 2: Register as a Baidu Developer](#) (p. 52)
- [Step 3: Create a Baidu Cloud Push Project](#) (p. 56)
- [Step 4: Download and Install the Android Demo App from Baidu](#) (p. 59)
- [Step 5: Obtain a User Id and Channel Id from Baidu](#) (p. 63)
- [Step 6: Send a Push Notification Message to a Mobile Endpoint using Amazon SNS and Baidu](#) (p. 63)

Baidu Prerequisites

To send a push notification message to mobile devices using Amazon SNS and Baidu, you need the following:

- Baidu account
- Registration as a Baidu developer
- Baidu cloud push project
- API key and secret key from a Baidu cloud push project
- Baidu user ID and channel ID
- Android demo app

If you already have these prerequisites, then you can send a push notification message to a mobile endpoint using the Amazon SNS API. For more information about using the Amazon SNS API, see [Step 6: Send a Push Notification Message to a Mobile Endpoint using Amazon SNS and Baidu](#) (p. 63).

Step 1: Create a Baidu Account

To use Baidu, you must first create an account.

Important

In order to create a Baidu account there is a verification step where you must enter Chinese Simplified characters. The easiest way to accomplish this task is for someone that can read Chinese to assist. Another option is to use Amazon Mechanical Turk for creating the Baidu account. Once you have the account and password created for Baidu, you could login and change the password without needing to enter Chinese Simplified characters. For more information about Mechanical Turk, see the [Amazon Mechanical Turk Requester User Interface](#).

To create a Baidu account

1. On the [Baidu Portal](#), in the top right corner, click **(Registration)**.

搜索设置 登录 **注册**



2. Enter an email address, password, and verification code, and then click **(Registration)**.



You should then see a page similar to the following, informing you that an activation email has been sent to the email address you entered.

Amazon Simple Notification Service Developer Guide Step 2: Register as a Baidu Developer



3. Login to your email account, open the activation email you received from Baidu, and click the provided link:



4. After you click the provided link in the activation email from Baidu, you must then enter the verification code (Chinese Simplified characters).



Once you have created a Baidu account, you can then register as a developer.

Step 2: Register as a Baidu Developer

You must register as a Baidu developer to use the Baidu push notification service.

To register as a Baidu developer

1. On the [Baidu Portal](#), click **(More)**.



2. Click **(Baidu's Open Cloud Platform)**



3. On the next page, near the top right corner, click **(Developer Services)**.



4. Click **(Start Now)**



Amazon Simple Notification Service Developer Guide

Step 2: Register as a Baidu Developer

5. Enter your name, description, and mobile phone number for receiving a verification text message, and then click **(Send Verification Code)**.

填写开发者信息 验证邮箱 提交应用 / 使用开放云服务

成为一名百度开发者需要您完善以下信息，以便使用强大的应用分发渠道和丰富的开放云服务

* 类型： 个人 公司

* 开发者来源：

* 开发者姓名：

* 开发者简介：

* Email地址： [修改](#)

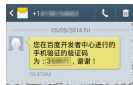
* 手机号：

* 验证码：

开发者官方网站：

品牌LOGO： [Web](#)

You should then receive a text message with a verification number, similar to the following:



6. Complete the developer registration by entering the verification number and then click **(Submit)** on the bottom of the page.

Amazon Simple Notification Service Developer Guide Step 2: Register as a Baidu Developer

The screenshot shows the registration form for Baidu Open Cloud. At the top, there are three steps: '填写开发者信息' (Fill in developer information), '验证邮箱' (Verify email), and '提交应用 / 使用开放云服务' (Submit application / Use open cloud services). The first step is active. Below the steps, there is a message: '成为一名百度开发者需要您完善以下信息, 以便使用强大的应用分发渠道和丰富的开放云服务' (To become a Baidu developer, you need to complete the following information to use powerful application distribution channels and rich open cloud services). The form fields include: '类型' (Type) with radio buttons for '个人' (Personal) and '公司' (Company); '开发者来源' (Developer source) with a dropdown menu; '开发者姓名' (Developer name) with a text input field; '开发者简介' (Developer introduction) with a text area; 'Email地址' (Email address) with a text input field and a '修改' (Modify) link; '手机号' (Mobile phone number) with a text input field and a '发送验证码' (Send verification code) button; '验证码' (Verification code) with a text input field; '开发者官方网站' (Developer official website) with a text input field; and '品牌LOGO' (Brand logo) with a note: '112px*54px, 支持PNG/JPG/GIF格式, 应用提交至PC' (112px*54px, supports PNG/JPG/GIF format, application submitted to PC).

Upon successful registration, you should see the following:



After registering as a Baidu developer, you can then proceed to the next step to create a Baidu cloud push project. This assumes that you are still logged in. If you are not logged in, then you can use the following login procedure.

To login to Baidu

1. On the [Baidu Portal](#), in the top right corner, click **(Login)**.

Amazon Simple Notification Service Developer Guide Step 3: Create a Baidu Cloud Push Project

[搜索设置](#) [登录](#) [注册](#)



2. Enter your Baidu username (email address) and password and then click **(Login)**.



Step 3: Create a Baidu Cloud Push Project

When you create a Baidu cloud push project, you receive your app ID, API key, and secret key.

To create a Baidu cloud push project

1. On the [Baidu Portal](#), click **(More)**.



2. Click **(Baidu's Open Cloud Platform)**

Amazon Simple Notification Service Developer Guide Step 3: Create a Baidu Cloud Push Project



3. On the next page, near the top right corner, click **(Developer Services)**.



4. Click **(Cloud Push)**.



5. Click **(Management Console)**.



6. Click **(Management Console)** to enter information for an Android project.

Amazon Simple Notification Service Developer Guide Step 3: Create a Baidu Cloud Push Project



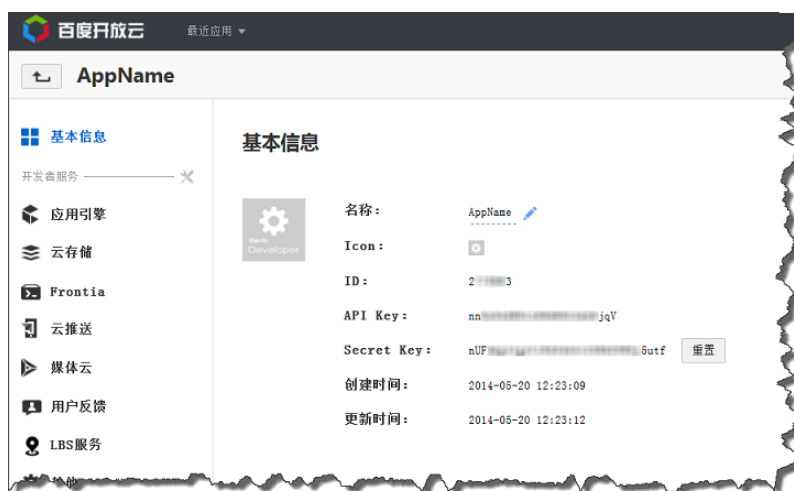
7. Click **(Create Project)**.



8. Enter an app name and then click **(Create)**.



- Upon successful completion of the project, you will then see a page similar to the following with your **app ID**, **API Key**, and **Secret Key**. Make note of the API key and secret key, as they will be needed later.



Step 4: Download and Install the Android Demo App from Baidu

Baidu generates an Android demo app that you can download and install to your mobile device.

To download and install the Android demo app from Baidu

- Starting from the page that displays the app ID, API Key, and Secret Key, click **(Cloud Push)**

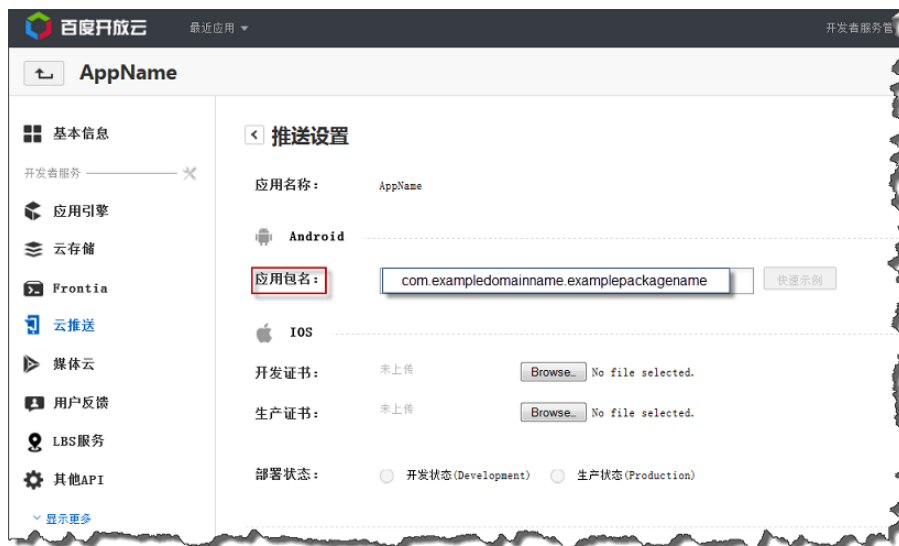
Amazon Simple Notification Service Developer Guide Step 4: Download and Install the Android Demo App



2. Click **(Push Settings)**

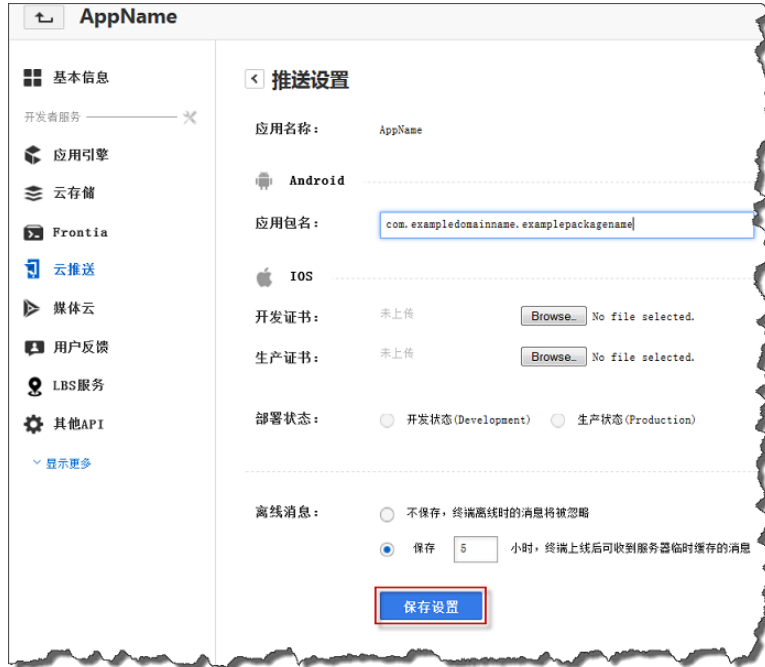


3. Using reverse domain name notation, enter a package name in the **(App Package Name)** box.



4. Click **(Save Settings)**

Amazon Simple Notification Service Developer Guide Step 4: Download and Install the Android Demo App



You should then see the **(Successfully saved!)** message displayed.



- Next, click **(Quick Example)**.



You should then see a page similar to the following:



6. On the Android mobile device you want to test with, scan the QR code icon using a code scanner, such as [QR Droid](#), to get a link to a demo app provided by Baidu.

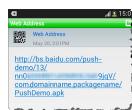
Note

You can also download the demo app by clicking **Android (Download Android Sample)**

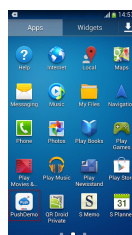


The Baidu Push Demo app is included in the downloaded `PushDemo.zip` package. You can use the demo app as an example for creating your own app to use with Baidu. In addition, the push service jar file (`pushservice-4.0.0.jar`) from Baidu is included in the `PushDemo.zip` package. You must use the jar file from Baidu in order to create a new app.

7. Click the link you receive after scanning the scan code. This will download the demo app provided by Baidu onto your mobile device.



8. After the download has completed, install the demo app onto your mobile device. You should then see the following **Push Demo** app installed:



Step 5: Obtain a User Id and Channel Id from Baidu

Baidu generates a user Id and channel Id that you will need to send a push notification message using Baidu.

To obtain the user Id and channel Id from Baidu

1. Open **Push Demo** and then click, in the bottom right, **(Bind Without Baidu Account)**.



You should then see a screen similar to the following with the **userId** and **channelId**.



2. Make a note of the **userId** and **channelId**, as you will be using them in the next step.

Note

For an example of Java code that is used to retrieve the `userId` and `channelId`, see the `onBind` method in the `MyPushMessageReceiver.java` file of the Push Demo app from Baidu. For more information, see the [Android integration guide](#). To translate this guide into English, you can paste the URL, <http://developer.baidu.com/wiki/index.php?title=docs/cplat/push/guide>, into [Bing Translator](#) and then click **Translate**.

Step 6: Send a Push Notification Message to a Mobile Endpoint using Amazon SNS and Baidu

This section describes how to send a push notification message to your mobile endpoint. You add the gathered prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file. Included in the `SNSMobilePush.java` file are examples on how to create a mobile endpoint and use message attributes for structuring the message. For additional information and examples on how to create mobile endpoints and use message attributes with Baidu, see [Creating an Amazon SNS Endpoint for Baidu \(p. 65\)](#) and [Using Message Attributes for Structuring the Message \(p. 66\)](#).

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*.

To add the sample to Eclipse

1. In Eclipse, create a new Java project (**File | New | Java Project**).

Amazon Simple Notification Service Developer Guide
Step 6: Send a Push Notification Message to a Mobile
Endpoint using Amazon SNS and Baidu

2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library....**
3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open `SNSMobilePush.java` in Eclipse and uncomment `sample.demoBaiduAppNotification()`; . It should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);
// TODO: Uncomment the services you wish to use.
//sample.demoAndroidAppNotification();
//sample.demoKindleAppNotification();
//sample.demoAppleAppNotification();
//sample.demoAppleSandboxAppNotification();
sample.demoBaiduAppNotification();
//sample.demoWNSAppNotification();
//sample.demoMPNSAppNotification();
```

2. Locate the `demoBaiduAppNotification` method and enter the user ID and channel ID you received from Baidu for the value of the `userId` and `channelId` strings. For example, it should look similar to the following:

```
String userId = "EXAMPLE-kLMchcX0v3x0xWVhG6TfdBp...KT2TGkvnKyTvLuS
pzK_qsHgxVB_UpmcUa7G16g3EXAMPLE";
String channelId = "EXAMPLE<channelId>EXAMPLE";
```

3. Enter the secret key for your application. For example, it should look similar to the following:

```
String secretKey = "EXAMPLE<secretkey>EXAMPLE";
```

4. Enter the API key for your application. For example, it should look similar to the following:

```
String apiKey = "EXAMPLExV21cV2zEKTLNys625zfk2jh4EXAMPLE";
```

5. Enter a name for your application. Application 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. For example, it should look similar to the following:

```
String applicationName = "baidupushapp";
```

Amazon Simple Notification Service Developer Guide

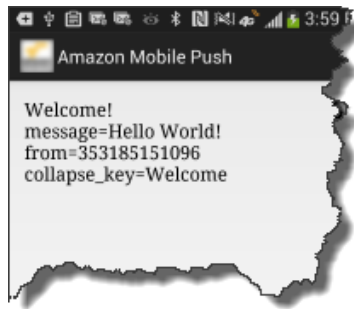
Step 6: Send a Push Notification Message to a Mobile Endpoint using Amazon SNS and Baidu

- Run the application. You should see output similar to the following in the output window of your IDE:

```
=====
Getting Started with Amazon SNS
=====

{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/BAIDU/TestApp}
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/BAIDU/Test
App/8f3fdf0d-520b-38d1-8ed2-3301a477eef3}
{Message Body: {"BAIDU":{"title":"New Notification Received from
SNS","description":"Hello World!"}}}
{Message Attributes: {"AWS.SNS.MOBILE.BAIDU.MessageKey": "default-channel-
msg-key"}, {"AWS.SNS.MOBILE.BAIDU.DeployStatus": "1"}, {"AWS.SNS.MO
BILE.BAIDU.MessageType": "0"}}
Published!
{MessageId=56a3a3e6-4b4b-59b4-8d1d-eff592c0ffa1}
```

On your Android device, you should see a push notification message appear within the Android app, similar to the following:



Creating an Amazon SNS Endpoint for Baidu

This section provides additional information and examples on how to create an Amazon SNS endpoint to use with Baidu. You create an Amazon SNS endpoint, using the combined `userId` and `channelId` received from Baidu, to represent the app and mobile device. The endpoint is then used by Amazon SNS for publishing notification messages using the Baidu push notification service to the app on the mobile device.

The following Java example shows how to create an Amazon SNS endpoint for a Baidu app and mobile device.

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

// Insert your UserId. This is a mandatory field.
attributes.put("UserId", "9999999999");

// Insert your ChannelId. This is a mandatory field.
attributes.put("ChannelId", "1234567890");

CreatePlatformEndpointRequest createPlatformEndpointRequest = new CreatePlatfor
```

Amazon Simple Notification Service Developer Guide
Step 6: Send a Push Notification Message to a Mobile
Endpoint using Amazon SNS and Baidu

```
mEndpointRequest();

// Baidu endpoints are identified by a combination of the userId and channelId
// which must be supplied as endpoint attributes,
// without which a valid endpoint cannot be successfully created.
createPlatformEndpointRequest.setAttributes(attributes);

// Insert your ChannelId. This is a mandatory field.
createPlatformEndpoint.setPlatformToken("1234567890");

// Insert your Customer User Data. This is an optional field.
createPlatformEndpoint.setCustomUserData("Test Endpoint");

// Insert your Platform Application Arn. This is a mandatory field.
createPlatformEndpoint.setPlatformApplicationArn("arn:aws:sns:us-west-
2:123456789012:app/BAIDU/TestApp");
String endpointArn = snsClient.createPlatformEndpoint(createPlatformEndpointRe
quest);
```

Note the following considerations when using the Amazon SNS API to create an endpoint for use with Baidu:

- In `CreateEndpointRequest`, the platform token field should contain the channelId.
- If you specify the endpoint attribute "Token" in the attributes map, this field must encapsulate the channelId as well.
- The channelId should also be specified as an endpoint attribute with the name "ChannelId".
- The value of the "ChannelId" endpoint attribute and the platform token field and/or "Token" endpoint attribute must be identical to construct a successful request.
- The userId should be specified as an endpoint attribute with the name "UserId".
- For a successful response, the request must contain valid UserId and ChannelId values in the attributes. Also, the ChannelId parameter entered using `setPlatformToken(String)`, which is a part of `CreatePlatformEndpointRequest`, must be the same as the ChannelId specified in the attributes map.

Using Message Attributes for Structuring the Message

This section provides additional information and examples for using message attributes to structure a message and send a push notification message to a mobile endpoint.

The following Java example shows how to send a push notification message to a mobile endpoint and how to use the optional message attributes for structuring the message. If an attribute is not sent, a default value is auto-set in its place.

Note

The push notification message cannot exceed 256 bytes, which is the maximum size allowed by Baidu.

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

// Insert your desired value of Deploy Status here. 1 = DEV, 2 = PROD
messageAttributes.put("AWS.SNS.MOBILE.BAIDU.DeployStatus", new MessageAttribute
Value().withDataType("String").withStringValue("1"));

// Insert your desired value of Message Type here. 0 = IN-APP MESSAGE, 1 = ALERT
NOTIFICATION
```



```
messageAttributes.put("AWS.SNS.MOBILE.BAIDU.MessageType", new MessageAttribute  
Value().withDataType("String").withStringValue("1"));  
  
// Insert your desired value of Message Key  
messageAttributes.put("AWS.SNS.MOBILE.BAIDU.MessageKey", new MessageAttribute  
Value().withDataType("String").withStringValue("test-message"));  
  
PublishRequest publishRequest = new PublishRequest();  
publishRequest.setMessageAttributes(messageAttributes);  
String message = "{\"title\":\"Test_Title\",\"description\":\"Test_Descrip  
tion\"}";  
publishRequest.setMessage(message);  
publishRequest.setTargetArn("arn:aws:sns:us-west-2:999999999999:end  
point/BAIDU/TestApp/309fc7d3-bc53-3b63-ac42-e359260ac740");  
PublishResult publishResult = snsClient.publish(publishRequest);
```

Note the following considerations when using the optional message attributes for structuring the message:

- `AWS.SNS.MOBILE.BAIDU.DeployStatus`

Possible Values (Default = 1):

- 1 – Tags the notification as being sent in a development environment
- 2 – Tags the notification as being sent in a production environment

- `AWS.SNS.MOBILE.BAIDU.MessageType`

Possible Values (Default = 1):

- 0 – Generates an in-app message
- 1 – Generates an alert notification. Alert notifications are restricted to the following format:

```
{"title": "<TITLE>", "description": "<DESCRIPTION>" }
```

<TITLE> and <DESCRIPTION> are the title and description you desire for your alert notification. If the message is incorrectly formatted JSON, the request fails.

- `AWS.SNS.MOBILE.BAIDU.MessageKey`

A short message identifier you can attach to your message

Getting Started with Google Cloud Messaging for Android

Google Cloud Messaging for Android (GCM) is a service that enables you to send push notification messages to an Android app. This section describes how to obtain the GCM prerequisites and send a push notification message to a mobile endpoint.

Topics

- [GCM Prerequisites \(p. 68\)](#)
- [Step 1: Create a Google API Project and Enable the GCM Service \(p. 68\)](#)
- [Step 2: Obtain the Server API Key \(p. 68\)](#)
- [Step 3: Obtain a Registration ID from GCM \(p. 69\)](#)

- [Step 4: Send a Push Notification Message to a Mobile Endpoint using GCM \(p. 70\)](#)

GCM Prerequisites

To send push notification messages to an Android app, you need the following:

- Android app registered with GCM
- Registration ID
- Server API key (sender auth token)

If you already have these prerequisites, then you can either use the Amazon SNS console to send a push notification message to the mobile endpoint or you can use the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#). For more information about using the Amazon SNS API, see [Step 4: Send a Push Notification Message to a Mobile Endpoint using GCM \(p. 70\)](#).

Step 1: Create a Google API Project and Enable the GCM Service

To send an push notification message to an Android app, you must have a Google API project and enable the GCM service.

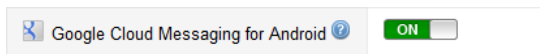
To create a Google API project and enable the GCM service

1. If you do not already have a Google API project, then see the [Creating a Google API project](#) in the Android developer documentation.

Note

If you do not already have an Android app registered with GCM, then you can use the sample Android app provided by AWS as a template to get started. For more information, see [Step 3: Obtain a Registration ID from GCM \(p. 69\)](#).

2. On the [Google APIs Console web site](#), verify that you have a Google API project.
3. Click **Services**, and make sure **Google Cloud Messaging for Android** is turned on.



Step 2: Obtain the Server API Key

To communicate with GCM on your behalf, Amazon SNS uses your server API key. This key will be used in a later step to send a push notification to a mobile endpoint.

To obtain the server API key

1. On the [Google APIs Console web site](#), click **API Access** and make note of the server API key with the **Key for server apps (with IP locking)** label.
2. If you have not yet created a server API key, then click **Create new Server key**. This key will be used later in this section to send a push notification to a mobile endpoint.

Step 3: Obtain a Registration ID from GCM

When you register your app with GCM to receive push notification messages, a registration ID is generated. Amazon SNS uses this value to determine which app and associated device to send mobile push notifications to.

The following steps show how to use the sample Android app provided by AWS to obtain a registration ID from GCM. You can use this sample Android app to help you get started with Amazon SNS push notifications. This sample app requires the Android SDK, the Google Play Services SDK, and the Android Support Library package. For more information about these SDKs, see [Get the Android SDK](#) and [Setup Google Play Services SDK](#). For more information about the Android Support Library package, see [Support Library Setup](#).

Note

The provided sample Android app is compatible with physical devices running Android version 2.3 or later and with virtual devices running Google API 17 or later.

To obtain a registration ID from GCM for your app

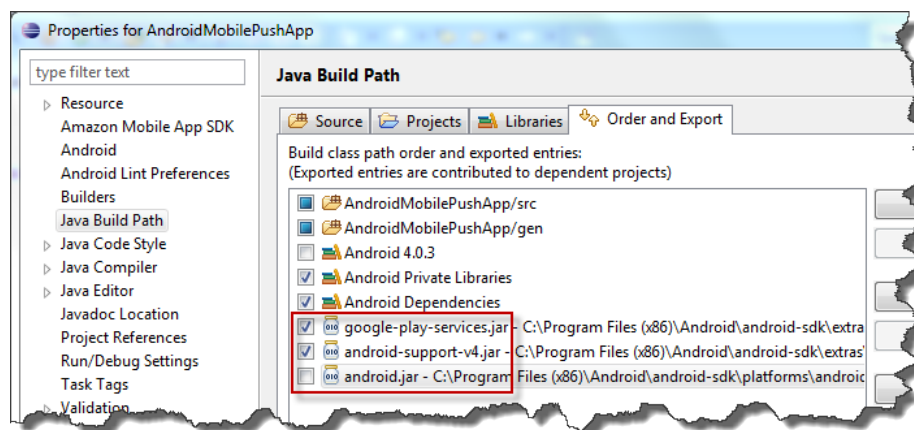
1. Download and unzip the [snsmobilepush.zip](#) file.
2. Import the `AndroidMobilePushApp` folder into your IDE. In Eclipse, click **File, Import**, expand the **Android** folder, click **Existing Android Code Into Workspace**, click **Next**, browse to the folder `AndroidMobilePushApp`, click **OK**, and then click **Finish**.

After the sample Android app has been imported into your IDE, you need to add the Project Number for your Google API project to the `strings.xml` file, which is included in the sample Android app.

3. Add the Project Number for your Google API project to the `strings.xml` file. In your IDE, you will find the file included in the **values** folder, which is a subfolder of **res**. The string will look similar to the following:

```
<string name="project_number">012345678912</string>
```

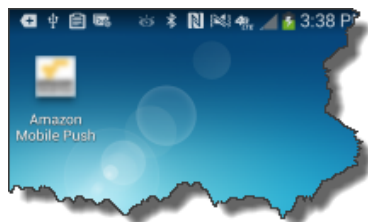
4. Add `google-play-services.jar`, `android-support-v4.jar`, and `android.jar` to the Java Build Path. Select `google-play-services.jar` and `android-support-v4.jar` for export, but do not select `android.jar` for export.



5. Run the app to see the registration ID as output to the Android logging system. If you are using Eclipse with the Android ADT plug-in, you can see the registration ID in the **LogCat** display window. For example, the output containing the registration ID will look similar to the following:

```
06-05 11:50:43.587: V/Registration(14146): Registered, registrationId: =  
Examplei7fFachkJ1xj1qT64RaBkcGHochmf1VQAr9k-IB  
JtKjP7fedYPzEwT_Pq3Tu01roqrolcwWJUvgkcPPYcaXCpPwMg3Bqn-  
wiqIEzp5zZ7y_jsM0PKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw, error = null,  
unregistered = null
```

The installed app will appear on your Android device:



You should now have a registration ID, server API key, and Android app registered with GCM. You can now send a notification to the Android app on your device by either using the Amazon SNS console or the Amazon SNS API. To use the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#). To use the Amazon SNS API, see [Step 4: Send a Push Notification Message to a Mobile Endpoint using GCM \(p. 70\)](#).

Step 4: Send a Push Notification Message to a Mobile Endpoint using GCM

This section describes how to send a push notification message to your mobile endpoint. You add the gathered prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file.

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*.

To add the sample to Eclipse

1. In Eclipse, create a new Java project (**File | New | Java Project**).
2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples/src/com/amazonaws/sns/samples/mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library...**
3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples/src/com/amazonaws/sns/samples/mobilepush` folder, open `SNSMobilePush.java` in Eclipse and uncomment `sample.demoAndroidAppNotification()`. It should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);
// TODO: Uncomment the services you wish to use.
sample.demoAndroidAppNotification();
// sample.demoKindleAppNotification();
// sample.demoAppleAppNotification();
// sample.demoAppleSandboxAppNotification();
// sample.demoBaiduAppNotification();
// sample.demoWNSAppNotification();
// sample.demoMPNSAppNotification();
```

2. Locate the `demoAndroidAppNotification` method and enter the registration ID you received from GCM for the value of the registration ID string. For example, it should look similar to the following:

```
String registrationId = "EXAMPLE-kLMchcX0v3xOxWVhG6TfdBp...KT2TGkvnKyTvLuS
pzK_qsHgxB_UpmcUa7G16g3EXAMPLE";
```

3. Enter the API key for your application. For example, it should look similar to the following:

```
String serverAPIKey = "EXAMPLExV21cV2zEKTLNys625zfk2jh4EXAMPLE";
```

4. Enter a name for your application. Application 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. For example, it should look similar to the following:

```
String applicationName = "gcmpushapp";
```

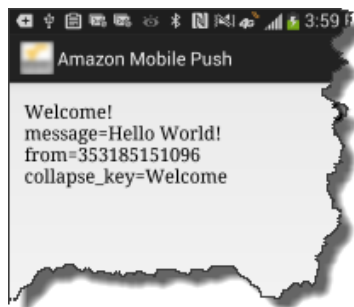
5. Run the application. You should see output similar to the following in the output window of your IDE:

```
=====
Getting Started with Amazon SNS
=====

{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/GCM/gcmpush
app}
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/GCM/gcmpush
app/5e3e9847-3183-3f18-a7e8-671c3a57d4b3}
{"default":"This is the default mes
```

```
sage", "GCM": {"delay_while_idle": true, "collapse_key": "Welcome", "data": {"message": "Visit Amazon!", "url": "http://www.amazon.com/"}, "time_to_live": 125, "dry_run": false}}  
Published. MessageId=1ca8d7d1-c261-5bfc-8689-9db269c4e46c
```

On your Android device, you should see a push notification message appear within the Android app, similar to the following:



Getting Started with MPNS

Microsoft Push Notification Service for Windows Phone (MPNS) is a service that enables you to send push notification messages to Windows Phone 7+ and Windows Phone 8.0 apps. This section describes how to obtain the MPNS prerequisites and send a push notification message using Amazon SNS and MPNS. You can send both unauthenticated and authenticated push notification messages with MPNS. For better security and to avoid throttling limits imposed by MPNS, you should send authenticated push notification messages.

Topics

- [MPNS Prerequisites \(p. 72\)](#)
- [Step 1: Set Up Your Windows Phone App to Receive Push Notifications Messages \(p. 73\)](#)
- [Step 2: Get a Push Notification URI from MPNS \(p. 73\)](#)
- [Step 3: Create a Windows Developer Account \(p. 73\)](#)
- [Step 4: Upload TLS Certificate \(p. 73\)](#)
- [Step 5: Send a Push Notification Message to a Windows Phone app using Amazon SNS and MPNS \(p. 73\)](#)

MPNS Prerequisites

To send an unauthenticated push notification message to a Windows Phone app using Amazon SNS and MPNS, you need the following:

- Windows Phone app configured to use MPNS
- Push notification URI from MPNS

To send an authenticated push notification message to a Windows Phone app using Amazon SNS and MPNS, you also need the following:

- HTTPS Push notification URI from MPNS
- Registered as a Windows app developer
- Transport Layer Security (TLS) certificate

If you already have these prerequisites, then you can send a push notification message to a Windows Phone app using either the Amazon SNS console or the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#). For more information about using the Amazon SNS API, see [Step 5: Send a Push Notification Message to a Windows Phone app using Amazon SNS and MPNS \(p. 73\)](#).

Step 1: Set Up Your Windows Phone App to Receive Push Notifications Messages

To send a push notification message to your Windows Phone app, you must enable the app for the MPNS service. For more information, see [Setting up your app to receive push notifications for Windows Phone 8](#).

Step 2: Get a Push Notification URI from MPNS

To create a mobile endpoint with Amazon SNS you need a push notification URI from MPNS. You can either get an HTTP or HTTPS push notification URI from MPNS. For better security and to avoid throttling limits imposed by MPNS, you should get an HTTPS push notification URI for sending authenticated messages. For more information about getting an HTTPS push notification URI, see [Setting up an authenticated web service to send push notifications for Windows Phone 8](#).

Step 3: Create a Windows Developer Account

To send authenticated messages using MPNS you must create a Windows developer account. For more information about opening a Windows developer account, see [Opening a developer account](#).

Step 4: Upload TLS Certificate

To send authenticated messages using MPNS, you must upload a TLS certificate obtained from one of the trusted certificate authorities (CA) for Windows Phone to your Windows developer account. You must also submit the complete TLS certificate chain and associated private key to Amazon SNS. This is to help with establishing a secure connection to MPNS with Amazon SNS on your behalf. Amazon SNS requires the TLS certificate and private key in the .pem format. You can use different utilities, such as openssl, for converting and exporting certificates. For more information, see [Setting up an authenticated web service to send push notifications for Windows Phone 8](#) and [SSL root certificates for Windows Phone OS 7.1](#). For more information about openssl, see <http://www.openssl.org/>.

Step 5: Send a Push Notification Message to a Windows Phone app using Amazon SNS and MPNS

This section describes how to use the prerequisite information with the Amazon SNS API to send a push notification message to your Windows Phone app using Amazon SNS and MPNS. You add the gathered prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file.

You can also use the Amazon SNS console. However, to send toast notifications, you must use the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push \(p. 78\)](#).

Amazon Simple Notification Service Developer Guide

Step 5: Send a Push Notification Message to a Windows Phone app using Amazon SNS and MPNS

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*.

To add the sample to Eclipse

1. In Eclipse, create a new Java project (**File | New | Java Project**).
2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library...**
3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open `SNSMobilePush.java` in Eclipse and uncomment `sample.demoMPNSAppNotification()`; It should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);
// TODO: Uncomment the services you wish to use.
//sample.demoAndroidAppNotification();
//sample.demoKindleAppNotification();
//sample.demoAppleAppNotification();
//sample.demoAppleSandboxAppNotification();
//sample.demoBaiduAppNotification();
//sample.demoWNSAppNotification();
sample.demoMPNSAppNotification();
```

2. Locate the `demoMPNSAppNotification` method and enter the notification URI you received from MPNS for the value of the `notificationChannelURI` string.
3. Enter a name for your application. Application 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. For example, it should look similar to the following:

```
String applicationName = "mpnspushapp";
```

4. Enter the MPNS TLS certificate in `.pem` file format. You must include the complete certificate chain, beginning with the root CA certificate at the top and ending with the issued certificate at the bottom. At the beginning of each new line in your certificate, you must add `\n`. For example, it should look similar to the following:


```
String certificateChain = "-----BEGIN CERTIFICATE-----\nMIICiTCCAFIC  
CQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBIDELMAkGALUEBhMC\nvVMxCzAJBgNVBAGTAldB  
MRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6\nnb24xFDASBgNVBAStC01BTSB  
Db25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWx1eXZAd\nnBgkqhkiG9w0BCQEWEG5vb25lQGFTYX  
pvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN\nnMTIwNDI0MjA0NTIxWjCBiDELMAkGALUEBhMCVVMx  
CzAJBgNVBAGTAldBMRAdGyYD\nvVQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6b24xFDASBgN  
VBAsTC01BTSBDb25z\nnb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWx1eXZAd  
BgkqhkiG9w0BCQEWEG5vb25lQGFT\nnyXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIG  
JAoGBAMaK0dn+a4GmWIWJ\nn21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyN  
oH/f0wYK8m9T\nnrDHudUZg3qX4waLG5M43q7Wgc/MbQITxOUsQv7c7ugFFDzQGBzZ  
swY6786m86gpe\nnIbb3OhjZncvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEF  
BQADgYEAtCu4\nnnUhVVxYUntneD9+h8Mg9q6q+auNKyExzyLwaxlAoo7TJHidbtS4J5iN  
mZgXL0Fkb\nnFFBjvSfpJiLJ00zbbNYS5f6GuoEDmFJl0ZxB  
HjJnyP378OD8uTs7fLvJx79LjStb\nnNYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=\n--  
---END CERTIFICATE-----";
```

5. Enter the private key for the MPNS TLS certificate in .pem file format. At the beginning of each new line in your certificate, you must add \n. For example, it should look similar to the following:

```
String privateKey = "-----BEGIN RSA PRIVATE KEY-----\nMJICiTCHAFIC  
CQD9m7oRw0uXOjANBgkqhkiG7w0BAQUFADCBIDELMAkGALUEBhMC\nwVMxCzAJBgNVBAGTAldB  
MRAwDgYDVQQHEwdTZWF0dGx2MQ8wDQYDVQKKEwZBbWF6\nnb24xFDASBgNVBAStC01BTSB  
Db25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWx1eXZAd\nnBgkqhkiG9w0BCQEWEG5vb25lQGFTYX  
pvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN\nnMTIwNDI0MjA0NTIxWjCBiDELMAkGALUEBhMCVVMx  
CzAJBgNVBAGTAldBMRAdGyYD  
vVQHEwdTZWF0dGx1MQ8wDQYDVQKKEwZBbWF6\nn4MXNchZOFFreg4Rr3Xzhh9Rvh  
lIRgsr3wU4/FYai3z96EXAMPLE=\n-----END RSA PRIVATE KEY-----";
```

6. Run the application. You should see output similar to the following in the output window of your IDE:

```
=====  
Getting Started with Amazon SNS  
=====
```

```
{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/MPNS/TestApp}  
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/MPNS/Test  
App/557597f8-be4a-3035-8c6d-bb7fa8b20fef}  
{Message Body: {"MPNS": "<?xml version='1.0' encoding='utf-8'><wp:Noti  
fication xmlns:wp='WPNotifica  
tion'><wp:Tile><wp:Count>23</wp:Count><wp:Title>This is a tile notifica  
tion</wp:Title></wp:Tile></wp:Notification"}}
```

```
{Message Attributes: ("AWS.SNS.MOBILE.MPNS.Type": "token"), ("AWS.SNS.MO  
BILE.MPNS.NotificationClass": "realtime")}  
Published!  
{MessageId=ce9855bf-395f-5a1a-a4b9-19ace305780d}
```

On your Windows Phone, you should see a push notification message appear within the app.

Getting Started with WNS

Windows Push Notification Services (WNS) is a service that enables you to send push notification messages and updates to Windows 8 (and later) and Windows Phone 8.1 (and later) apps. This section describes how to obtain the WNS prerequisites and send a push notification message using Amazon SNS and WNS.

Topics

- [WNS Prerequisites](#) (p. 76)
- [Step 1: Set Up Your App to Receive Push Notifications Messages](#) (p. 76)
- [Step 2: Get a Push Notification URI from WNS](#) (p. 76)
- [Step 3: Get a Package Security Identifier from WNS](#) (p. 76)
- [Step 4: Get a Secret Key from WNS](#) (p. 76)
- [Step 5: Send a Push Notification Message to an App using Amazon SNS and WNS](#) (p. 77)

WNS Prerequisites

To send push notification messages to Windows devices using Amazon SNS and WNS, you need the following:

- Windows 8 (and later) or Windows Phone 8.1 app configured to use WNS
- Push notification URI from WNS
- Package security identifier
- Secret key

If you already have these prerequisites, then you can send a push notification message to an app using either the Amazon SNS console or the Amazon SNS API. For more information about using the Amazon SNS console, see [Using Amazon SNS Mobile Push](#) (p. 78). For more information about using the Amazon SNS API, see [Step 5: Send a Push Notification Message to an App using Amazon SNS and WNS](#) (p. 77).

Step 1: Set Up Your App to Receive Push Notifications Messages

To send push notification message to your app, you must enable the app for the WNS service. For more information, see [Windows Push Notification Services](#).

Step 2: Get a Push Notification URI from WNS

To create a mobile endpoint with Amazon SNS, you need a push notification URI from WNS. For more information, see [Windows Push Notification Services](#).

Step 3: Get a Package Security Identifier from WNS

To create a mobile endpoint with Amazon SNS, you need a package security identifier from WNS. For more information, see [Windows Push Notification Services](#).

Step 4: Get a Secret Key from WNS

To create a mobile endpoint with Amazon SNS, you need a secret key from WNS. For more information, see [Windows Push Notification Services](#).

Step 5: Send a Push Notification Message to an App using Amazon SNS and WNS

This section describes how to use the prerequisite information to send a push notification message to your app using Amazon SNS and WNS. You add the gathered prerequisite information to the AWS sample file `SNSMobilePush.java`, which is included in the `snsmobilepush.zip` file.

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*.

To add the sample to Eclipse

1. In Eclipse, create a new Java project (**File | New | Java Project**).
2. Import the `SNSSamples` folder to the top-level directory of the newly created Java project. In Eclipse, right-click the name of the Java project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **Finish**.
3. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open the `AwsCredentials.properties` file and add your AWS security credentials.

To add the AWS SDK for Java to the Build Path

1. Right-click the Java project folder, click **Build Path**, and then click **Configure Build Path...**
2. Click the Libraries tab, and then click **Add Library...**
3. Click **AWS SDK for Java**, click **Next**, and then click **Finish**.

To add the prerequisite information to `SNSMobilePush.java`

1. In the `SNSSamples\src\com\amazonaws\sns\samples\mobilepush` folder, open `SNSMobilePush.java` in Eclipse and uncomment `sample.demoWNSAppNotification()`. It should look similar to the following:

```
SNSMobilePush sample = new SNSMobilePush(sns);  
// TODO: Uncomment the services you wish to use.  
//sample.demoAndroidAppNotification();  
//sample.demoKindleAppNotification();  
//sample.demoAppleAppNotification();  
//sample.demoAppleSandboxAppNotification();  
//sample.demoBaiduAppNotification();  
sample.demoWNSAppNotification();  
//sample.demoMPNSAppNotification();
```

2. Locate the `demoWNSAppNotification` method and enter the string values for the push notification URI, package security identifier, and secret key.
3. Enter a name for your application. Application 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. For example, it should look similar to the following:

```
String applicationName = "wnspushapp";
```

4. Run the application. You should see output similar to the following in the output window of your IDE:

```
=====
Getting Started with Amazon SNS
=====

{PlatformApplicationArn: arn:aws:sns:us-west-2:111122223333:app/WNS/TestApp}
{EndpointArn: arn:aws:sns:us-west-2:111122223333:endpoint/WNS/Test
App/17cc2f2a-dfa8-3450-90c6-e1f88d820f3d}
{Message Body: {"WNS": "<badge version=\"1\" value=\"23\" />"}}
{Message Attributes: ("AWS.SNS.MOBILE.WNS.Type": "wns/badge"), ("AWS.SNS.MO
BILE.WNS.CachePolicy": "cache")}
Published!
{MessageId=d4899281-927e-5f68-9fd0-de9248be6d47}
```

On your Windows device, you should see a push notification message appear within the app.

Using Amazon SNS Mobile Push

This section describes how to use the AWS Management Console with the information described in [Prerequisites \(p. 37\)](#) 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 \(p. 78\)](#)
- [Add Device Tokens or Registration IDs \(p. 80\)](#)
- [Create a Platform Endpoint and Manage Device Tokens \(p. 83\)](#)
- [Send a Direct Message to a Mobile Device \(p. 88\)](#)
- [Send Messages to Mobile Devices Subscribed to a Topic \(p. 88\)](#)
- [Send Custom Platform-Specific Payloads in Messages to Mobile Devices \(p. 88\)](#)

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, select 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. Go to <https://console.aws.amazon.com/sns/> and click **Create platform application**.
2. 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.

3. In the **Push notification platform** box, select 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.

For detailed instructions on how to acquire the following information, see [Getting Started with Amazon Device Messaging \(p. 39\)](#), [Getting Started with Apple Push Notification Service \(p. 44\)](#), [Getting Started with Baidu Cloud Push \(p. 50\)](#), [Getting Started with Google Cloud Messaging for Android \(p. 67\)](#), [Getting Started with MPNS \(p. 72\)](#), or [Getting Started with WNS \(p. 75\)](#).

Platform	Credentials
ADM	Client ID – Go to the Amazon Mobile App Distribution Portal , click Apps and Services , click the name of your Kindle Fire app, and then click Security Profile .
	Client Secret – Go to the Amazon Mobile App Distribution Portal , click Apps and Services , click the name of your Kindle Fire app, and then click Security Profile .
APNS	Certificate – Select the password encrypted certificate and private key, as exported from Keychain Access on your Mac computer in the .p12 file format.
	Certificate Password – Enter the password.
APNS_SANDBOX	Certificate – Same as previous for APNS.
	Certificate Password – Same as previous for APNS.
APNS_VOIP	Certificate – Same as previous for APNS.
	Certificate Password – Same as previous for APNS.
APNS_VOIP_SANDBOX	Certificate – Same as previous for APNS.
	Certificate Password – Same as previous for APNS.
MACOS	Certificate – Same as previous for APNS.
	Certificate Password – Same as previous for APNS.
MACOS_SANDBOX	Certificate – Same as previous for APNS.
	Certificate Password – Same as previous for APNS.

Platform	Credentials
Baidu	API Key – Enter the API key you received after creating a Baidu cloud push project, as described in Step 3: Create a Baidu Cloud Push Project (p. 56).
	Client Secret – Enter the secret key you received after creating a Baidu cloud push project, as described in Step 3: Create a Baidu Cloud Push Project (p. 56).
GCM	API Key – Go to the Google APIs Console web site , click API Access , and make note of the server API key with the Key for server apps (with IP locking) label. If you have not yet created a server API key, then click Create new Server key....
MPNS	Certificate – Enter the TLS certificate for your Windows developer account, as described in Step 4: Upload TLS Certificate (p. 73).
	Private Key – Enter the private key for the TLS certificate, as described in Step 4: Upload TLS Certificate (p. 73).
WNS	Client Secret – Enter the client secret, as described in How to authenticate with the Windows Push Notification Service (WNS) .
	Package Security Identifier (SID) – Enter the SID, as described in How to authenticate with the Windows Push Notification Service (WNS) .

- After you have entered this information, then click **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 Google Cloud Messaging for Android (GCM), 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. Go to <https://console.aws.amazon.com/sns/>, click **Apps**, click your app, and then click **Add Endpoints**.
2. In the **Endpoint Token** box, enter either the token ID or registration ID, depending on which notification service. For example, with ADM and GCM you enter the registration ID.
3. (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.
4. Finally, click **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--KMTlmMWx
wRkxMaDNST2luZz01,"User data with spaces requires quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWx
wRkxMaDNST2luZz04,"Data,with,commas,requires,quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWx
wRkxMaDNST2luZz02,"Quoted data requires ""escaped"" quotes"
amzn1.adm-registration.v1.XpvSSUk0Rc3hTVVV--TOKEN--KMTlmMWx
wRkxMaDNST2luZz03,"{"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
APBTKzPglCyT6E6oOfpdwLpcRNxQp5vCPFiFerU9oZylc22HvZSwQTDgmmw9WdNlXMerUPxm
pX0wl,"Different token style"
```

1. Go to <https://console.aws.amazon.com/sns/>, click **Apps**, click your app, and then click **Add Endpoints**.
2. Click **Migrate existing tokens over to AWS**, click **Choose File**, select your CSV file, and then click **Add Endpoints**.

To upload several tokens using the `CreatePlatformEndpoint` API

The following steps show how to use the sample Java app (`bulkupload` package) provided by AWS to upload several tokens (device tokens or registration IDs) to Amazon SNS. You can use this sample app to help you get started with uploading your existing tokens.

Note

The following steps use the Eclipse Java IDE. The steps assume you have installed the AWS SDK for Java and you have the AWS security credentials for your AWS account. For more information, see [AWS SDK for Java](#). For more information about credentials, see [How Do I Get Security Credentials?](#) in the *AWS General Reference*.

1. Download and unzip the [snsmobilepush.zip](#) file.
2. Create a new Java Project in Eclipse.
3. Import the `SNSSamples` folder to the top-level directory of the newly created Java Project. In Eclipse, right-click the name of the Java Project and then click **Import**, expand **General**, click **File System**, click **Next**, browse to the `SNSSamples` folder, click **OK**, and then click **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/GCM/gcmpushapp
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/GCM/gcmpushapp/165j2214-051z-3176-b586-138o3d420071
<2>[ERROR: MALFORMED CSV FILE] Null token found in /mytokendirectory/my
tokens.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 [Creating Temporary Security Credentials for Mobile Apps Using Identity Providers](#). 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 \(p. 37\)](#) and [Amazon SNS Mobile Push High Level Steps \(p. 38\)](#).

This section describes the recommended approach for creating a platform endpoint and managing device tokens.

Topics

- [Create a Platform Endpoint \(p. 83\)](#)
- [Pseudo Code \(p. 84\)](#)
- [Java Example \(p. 84\)](#)
- [Troubleshooting \(p. 87\)](#)

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.

Java Example

Here is an implementation of the previous pseudo code in Java:

```
class RegistrationExample {

    AmazonSNSClient client = new AmazonSNSClient(); //provide credentials here

    private 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();
        }

        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 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 token.*");
        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 \(p. 97\)](#).

Troubleshooting

Repeatedly Calling Create Platform Endpoint with an Outdated Device Token

Especially for GCM 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 GCM to update expired device tokens to new device tokens. GCM retains information on old device tokens for some time, but not indefinitely. Once GCM 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 limit to 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 GCM) 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 app and mobile device, by completing the following steps.

To send a direct message

1. Go to <https://console.aws.amazon.com/sns/>.
2. In the left **Navigation** pane, click **Apps** and click the app that you want to send a message to.
3. On the **Application Details** screen, select **Endpoint Actions** and then click **Publish**.
4. On the Publish dialog box, enter the message to appear in the app on the mobile device and then click **Publish**.

The notification message will then be sent from Amazon SNS to the platform notification service, which will then send the message to the app.

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.

To send to endpoints subscribed to a topic

1. Follow the steps as described in [Subscribe to a Topic \(p. 6\)](#). You just need to select **Application** in the **Protocol** drop-down menu and then enter the mobile endpoint Amazon Resource Name (ARN) in the **Endpoint** box.
2. Follow the steps to publish messages to a topic, as described in [Publish to a Topic \(p. 7\)](#), then all mobile endpoints that are subscribed to the topic will be sent the message.

Send Custom Platform-Specific Payloads in Messages to Mobile Devices

You can use either the Amazon SNS console or APIs to send custom platform-specific payloads in messages to mobile devices. The following sections describe how to use the Amazon SNS console to create and send custom platform-specific payloads for each of the supported notification services. For information on using the Amazon SNS APIs, see [Using Amazon SNS Mobile Push APIs \(p. 97\)](#) and the AWS sample file `SNSMobilePush.java`, which is included in the `snsmobilepush.zip` file.

JSON Formatted Message Data

When sending platform-specific payloads in messages using the Amazon SNS console, the data must be key-value pair strings and formatted as JSON with quotation marks escaped. The following example, including formatting and spaces for readability, shows a sample custom message for the GCM platform with key-value pair within the message body and formatted as JSON.

```
{
  "GCM": "{
    "data": {
      "message": "Check out these awesome deals!",
      "url": "www.amazon.com"
    }
  }"
```

When sending messages using the console quotation marks must be escaped (\), as the following example shows.

```
{
  "GCM": "{
    \"data\": {
      \"message\": \"Check out these awesome deals!\",
      \"url\": \"www.amazon.com\"
    }
  }"
```

When entered in the Amazon SNS console, the example should look similar to the following:

```
{
"GCM": "{ \"data\": { \"message\": \"Check out these awesome
deals!\", \"url\": \"www.amazon.com\" } }"
```

Platform-Specific Key-Value Pairs

In addition to sending custom data as key-value pairs, you can also send platform-specific key-value pairs within the JSON payload. For example, if you wanted to include *time_to_live* and *collapse_key* GCM parameters after the custom data key-value pairs included in the *data* GCM parameter, then the JSON payload without escaped quotation marks would look similar to the following:

```
{
  "GCM": "{
    "data": {
      "message": "Check out these awesome deals!",
      "url": "www.amazon.com"
    },
    "time_to_live": 3600,
    "collapse_key": "deals"
  }"
```

When entered in the Amazon SNS console, the example should look similar to the following:

```
{
  "GCM": "{ \"data\": { \"message\": \"Check out these awesome
deals!\", \"url\": \"www.amazon.com\" }, \"time_to_live\": 3600, \"col
lapse_key\": \"deals\" }"
}
```

For a list of the supported key-value pairs in each of the push notification services supported in Amazon SNS, see the following links:

- APNS – [Apple Push Notification Service](#)
- GCM – [Implementing GCM Server Message Parameters](#)
- ADM – [Sending a Message](#)

Messages to an App on Multiple Platforms

To send a message to an app installed on devices for multiple platforms, such as GCM 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, GCM, 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\" } }"
}
```

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 with the AWS Management Console \(p. 91\)](#)
- [Amazon SNS Message Delivery Status CloudWatch Log Examples \(p. 91\)](#)
- [Configuring Message Delivery Status Attributes with the AWS SDKs \(p. 92\)](#)
- [Platform Response Codes \(p. 93\)](#)

Configuring Message Delivery Status Attributes with the AWS Management Console

You can configure message delivery status attributes with the AWS Management Console.

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left **Navigation** pane, click **Apps**, and then click the app containing the endpoints for which you want receive CloudWatch Logs.
3. Click **Application Actions** and then click **Delivery Status**.
4. On the **Delivery Status** dialog box, click **Create IAM Roles**.

You will then be redirected to the IAM console.

5. Click **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, click **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": "ExampleI7fFackhJ1xj1qT64RaBkcGHochmf1VQAr9k-IB
JtKjp7fedYPzEwT_Pq3Tu0lroqrolcwWJUvgkcPPYcaXCpPwmG3Bqn-
wiqIEzp5zZ7y_jsM0PKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw",
      "attempts": 1,
      "providerResponse": "{\"multicast_id\":5138139752481671853,\"suc
cess\":1,\"failure\":0,\"canonical_ids\":0,\"results\":[{\\"mes
sage_id\":\\"0:1422313659698010%d6ba8edff9fd7ecd\\\"}]}",
      "destination": "arn:aws:sns:us-east-1:111122223333:endpoint/GCM/GCMPush
App/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": "ex
ample29z6j5c4df46f809505189c4c83fjcgf7f6257e98542d2jt3395kj73",
    "attempts": 1,
    "providerResponse": "NotificationErrorResponse(command=8, status=Inval
idToken, id=1, cause=null)",
    "destination": "arn:aws:sns:us-east-1:111122223333:endpoint/APNS_SAND
BOX/APNSPushApp/986cb8a1-4f6b-34b1-9a1b-d9e9cb553944"
  }
}
```

For a list of push notification service response codes, see [the section called “Platform Response Codes” \(p. 93\)](#).

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.setPlatformApplication
Arn("arn:aws:sns:us-west-2:111122223333:app/GCM/GCMPushApp");
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 Sending a Message via Amazon Device Messaging
Apple Push Notification Service (APNS)	See Codes in error-response packet in the iOS Developer Library
Google Cloud Messaging for Android (GCM)	See Downstream message error response codes in the GCM Connection Server Reference
Microsoft Push Notification Service for Windows Phone (MPNS)	See Push Notification Service response codes
Windows Push Notification Services (WNS)	See "Response codes" in Push notification service request and response headers

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), Google Cloud Messaging for Android (GCM), 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 \(p. 93\)](#)
- [How to Set Application Event Notifications \(p. 94\)](#)

Available Application Events

Application event notifications track when individual platform endpoints are created, deleted, and updated, along with delivery failures. The attribute name for each application event is as follows:

Attribute name	Description
EventEndpointCreated	A notification is triggered when a new platform endpoint is added to your application.
EventEndpointDeleted	A notification is triggered when any of the platform endpoints associated with your application is deleted.
EventEndpointUpdated	A notification is triggered when any of the attributes of the platform endpoints associated with your application are changed.
EventDeliveryFailure	A notification is triggered when a delivery to any of the platform endpoints associated with your application encounters a permanent failure. 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 .

Each of the preceding attributes can be associated with an application. The application can then receive these event notifications.

How to Set Application Event Notifications

You can set 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 AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the service navigation pane, choose **Applications**.
3. Choose the name of the application that you want to set event notifications for.
4. Choose **Actions, Configure events**.
5. For each of the events that you want to send events notifications for, type the corresponding Amazon SNS topic ARN.
6. Choose **Save configuration**. The event notifications are set.

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/GCM/MyGCMPatformApplication
--attributes EventEndpointCreated="arn:aws:sns:us-east-1:12345EXAMPLE:MyGCMPatformApplicationEvents",
EventEndpointDeleted="arn:aws:sns:us-east-1:12345EXAMPLE:MyGCMPatformApplicationEvents",
EventEndpointUpdated="arn:aws:sns:us-east-1:12345EXAMPLE:MyGCMPatformApplica
```

```
tionEvents",  
EventDeliveryFailure="arn:aws:sns:us-east-1:12345EXAMPLE:MyGCMPatformApplica  
tionEvents"
```

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 Google Cloud Messaging for Android (GCM).

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 GCM, 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 \(p. 96\)](#)
- [Precedence Order for Determining TTL \(p. 96\)](#)
- [Specifying TTL with the AWS Management Console \(p. 97\)](#)
- [Specifying TTL with the AWS SDKs \(p. 97\)](#)

TTL Message Attributes for Push Notification Services

The following is a list of the TTL message attributes for push notification services that you can use to set when using the AWS SDKs or query API:

Push Notification Service	TTL Message Attribute
Amazon Device Messaging (ADM)	AWS.SNS.MOBILE.ADM.TTL
Apple Push Notification Service (APNS)	AWS.SNS.MOBILE.APNS.TTL
Apple Push Notification Service Sandbox (APNS_SANDBOX)	AWS.SNS.MOBILE.APNS_SANDBOX.TTL
Baidu Cloud Push (Baidu)	AWS.SNS.MOBILE.BAIDU.TTL
Google Cloud Messaging for Android (GCM)	AWS.SNS.MOBILE.GCM.TTL
Windows Push Notification Services (WNS)	AWS.SNS.MOBILE.WNS.TTL

Each of the push notification services handle TTL differently. Amazon SNS provides an abstract view of TTL over all the push notification services, which makes it easier to specify TTL. When you use the AWS Management Console to specify TTL (in seconds), you only have to enter the TTL value once and Amazon SNS will then calculate the TTL for each of the selected push notification services when publishing the message.

TTL is relative to the publish time. Before handing off a push notification message to a specific push notification service, Amazon SNS computes the dwell time (the time between the publish timestamp and just before handing off to a push notification service) for the push notification and passes the remaining TTL to the specific push notification service. If TTL is shorter than the dwell time, Amazon SNS won't attempt to publish.

If you specify a TTL for a push notification message, then the TTL value must be a positive integer, unless the value of 0 has a specific meaning for the push notification service—such as with APNS and GCM. 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 [expiration date](#). For more information about the TTL parameter set to 0 when using GCM, see [Lifetime of a message](#).

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 with the AWS Management Console

You can specify TTL with the AWS Management Console.

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left **Navigation** pane, click **Apps**, and then click the app containing the endpoints you want to set TTL for when publishing a message.
3. Select the endpoints to publish a message to, click **Endpoint Actions** and then click **Publish**.
4. On the **Publish** dialog box, enter the number of seconds for Time to Live (TTL) and then click **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 MessageAttribute
Value().withDataType("String").withStringValue("86400"));

PublishRequest publishRequest = new PublishRequest();
publishRequest.setMessageAttributes(messageAttributes);
String message = "{\"title\":\"Test_Title\",\"description\":\"Test_Descrip
tion\"}";
publishRequest.setMessage(message);
publishRequest.setMessageStructure("json");
publishRequest.setTargetArn("arn:aws:sns:us-east-1:999999999999:end
point/BAIDU/TestApp/318fc7b3-bc53-3d63-ac42-e359468ac730");
PublishResult publishResult = snsClient.publish(publishRequest);
```

For more information about using message attributes with Amazon SNS, see [Using Amazon SNS Message Attributes](#) (p. 164).

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 Google Cloud Messaging for Android (GCM). For more information about the prerequisites, see [Prerequisites](#) (p. 37).

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 [Amazon SNS Mobile Push High Level Steps \(p. 38\)](#).

The following is a list and description of the Amazon SNS mobile push APIs:

API	Description
<code>CreatePlatformApplication</code>	Creates a platform application object for one of the supported push notification services, such as APNS and GCM, to which devices and mobile apps may register. Returns a <i>PlatformApplicationArn</i> attribute, which is used by the <code>CreatePlatformEndpoint</code> action. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.
<code>SetPlatformApplicationAttributes</code>	Sets the attributes of the platform application object. For more information, see SetPlatformApplicationAttributes in the Amazon Simple Notification Service API Reference.
<code>GetPlatformApplicationAttributes</code>	Retrieves the attributes of the platform application object. For more information, see GetPlatformApplicationAttributes in the Amazon Simple Notification Service API Reference.
<code>ListPlatformApplications</code>	Lists the platform application objects for the supported push notification services. For more information, see ListPlatformApplications in the Amazon Simple Notification Service API Reference.
<code>DeletePlatformApplication</code>	Deletes a platform application object. For more information, see DeletePlatformApplication in the Amazon Simple Notification Service API Reference.
<code>CreatePlatformEndpoint</code>	Creates an endpoint for a device and mobile app on one of the supported push notification services. <code>CreatePlatformEndpoint</code> uses the <i>PlatformApplicationArn</i> attribute returned from the <code>CreatePlatformApplication</code> action. The <i>EndpointArn</i> attribute, which is returned when using <code>CreatePlatformEndpoint</code> , is then used with the <code>Publish</code> action to send a notification message to a mobile app and device. For more information, see CreatePlatformEndpoint in the Amazon Simple Notification Service API Reference.
<code>SetEndpointAttributes</code>	Sets the attributes for an endpoint for a device and mobile app. For more information, see SetEndpointAttributes in the Amazon Simple Notification Service API Reference.
<code>GetEndpointAttributes</code>	Retrieves the endpoint attributes for a device and mobile app. For more information, see GetEndpointAttributes in the Amazon Simple Notification Service API Reference.

API	Description
ListEndpointsByPlatformApplication	Lists the endpoints and endpoint attributes for devices and mobile apps in a supported push notification service. For more information, see ListEndpointsByPlatformApplication in the Amazon Simple Notification Service API Reference.
DeleteEndpoint	Deletes the endpoint for a device and mobile app on one of the supported push notification services. For more information, see DeleteEndpoint in the Amazon Simple Notification Service API Reference.

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 \(p. 97\)](#).

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

Amazon Simple Notification Service Developer Guide
API Errors

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	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 in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
APNS — Credential are not in a valid .pem format	The APNS credential, which is the private key, is not in a valid .pem format.	400	CreatePlatformApplication
GCM — serverAPIKey is not provided	The GCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	CreatePlatformApplication
GCM — serverAPIKey is empty	The GCM credential, which is the API key, is empty.	400	CreatePlatformApplication
GCM — serverAPIKey is a null string	The GCM credential, which is the API key, is null.	400	CreatePlatformApplication
GCM — serverAPIKey is invalid	The GCM 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

Amazon Simple Notification Service Developer Guide
API Errors

Error	Description	HTTPS Status Code	Action that Returns this Error
ADM — client_secret is not valid	The required string for the client secret is not valid.	400	CreatePlatformApplication
ADM — client_id is empty string	The required string for the client ID is empty.	400	CreatePlatformApplication
ADM — clientId is not provided	The required string for the client ID is not provided.	400	CreatePlatformApplication
ADM — clientid is a null string	The required string for the client ID is null.	400	CreatePlatformApplication
ADM — client_id is not valid	The required string for the client ID is not valid.	400	CreatePlatformApplication
EventEndpointCreated has invalid ARN format	EventEndpointCreated has invalid ARN format.	400	CreatePlatformApplication
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	CreatePlatformApplication
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryAttempt-Failure has invalid ARN format	EventDeliveryAttempt-Failure 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
EventDeliveryAttempt-Failure is not an existing Topic	EventDeliveryAttempt-Failure 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

Error	Description	HTTPS Status Code	Action that Returns this Error
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
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
GCM — serverAPIKey is not provided	The GCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	SetPlatformAttributes

Amazon Simple Notification Service Developer Guide
API Errors

Error	Description	HTTPS Status Code	Action that Returns this Error
GCM — serverAPIKey is a null string	The GCM 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
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	SetPlatformAttributes
EventDeliveryAttempt-Failure has invalid ARN format	EventDeliveryAttempt-Failure 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
EventDeliveryAttempt-Failure is not an existing Topic	EventDeliveryAttempt-Failure 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

Amazon Simple Notification Service Developer Guide
API Errors

Error	Description	HTTPS Status Code	Action that Returns this Error
Platform ARN is invalid	The platform ARN is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	ListEndpointsByPlatformApplication
Token specified is invalid	The specified token is invalid.	400	ListEndpointsByPlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	DeletePlatformApplication
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	DeletePlatformApplication
Platform ARN is invalid	The platform ARN is invalid.	400	CreatePlatformEndpoint
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	CreatePlatformEndpoint
Token is not specified	The token is not specified.	400	CreatePlatformEndpoint
Token is not of correct length	The token is not the correct length.	400	CreatePlatformEndpoint
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	CreatePlatformEndpoint
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

Amazon Simple Notification Service Developer Guide
API Errors

Error	Description	HTTPS Status Code	Action that Returns this Error
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

Sending Amazon SNS Messages to Amazon SQS Queues

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. By 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-ae9f9-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" : "EXAMPLEnTrFPa37tnVO0FF9Iau3MGzjlJLRfySEoWz4uZHSj6ycK4ph71Zm
dv0NtJ4dC/E19FOGp3VuvchpaTraNHWhhQ/OsN1HVz20zxmF9b88R8GtqjfkB5woZZmz87HiM6CY
DTo3l7LMwFT4VU7ELtyaBBafhPTg9O5CnKkg=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&Sub
scriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-
db410a0f2bee"
}
```


Note

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. 107)
2. [Give `sqs:SendMessage` permission to the Amazon SNS topic so that it can send messages to the queue.](#) (p. 108)
3. [Subscribe the queue to the Amazon SNS topic.](#) (p. 109)
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. 109)
5. [Test it out by publishing a message to the topic and reading the message from the queue.](#) (p. 111)

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. 112).

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. 115).

Step 1. Get the ARN of the queue and the 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 AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the navigation pane, select the topic whose ARN you want to get.
3. From the **Topic Details** pane, 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](#) in the *Amazon Simple Notification Service Getting Started Guide*. For more information, see [Creating a Queue](#) in the *Amazon Simple Queue Service Getting Started 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, click the **Permissions** tab, and then click **Add a Permission**.
3. In the **Add a Permission** dialog box, select **Allow** for **Effect**, select **Everybody (*)** for **Principal**, and then select **SendMessage** from the **Actions** drop-down.
4. Add a condition that allows the action for the topic. Click **Add Conditions (optional)**, select **ArnEquals** for **Condition**, select **aws:SourceArn** for **Key**, and paste in the topic ARN for **Value**. Click **Add Condition**. The new condition should appear at the bottom of the box (you may have to scroll down to see this).
5. Click **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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MySQSPolicy001",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:123456789012:MyQueue",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012: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](#) command, or the [Subscribe](#) API action. Before you start, make sure you have the ARN for the queue that you want to subscribe.

To subscribe a queue to a topic using the Amazon SNS console

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the navigation pane, select the topic.
3. Click **Create Subscription**, select **Amazon SQS** for **Protocol**, paste in the ARN for the queue that you want the topic to send messages to for **Endpoint**, and click **Subscribe**.
4. For the **Subscription request received!** message, click **Close**.

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. 112).

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 [Controlling User Access to Your AWS Account](#) in the *Amazon Simple Notification Service Getting Started Guide* and [Controlling User Access to Your AWS Account](#) in the *Amazon SQS Developer Guide*.

There are two ways to control access to a topic or queue:

- [Add a policy to an IAM user or group](#) (p. 110). 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. 110). 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`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowPublishToMyTopic",
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:123456789012: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`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowReadDeleteMessageOnMyQueue",
    "Effect": "Allow",
    "Action": [
      "sqs:ReceiveMessage",
      "sqs>DeleteMessage"
    ],
    "Resource": [
      "arn:aws:sns:us-east-1:123456789012:MyQueue1",
      "arn:aws:sns:us-east-1:123456789012: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 `11122223333` permission to perform the `sns:Publish` action on that topic.

```
{
  "Version": "2012-10-17",
```

```
"Id": "MyTopicPolicy",
"Statement": [{
  "Sid": "Allow-publish-to-topic",
  "Effect": "Allow",
  "Principal": {
    "AWS": "111122223333"
  },
  "Action": "sns:Publish",
  "Resource": "arn:aws:sns:us-east-1:123456789012: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.

```
{
  "Version": "2012-10-17",
  "Id": "MyQueuePolicy",
  "Statement": [
    {
      "Sid": "Allow-Processing-Of-Messages-for-Queue",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": [
        "sqs>DeleteMessage",
        "sqs:ReceiveMessage"
      ],
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:MyQueue",
      ]
    }
  ]
}
```

Step 5. Test it

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. In the navigation pane, select the topic and click **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 click **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, select **View/Delete Messages** and click **Start Polling for Messages**. A message with a type of **Notification** appears.
4. In the **Body** column, click **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" : "EXAMPLEnTrFPa37tnVO0FF9Iau3MGzj1JLRfySEoWz4uZHSj6ycK4ph71Zm
dv0NtJ4dC/E19FOGp3VuvchpaTraNHWhhQ/OsN1HVz20zxmF9b88R8GtqjfkB5woZZmz87HiM6CY
DTo3l7LMwFT4VU7ELtyaBBafhPTg9O5CnKkg=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotification
Service-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsub
scribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-
ab0e-4ec2-88e0-db410a0f2bee"
}
```

5. Click **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 [Sending Amazon SNS Messages to Amazon SQS Queues \(p. 106\)](#)). 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 \(p. 112\)](#)
- [User Who Does Not Own the Queue Creates Subscription \(p. 114\)](#)

Queue Owner Creates Subscription

When the queue owner creates the subscription, the subscription does not require confirmation. The queue starts receiving notifications from the topic as soon as the `Subscribe` action completes. To enable the queue owner to 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. When added to the topic `MyTopic` in the

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

```
{
  "Version": "2012-10-17",
  "Id": "MyTopicSubscribePolicy",
  "Statement": [{
    "Sid": "Allow-other-account-to-subscribe-to-topic",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": "sns:Subscribe",
    "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"
  }]
}
```

After this policy has been set on `MyTopic`, a user can log in to the Amazon SNS console with credentials for account 111122223333 to subscribe to the topic.

To add an Amazon SQS queue subscription to a topic in another account using the Amazon SQS console

1. Using the credentials of the AWS account containing the queue or an IAM user in that account, sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Make sure you have the ARNs for both the topic and the queue. You will need them when you create the subscription.
3. Make sure you have set `sqs:SendMessage` permission on the queue so that it can receive messages from the topic. For more information, see [Step 2. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue \(p. 108\)](#).
4. In the navigation pane, select the **SNS Dashboard**.
5. In the **Dashboard**, in the **Additional Actions** section, click **Create New Subscription**.
6. In the **Topic ARN** box, enter the ARN for the topic.
7. For **Protocol**, select **Amazon SQS**.
8. In the **Endpoint** box, enter the ARN for the queue.
9. Click **Subscribe**.
10. For the **Subscription request received!** message, you'll notice text that says you must confirm the subscription. Because you are the queue owner, the subscription does not need to be confirmed. Click **Close**. You've completed the subscription process and notification messages published to the topic can now be sent to the queue.

The user can also use the access key and secret key for the AWS account 111122223333 to issue the `sns-subscribe` command or call the [Subscribe](#) API action to subscribe an Amazon SQS queue to `MyTopic` in the account 123456789012. The following `sns-subscribe` command subscribes the queue `MyQ` from account 111122223333 to the topic `MyTopic` in account 123456789012.

```
sns-subscribe arn:aws:sns:us-east-1:123456789012:MyTopic --protocol sqs --end
point arn:aws:sqs:us-east-1:111122223333:MyQ
```

User Who Does Not Own the Queue Creates Subscription

When a user who is not the queue owner creates the subscription (for example, when the topic owner in account A adds a subscription for a queue in account B), the subscription must be confirmed.

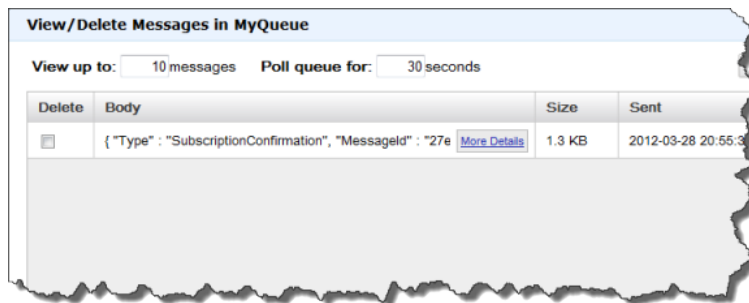
Important

Before you subscribe to the topic, make sure you have set `sqs:SendMessage` permission on the queue so that it can receive messages from the topic. See [Step 2. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue](#) (p. 108).

When the user calls the `Subscribe` action, a message of type `SubscriptionConfirmation` is sent to the queue and the subscription is displayed in the Amazon SNS console with its Subscription ID set to **Pending Confirmation**. To confirm the subscription, a user who can read messages from the queue must visit the URL specified in the `SubscribeURL` value in the message. Until the subscription is confirmed, no notifications published to the topic are sent to the queue. To confirm a subscription, you can use the Amazon SQS console or the [ReceiveMessage](#) API action.

To confirm a subscription using the Amazon SQS console

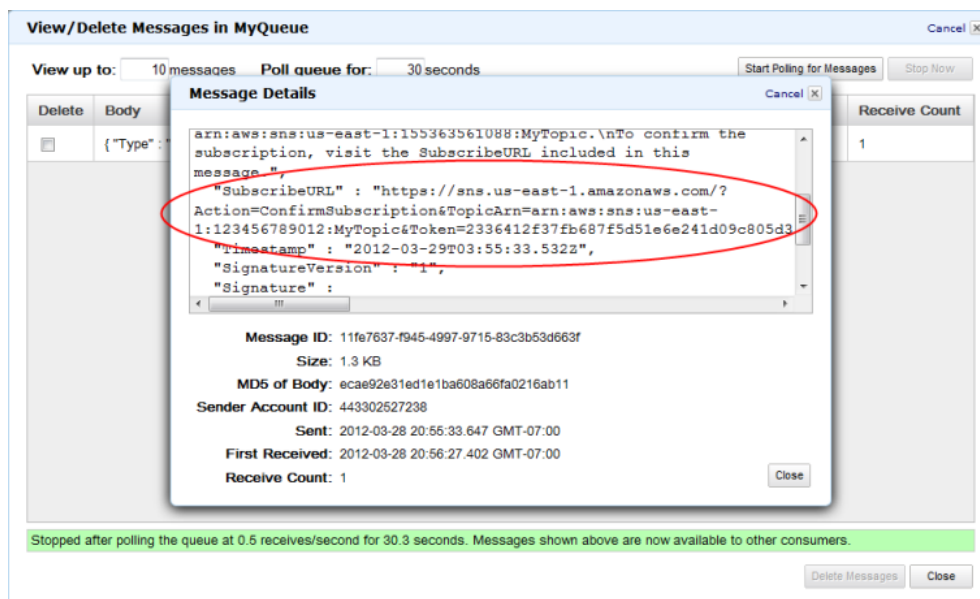
1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Select the queue that has a pending subscription to the topic.
3. From the **Queue Action** drop-down, select **View/Delete Messages** and click **Start Polling for Messages**. A message with a type of **SubscriptionConfirmation** appears.
4. In the **Body** column, click **More Details**.



5. In the text box, find the **SubscribeURL** value and copy the URL. It will look similar to the following URL.

```
https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic-231367516218651488710116167756216515045157a390019148367024344E270E90D27832084588292720000
```


Amazon Simple Notification Service Developer Guide Using an AWS CloudFormation Template to Create a Topic that Sends Messages to Amazon SQS Queues



- In a web browser, paste the URL into the address bar to visit the URL. You will see a response similar to the following XML document.

```
<ConfirmSubscriptionResponse xmlns="http://sns.amazonaws.com/doc/2010-03-31/">
  <ConfirmSubscriptionResult>
    <SubscriptionArn>arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-
ab0e-4ec2-88e0-db410a0f2bee</SubscriptionArn>
  </ConfirmSubscriptionResult>
  <ResponseMetadata>
    <RequestId>dd266ecc-7955-11e1-b925-5140d02da9af</RequestId>
  </ResponseMetadata>
</ConfirmSubscriptionResponse>
```

If you view the topic subscription in the Amazon SNS console, you will now see that subscription ARN replaces the **Pending Confirmation** message in the **Subscription ID** column. The subscribed queue is ready to receive messages from the topic.

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. Note that 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",
```

Amazon Simple Notification Service Developer Guide
Using an AWS CloudFormation Template to Set Up
Topics and Queues Within an AWS Account

```
    "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": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "sns:Publish"
                ],
                "Resource": { "Ref": "MySNSTopic" }
              }
            ]
          }
        }
      ]
    }
  }
}
```

Amazon Simple Notification Service Developer Guide
Using an AWS CloudFormation Template to Set Up
Topics and Queues Within an AWS Account

```
    }
  ]}
}
]
},
"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":{
          "Version": "2012-10-17",
          "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"}]
  }
}
```

Amazon Simple Notification Service Developer Guide
Using an AWS CloudFormation Template to Set Up
Topics and Queues Within an AWS Account

```
    },
    "MyQueuePolicy": {
      "Type": "AWS::SQS::QueuePolicy",
      "Properties": {
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Id": "MyQueuePolicy",
          "Statement": [
            {
              "Sid": "Allow-SendMessage-To-Both-Queues-From-SNS-Topic",
              "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" ] },

```

Amazon Simple Notification Service Developer Guide
Using an AWS CloudFormation Template to Set Up
Topics and Queues Within an AWS Account

```
        "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"]}
    ]
  ]}
}
}
```

Sending and Receiving SMS Notifications Using Amazon SNS

You can use [Amazon SNS](#) to send and receive Short Message Service (SMS) notifications to SMS-enabled mobile phones and smart phones.

To send an SMS message using Amazon SNS, select one of your Amazon SNS topics that has a display name and publish a message to the topic. The topic must have a display name assigned to it because the first ten (10) characters of the display name are used as the initial part of the text message prefix. SMS messages can contain up to 140 ASCII characters or 70 Unicode characters. Because Amazon SNS includes a display name prefix with all SMS messages that you send, the sum of the display name prefix and the message payload cannot exceed 140 ASCII characters or 70 Unicode characters. Amazon SNS truncates messages that exceed these limits.

To receive SMS messages using Amazon SNS, select the SMS protocol setting when you subscribe to an Amazon SNS topic. The full message prefix comprises the display name followed by the > character. For example, if the display name of a topic is `MyTopic` and the message payload sent is `Hello World!`, the message delivered would appear as it does in the following example:

```
MYTOPIC>Hello World!
```

Note

Display names are not case sensitive, and Amazon SNS converts display names to uppercase characters for SMS messages.

You can use SMS notifications in conjunction with other notification types, such as email. For example, if you use CloudWatch to monitor your AWS application, you can create a CloudWatch alarm that is associated with an Amazon SNS topic. You can then subscribe to the topic via both email and SMS so that you receive notifications not only through email, but also on your SMS-enabled device.

To facilitate the use of a single message for both SMS and email notifications, Amazon SNS checks whether your message contains both a message body and a subject. If you publish a message that contains only a message body, both SMS and email subscribers receive the same message, up to the size limits for each protocol (140 characters for SMS and 256 KB for email). If your message is longer than 140 characters, your SMS message will be truncated.

To avoid a truncated SMS message when your message payload is longer than 140 characters, publish a message with both a subject and a message payload. For messages with both a subject and a message

payload, Amazon SNS sends only the subject to SMS subscribers, but sends both the subject and the message to any email subscribers. This allows you to send email notifications up to 256 KB long and also have the subject line delivered as an SMS message to your mobile device.

Note

SMS notifications are currently supported for phone numbers in the United States. SMS messages can be sent only from topics created in the US East (N. Virginia) region. However, you can publish messages to topics that you create in the US East (N. Virginia) region from any other region.

Amazon SNS uses short code 30304 to send and receive SMS messages.

Prerequisites

- **Sign up for Amazon SNS**—Create an AWS account if you don't have one.
For more information, see [Before You Begin \(p. 5\)](#).
- **Create an Amazon SNS topic**—Create an Amazon SNS topic if you don't have one.
For more information, see [Create a Topic \(p. 6\)](#).

After you have completed both of the prerequisite tasks, you can use the following process to publish and receive SMS messages with Amazon SNS.

Process for Sending and Receiving SMS Messages with Amazon SNS

Task 1: Assign a Topic Display Name (p. 122)
Task 2: Subscribe to a Topic Using the SMS Protocol (p. 123)
Task 3: Publish a Message (p. 124)
Task 4: Cancel SMS Subscriptions (p. 125)

Task 1: Assign a Topic Display Name

To publish SMS messages for a topic, you must assign the topic a display name.

To assign a display name to a topic

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic.
3. Click the **Other actions** drop-down list and then select **Edit topic display name**.
4. In the **Display Name** box, type a display name and click **Set display name**.

The new topic display name appears in the **Topic Details** page.

Task 2: Subscribe to a Topic Using the SMS Protocol

Note

SMS notifications are currently supported for phone numbers in the United States. SMS messages can be sent only from topics created in the US East (N. Virginia) region. However, you can publish messages to topics that you create in the US East (N. Virginia) region from any other region.

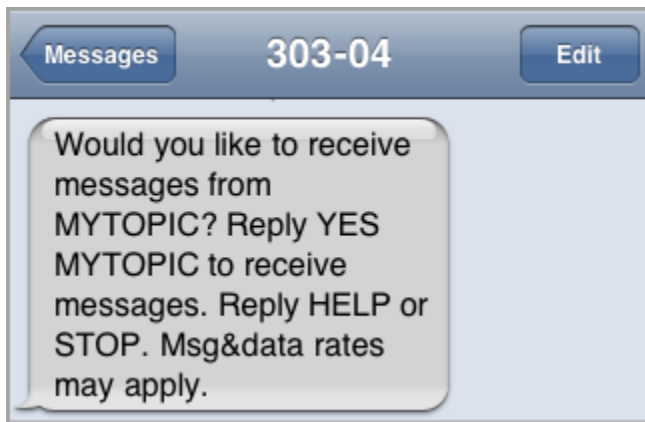
To subscribe to an Amazon SNS topic using the SMS protocol

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select the topic.
3. Click the **Other actions** drop-down list and then select **Subscribe to topic**.
4. In the **Protocol** drop-down list, select **SMS**.
5. In the **Endpoint** box, type the phone number of an SMS-enabled device and then click **Create subscription**.

Note

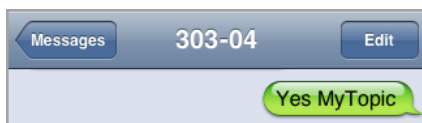
Use numbers only. Do not include dashes, spaces, or parentheses.

Amazon SNS sends a confirmation text message to the SMS-enabled device associated with the number you entered.

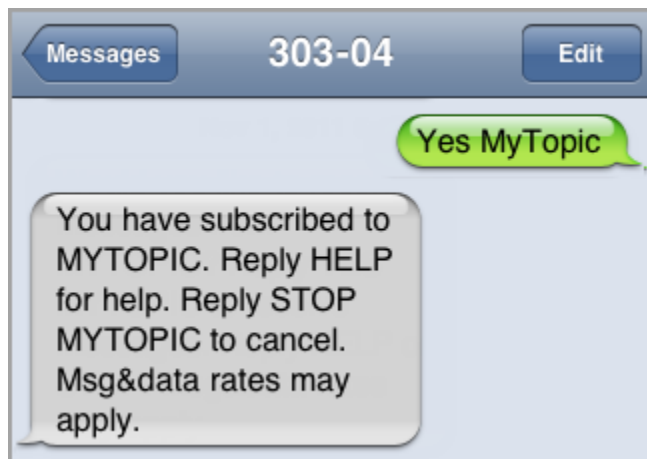


In the Amazon SNS console, the subscription is listed as **PendingConfirmation** until the SMS-enabled device confirms the subscription.

6. Use the SMS-enabled device associated with the phone number you entered in the previous step to reply affirmatively to the confirmation text message. For example, the following text message confirms a subscription to the `MyTopic` Amazon SNS topic.



Amazon SNS responds with a subscription confirmation message.



Task 3: Publish a Message

To publish a message to a topic

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select the topic you want to publish to.
3. Click the **Publish to topic** button.
4. In the **Subject** box, type a subject if you want to use the **Message** box for messages to email subscribers.

If you include text in the **Subject** box, the SMS message will contain the subject text rather than the text from the **Message** box. Any email subscribers, however, will receive both the subject and the message body. This allows you to use a single published message to send a short SMS message using the subject and a longer email message using the message payload.

5. In the **Message** box, type a message.

Amazon SNS sends text that you enter in the **Message** box to SMS subscribers unless you also enter text into the **Subject** box.

6. Click **Publish message**.

Amazon SNS displays a confirmation dialog box.

The SMS message appears on your SMS-enabled device.

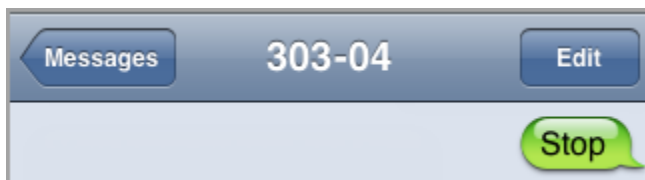


Task 4: Cancel SMS Subscriptions

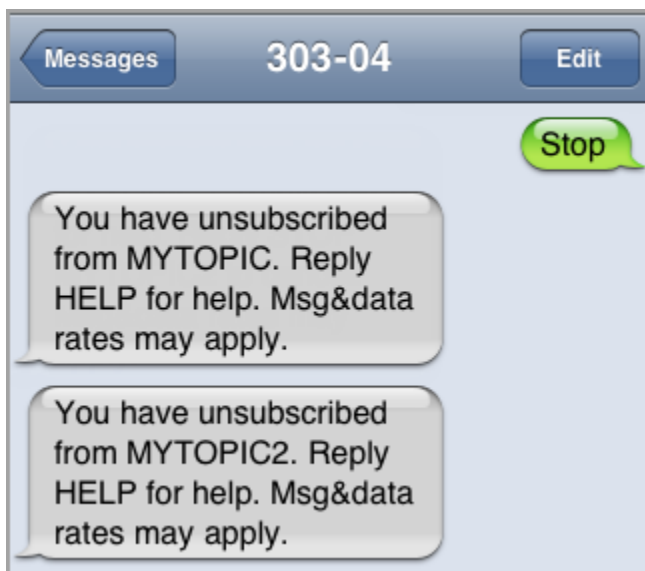
You have several options for canceling SMS subscriptions to a topic. You can stop receiving all SMS messages by replying `STOP` or `QUIT` to short code 30304. To cancel your subscription to a specific topic, send an SMS message that contains `STOP <TOPICNAME>` to short code 30304, where `<TOPICNAME>` is the display name of the topic. You can also cancel a subscription through the AWS Management Console or the Query API [Unsubscribe](#) action.

To stop receiving all SMS messages from Amazon SNS

- Use your SMS-enabled device to send a `STOP` or `QUIT` message to short code 30304. For example, the following text message cancels both of the subscriptions that this device had with Amazon SNS.

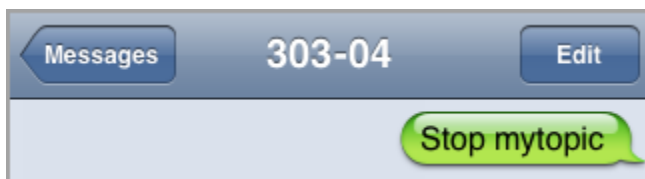


Amazon SNS responds with confirmation messages for each topic.

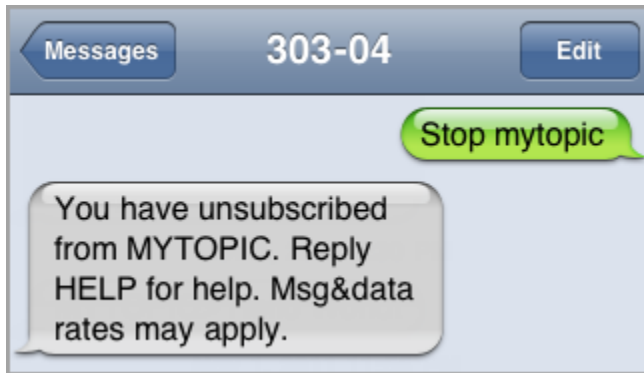


To stop receiving SMS messages from a specific topic

- Use your SMS-enabled device to send an SMS message that contains `STOP <TOPICNAME>` to short code 30304, where `<TOPICNAME>` is the display name of the topic. For example, the following SMS message cancels a subscription to a topic named `mytopic`.



Amazon SNS responds with a confirmation message.



Sending Amazon SNS Messages to HTTP/HTTPS Endpoints

You can use [Amazon SNS](#) to send notification messages to one or more HTTP or HTTPS endpoints. When you subscribe an endpoint to a topic, you can publish a notification to the topic and Amazon SNS sends an HTTP POST request delivering the contents of the notification to the subscribed endpoint. When you subscribe the endpoint, you select 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 http://en.wikipedia.org/wiki/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 <http://www.rfc-editor.org/info/rfc2617>

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 \(p. 175\)](#) and [HTTP/HTTPS Notification JSON Format \(p. 178\)](#).

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: da41e39f-ea4d-435a-b922-c6aae3915ebe
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 761
Content-Type: text/plain; charset=UTF-8
Host: ec2-50-17-44-49.compute-1.amazonaws.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
```

Amazon Simple Notification Service Developer Guide
Step 1: Make sure your endpoint is ready to process
Amazon SNS messages

```
{
  "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" : "EXAMPLElDMXvB8r9R83tGoNn0ecwd5Uj11zsvS
vbItzfaMpN2nk5HVSsw7XnOn/49IkxDKz8Yr1H2qJXj2iZB0Zo2071c4qQk1fMUDI3LG
pij7RCW7AW9vYYsSqIKRnFS94ilu7NFhUzLiiEYr4BKHpdTmdD6c0esKEYBpabxDSc=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&Sub
scriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-
fcfcc21c8f55"
}
```

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

[Step 1: Make sure your endpoint is ready to process Amazon SNS messages \(p. 128\)](#)

[Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic \(p. 131\)](#)

[Step 3: Confirm the subscription \(p. 132\)](#)

[Step 4: Set the delivery retry policy for the subscription \(optional\) \(p. 132\)](#)

[Step 5: Give users permissions to publish to the topic \(optional\) \(p. 132\)](#)

[Step 6: Send messages to the HTTP/HTTPS endpoint \(p. 133\)](#)

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.

Amazon Simple Notification Service Developer Guide
Step 1: Make sure your endpoint is ready to process
Amazon SNS messages

For details about the HTTP header, see [HTTP/HTTPS Headers \(p. 175\)](#). 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" :
  "2336412f371687f5f1e6241d09c805551b0712794c5f6986692768b6574ba6f3ab71854256ad0242b0cccc29417f1f0260c582ef
  bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "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 Sub
  scribeURL included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSub
  scription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTop
  ic&Token=2336412f371687f5f1e6241d09c805551b0712794c5f6986692768b6574ba6f3ab71854256ad0242b0cccc29417f1f0260c582ef
  bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH+DcEwjAPg809mY8dReBSwksfg2S7WKQcicNK
  WLQjwu6A4VbeS0QHVCkRS7fUQvi2egU3N858fiTDN6bkkOxYDvrY0Ad8L10Hs3zH81mtnPk5uvvol
  IC1CXGu43obcgFxeL3khZl8IKvO61GWB6jI9b5+gLPoBc1Q=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotification
  Service-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 (<http://wiki.fasterxml.com/JacksonHome>) 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 \(p. 153\)](#).

Amazon Simple Notification Service Developer Guide
Step 1: Make sure your endpoint is ready to process
Amazon SNS messages

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 \(p. 176\)](#). 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 \(p. 175\)](#). 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!",
```


Amazon Simple Notification Service Developer Guide

Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic

```
"Timestamp" : "2012-05-02T00:54:06.655Z",
"SignatureVersion" : "1",
"Signature" : "EXAMPLEw6JRNwmlLFQL4ICB0bnXrdB8ClRMTQFGBqwLp
GbM78tJ4etTwC5zU7O3tS6tGpey3eJedNdOJ+1fkIp9F2/LmNVKb5aF1Yq+9rk9ZiPph5Y1L
mWsDcyC5T+Sy9/umic5S0UQc2PEtgdpVBahwN0dMW4JPwk0kAJJztnc=",
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotific
ationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
"UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsub
scribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTop
ic: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 that the message request times out at 15 seconds. This means that if the message delivery failure was caused by a timeout, Amazon SNS will retry approximately 35 seconds after the previous delivery attempt. If you don't like the default delivery policy, you can set a different delivery policy on the endpoint.

To be clear, Amazon SNS attempts to retry only after a delivery attempt has failed. You can identify a message using the `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. For a list of trusted CAs, see [Certificate Authorities \(CA\) Recognized by Amazon SNS for HTTPS Endpoints \(p. 140\)](#).
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 AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select the topic.
3. Click 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 click **Create subscription**.
6. For the **Subscription request received!** message, click **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 \(p. 128\)](#) 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 \(p. 128\)](#), 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 [Controlling User Access to Your AWS Account](#).

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`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowPublishToMyTopic",
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:123456789012: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.

```
{
  "Version": "2012-10-17",
  "Id": "MyTopicPolicy",
  "Statement": [{
    "Sid": "Allow-publish-to-topic",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:123456789012: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` command, or the [Publish API](#).

If you followed [Step 1 \(p. 128\)](#), 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. In the left navigation pane, click **Topics** and then select a topic.
3. Click 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 click **Publish message**.

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

Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints

Topics

- [Applying Delivery Policies to Topics and Subscriptions \(p. 136\)](#)
- [Setting the Maximum Receive Rate \(p. 137\)](#)
- [Immediate Retry Phase \(p. 137\)](#)
- [Pre-Backoff Phase \(p. 138\)](#)
- [Backoff Phase \(p. 138\)](#)
- [Post-Backoff Phase \(p. 139\)](#)

A successful Amazon SNS delivery to an HTTP/HTTPS endpoint sometimes requires more than one attempt. This can be the case, for example, if the web server that hosts the subscribed endpoint is down for maintenance or is experiencing heavy traffic. If an initial delivery attempt doesn't result in a successful response from the subscriber, Amazon SNS attempts to deliver the message again. We call such an attempt a *retry*. In other words, a retry is an attempted delivery that occurs after the initial delivery attempt.

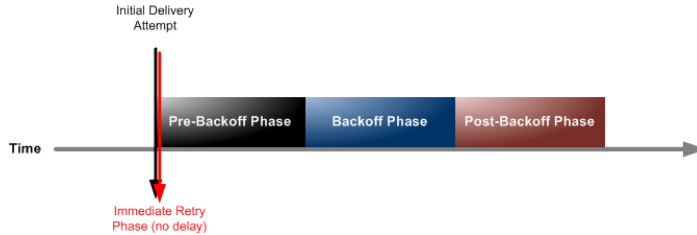
Amazon SNS only attempts a retry after a failed delivery attempt. Amazon SNS considers the following situations as a failed delivery attempt.

- HTTP status in the range 500-599.
- HTTP status outside the range 200-599.
- A request timeout (15 seconds). Note that if a request timeout occurs, the next retry will occur at the specified interval after the timeout. For example, if the retry interval is 20 seconds and a request times out, the start of the next request will be 35 seconds after the start of the request that timed out.
- Any connection error such as connection timeout, endpoint unreachable, bad SSL certificate, etc.

You can use delivery policies to control not only the total number of retries, but also the time delay between each retry. You can specify up to 100 total retries distributed among four discrete phases. The maximum lifetime of a message in the system is one hour. This one hour limit cannot be extended by a delivery policy.

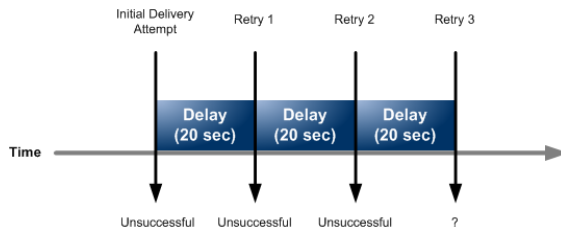
Amazon Simple Notification Service Developer Guide

Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints

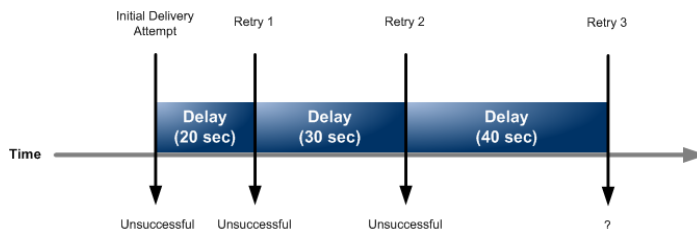


1. **Immediate Retry Phase (p. 137)**—Also called the *no delay phase*, this phase occurs immediately after the initial delivery attempt. The value you set for **Retries with no delay** determines the number of retries immediately after the initial delivery attempt. There is no delay between retries in this phase.
2. **Pre-Backoff Phase (p. 138)**—The pre-backoff phase follows the immediate retry phase. Use this phase to create a set of retries that occur before a backoff function applies to the retries. Use the **Minimum delay retries** setting to specify the number of retries in the Pre-Backoff Phase. You can control the time delay between retries in this phase by using the **Minimum delay** setting.
3. **Backoff Phase (p. 138)**—This phase is called the backoff phase because you can control the delay between retries in this phase using the retry backoff function. Set the **Minimum delay** and the **Maximum delay**, and then select a **Retry backoff function** to define how quickly the delay increases from the minimum delay to the maximum delay.
4. **Post-Backoff Phase (p. 139)**—The post-backoff phase follows the backoff phase. Use the **Maximum delay retries** setting to specify the number of retries in the post-backoff phase. You can control the time delay between retries in this phase by using the **Maximum delay** setting.

The backoff phase is the most commonly used phase. If no delivery policies are set, the default is to retry three times in the backoff phase, with a time delay of 20 seconds between each retry. The default value for both the **Minimum delay** and the **Maximum delay** is 20. The default number of retries is 3, so the default retry policy calls for a total of 3 retries with a 20 second delay between each retry. The following diagram shows the delay associated with each retry.



To see how the retry backoff function affects the time delay between retries, you can set the maximum delay to 40 seconds and leave the remaining settings at their default values. With this change, your delivery policy now specifies 3 retries during the backoff phase, a minimum delay of 20 seconds, and a maximum delay of 40 seconds. Because the default backoff function is linear, the delay between messages increases at a constant rate over the course of the backoff phase. Amazon SNS attempts the first retry after 20 seconds, the second retry after 30 seconds, and the final retry after 40 seconds. The following diagram shows the delay associated with each retry.



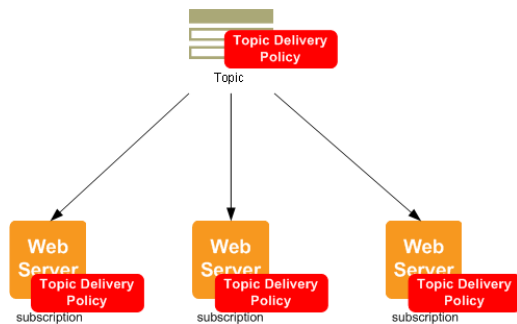
The maximum lifetime of a message in the system is one hour. This one hour limit cannot be extended by a delivery policy.

Note

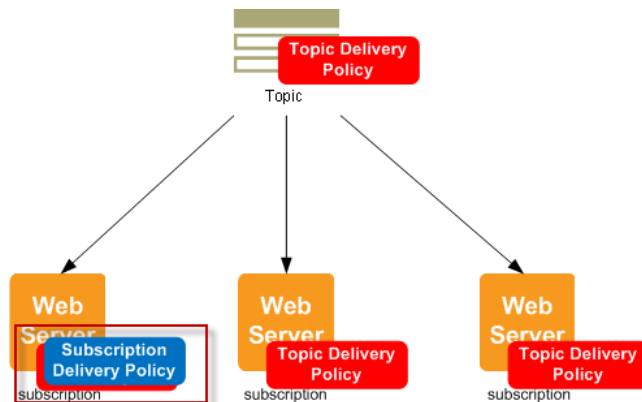
Only HTTP and HTTPS subscription types are supported by delivery policies. Support for other Amazon SNS subscription types (e.g., email, Amazon SQS, and SMS) is not currently available.

Applying Delivery Policies to Topics and Subscriptions

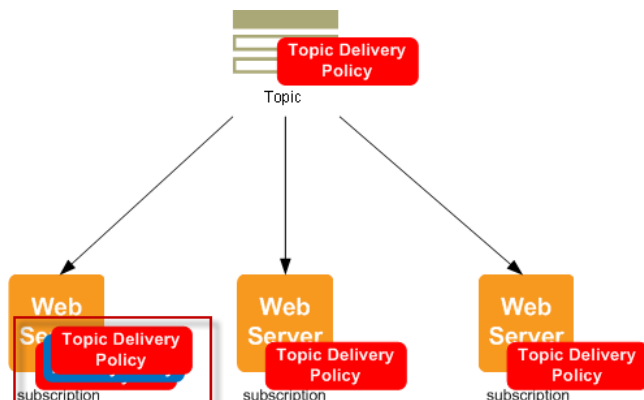
You can apply delivery policies to Amazon SNS topics. If you set a delivery policy on a topic, the policy applies to all of the topic's subscriptions. The following diagram illustrates a topic with a delivery policy that applies to all three subscriptions associated with that topic.



You can also apply delivery policies to individual subscriptions. If you assign a delivery policy to a subscription, the subscription-level policy takes precedence over the topic-level delivery policy. In the following diagram, one subscription has a subscription-level delivery policy whereas the two other subscriptions do not.



In some cases, you might want to ignore all subscription delivery policies so that your topic's delivery policy applies to all subscriptions even if a subscription has set its own delivery policy. To configure Amazon SNS to apply your topic delivery policy to all subscriptions, click **Ignore subscription override** in the **View/Edit Topic Delivery Policies** dialog box. The following diagram shows a topic-level delivery policy that applies to all subscriptions, even the subscription that has its own subscription delivery policy because subscription-level policies have been specifically ignored.



Setting the Maximum Receive Rate

You can set the maximum number of messages per second that Amazon SNS sends to a subscribed endpoint by setting the **Maximum receive rate** setting. Amazon SNS holds messages that are awaiting delivery for up to an hour. Messages held for more than an hour are discarded.

- To set a maximum receive rate that applies to all of a topic's subscriptions, apply the setting at the topic level using the **Edit Topic Delivery Policy** dialog box. For more information, see [To set the maximum receive rate for a topic \(p. 137\)](#).
- To set a maximum receive rate that applies to a specific subscription, apply the setting at the subscription level using the **Edit Subscription Delivery Policy** dialog box. For more information, see [To set the maximum receive rate for a subscription \(p. 137\)](#).

To set the maximum receive rate for a topic

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select the topic.
3. Click the **Other actions** drop-down list and select **Edit topic delivery policy**.
4. In the **Maximum receive rate** box, type an integer value (e.g., 2).
5. Click **Update policy** to save your changes.

To set the maximum receive rate for a subscription

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic ARN.
3. In the **Topic Details** pane, select a subscription and click **Edit topic delivery policy**.
4. In the **Maximum receive rate** box, type an integer value (e.g., 2).
5. Click **Update policy** to save your changes.

Immediate Retry Phase

The immediate retry phase occurs directly after the initial delivery attempt. This phase is also known as the No Delay phase because it happens with no time delay between the retries. The default number of retries for this phase is 0.

To set the number of retries in the immediate retry phase

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic ARN.
3. In the **Topic Details** pane, select **Edit topic delivery policy** from the **Other topic actions** drop-down list.
4. In the **Retries with no delay** box, type an integer value.
5. Click **Update policy** to save your changes.

Pre-Backoff Phase

The pre-backoff phase follows the immediate retry phase. Use this phase if you want to create a set of one or more retries that happen before the backoff function affects the delay between retries. In this phase, the time between retries is constant and is equal to the setting that you choose for the **Minimum delay**. The **Minimum delay** setting affects retries in two phases—it applies to all retries in the pre-backoff phase and serves as the initial time delay for retries in the backoff phase. The default number of retries for this phase is 0.

To set the number of retries in the pre-backoff phase

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic ARN.
3. In the **Topic Details** pane, select **Edit topic delivery policy** from the **Other topic actions** drop-down list.
4. In the **Minimum delay retries** box, type an integer value.
5. In the **Minimum delay** box, type an integer value to set the delay between messages in this phase.

The value you set must be less than or equal to the value you set for **Maximum delay**.
6. Click **Update policy** to save your changes.

Backoff Phase

The backoff phase is the only phase that applies by default. You can control the number of retries in the backoff phase using **Number of retries**.

Important

The value you choose for **Number of retries** represents the total number of retries, including the retries you set for **Retries with no delay**, **Minimum delay retries**, and **Maximum delay retries**.

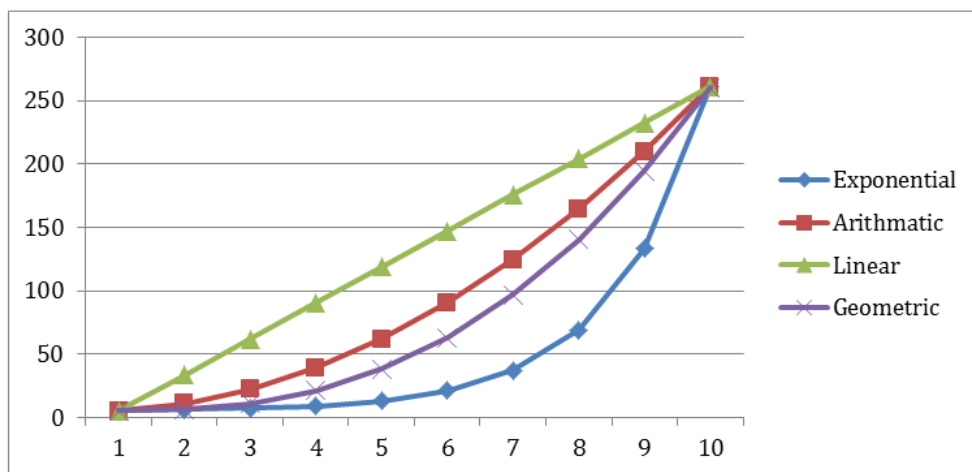
You can control the frequency of the retries in the backoff phase with three parameters.

- **Minimum delay**—The minimum delay defines the delay associated with the first retry attempt in the backoff phase.
- **Maximum delay**—The maximum delay defines the delay associated with the final retry attempt in the backoff phase.
- **Retry backoff function**—The retry backoff function 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 choose from four retry backoff functions.

- Linear
- Arithmetic
- Geometric
- Exponential

The following screen shot shows how each retry backoff function affects the delay associated with messages during the backoff period. The vertical axis represents the delay in seconds associated with each of the 10 retries. The horizontal axis represents the retry number. The minimum delay is 5 seconds, and the maximum delay is 260 seconds.



Post-Backoff Phase

The post-backoff phase is the final phase. Use this phase if you want to create a set of one or more retries that happen after the backoff function affects the delay between retries. In this phase, the time between retries is constant and is equal to the setting that you choose for the **Maximum delay**. The Maximum delay setting affects retries in two phases—it applies to all retries in the post-backoff phase and serves as the final time delay for retries in the backoff phase. The default number of retries for this phase is 0.

To set the number of retries in the post-backoff phase

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left navigation pane, click **Topics** and then select a topic ARN.
3. In the **Topic Details** pane, select **Edit topic delivery policy** from the **Other topic actions** drop-down list.
4. In the **Maximum delay retries** box, type an integer value.
5. In the **Maximum delay** box, type an integer value to set the delay between messages in this phase.

The value you set must be greater than or equal to the value you set for **Minimum delay**.

6. Click **Update policy** to save your changes.

Certificate Authorities (CA) Recognized by Amazon SNS for HTTPS Endpoints

If you subscribe an HTTPS endpoint to a topic, that endpoint must have a server certificate signed by a trusted Certificate Authority (CA). Amazon SNS will only deliver messages to HTTPS endpoints that have a signed certificate from a trusted CA that Amazon SNS recognizes. Amazon SNS recognizes the following CAs.

```
mozillacert81.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert99.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
swissignplatinumg2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
mozillacert145.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert37.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert4.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
amzninternalitseccag2, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
FA:07:FA:A6:35:D0:BC:98:72:3D:B3:08:8A:CD:CD:CD:3E:23:F9:ED
mozillacert70.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert88.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
mozillacert134.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
mozillacert26.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
verisignclass2g2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
mozillacert77.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert123.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
utndatacorpsgcca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert15.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
```

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert66.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert112.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
utnuserfirstclientauthemailca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
verisignc2gl.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
mozillacert55.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert101.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert119.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
verisignc3gl.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert44.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert108.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
mozillacert95.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
keynectisrootca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
mozillacert141.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
equifaxsecureglobalebusinesscal, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
baltimorecodesigningca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
mozillacert33.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert0.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
mozillacert84.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
mozillacert130.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert148.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
mozillacert22.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
verisignclg1.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
mozillacert7.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert73.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
mozillacert137.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
swissignsilverg2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert11.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert29.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
mozillacert62.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert126.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
soneraclasslca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
mozillacert18.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert51.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert69.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert115.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
verisignclass3g5ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
utnuserfirsthardwareca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
addtrustqualifiedca, Apr 22, 2014, trustedCertEntry,
```

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
Certificate fingerprint (SHA1):
4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert40.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
mozillacert58.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
verisignclass3g3ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert104.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
mozillacert91.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
thawtepersonalfreemailca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
certplusclass3pprimaryca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
verisignc3g4.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
swissigngoldg2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert47.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert80.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert98.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert144.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
starfieldclass2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert36.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert3.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
globalsignr2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert87.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert133.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
```

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert25.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignclasslg2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
mozillacert76.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert122.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
godaddysecurecertificationauthority, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
7C:46:56:C3:06:1F:7F:4C:0D:67:B3:19:A8:55:F6:0E:BC:11:FC:44
mozillacert14.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
equifaxsecureca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert65.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
mozillacert111.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
certumtrustednetworkca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert129.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
mozillacert54.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert100.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
mozillacert118.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
mozillacert151.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
thawteprimaryrootcag3, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
quovadisrootca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
thawteprimaryrootcag2, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
deprecateditsecca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
```

```
entrustrootcag2, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4  
mozillacert43.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A  
mozillacert107.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6  
trustcenterclass4caii, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50  
mozillacert94.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99  
mozillacert140.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7  
ttelesecglobalrootclass3ca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1  
amzninternalcorpca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
43:E3:E6:37:C5:88:05:67:91:37:E3:72:4D:01:7F:F4:1B:CE:3A:97  
mozillacert32.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C  
mozillacert83.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46  
verisignroot.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54  
mozillacert147.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4  
camerfirmachambersca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C  
mozillacert21.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB  
mozillacert39.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E  
mozillacert6.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4  
verisignuniversalrootca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54  
mozillacert72.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B  
geotrustuniversalca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79  
mozillacert136.pem, Apr 22, 2014, trustedCertEntry,
```

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
Certificate fingerprint (SHA1):
D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert10.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert28.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
mozillacert61.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert79.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert125.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert17.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert50.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert68.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
mozillacert114.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert57.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
verisignc2g3.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
verisignclass2g3ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
mozillacert103.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert90.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
verisignc3g3.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert46.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
godaddyclass2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
verisignc4g3.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
mozillacert97.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
```


Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
mozillacert143.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert35.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
mozillacert2.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
utnuserfirstobjectca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
mozillacert86.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert132.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
addtrustclasslca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert24.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
verisignclg3.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
mozillacert9.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
amzninternalrootca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
mozillacert75.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
entrustevca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
secomscrootca2, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
camerfirmachambersignca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
secomscrootcal, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert121.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert139.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert13.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
```

Amazon Simple Notification Service Developer Guide
Certificate Authorities for HTTPS Endpoints

```
mozillacert64.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert110.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert128.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
entrust2048ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
mozillacert53.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
mozillacert117.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert150.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
thawteserverca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
secomvalicertclass1ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert42.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
gtecybertrustglobalca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
mozillacert106.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
equifaxsecureebusinesscal, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert93.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
quovadisrootca3, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
soneraclass2ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert31.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
mozillacert49.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert82.pem, Apr 22, 2014, trustedCertEntry,
```

```
Certificate fingerprint (SHA1):  
2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3  
mozillacert146.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A  
baltimorecybertrustca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74  
mozillacert20.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61  
mozillacert38.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36  
mozillacert5.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6  
mozillacert71.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C  
verisignclass3g4ca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A  
mozillacert89.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D  
mozillacert135.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18  
camerfirmachamberscommerceca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1  
mozillacert27.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B  
verisignclass3g2ca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F  
mozillacert60.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3  
mozillacert78.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F  
certumca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18  
deutschetelekomrootca2, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF  
mozillacert124.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF  
mozillacert16.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13  
secomevrootcal, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):
```

Amazon Simple Notification Service Developer Guide

Certificate Authorities for HTTPS Endpoints

```
FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert67.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignr3ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert113.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
aolrootca2, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
trustcenteruniversalcai, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
aolrootcal, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert56.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
verisignc2g2.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
verisignclasslg3ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
mozillacert102.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
addtrustexternalca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
verisignclass3ca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
verisignc3g2.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
mozillacert45.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
verisignc4g2.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
digicertassuredidrootca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
verisignclasslca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
mozillacert109.pem, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
thawtepremiumserverca, Apr 22, 2014, trustedCertEntry,
Certificate fingerprint (SHA1):
E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
```

```
verisigntsaca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
BE:36:A4:56:2F:B2:EE:05:DB:B3:D3:23:23:AD:F4:45:08:4E:D6:56  
mozillacert96.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1  
mozillacert142.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85  
thawteprimaryrootca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81  
mozillacert34.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9  
mozillacert1.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C  
mozillacert85.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48  
valicertclass2ca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6  
mozillacert131.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79  
mozillacert149.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1  
geotrustprimaryca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96  
mozillacert23.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81  
verisignclg2.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47  
mozillacert8.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F  
mozillacert74.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F  
mozillacert120.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41  
geotrustglobalca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12  
mozillacert138.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD  
mozillacert12.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36  
comodoaaaca, Apr 22, 2014, trustedCertEntry,
```

```
Certificate fingerprint (SHA1):  
D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49  
mozillacert63.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E  
certplusclass2primaryca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB  
mozillacert127.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12  
ttelesecglobalrootclass2ca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9  
mozillacert19.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7  
digicerthighassuranceevrootca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25  
mozillacert52.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F  
mozillacert116.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21  
globalsignca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C  
mozillacert41.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3  
mozillacert59.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54  
mozillacert105.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC  
trustcenterclass2caii, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E  
mozillacert92.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0  
geotrustprimarycag3, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD  
entrustsslca, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39  
verisignc3g5.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5  
geotrustprimarycag2, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0  
mozillacert30.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):
```

```
E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE  
mozillacert48.pem, Apr 22, 2014, trustedCertEntry,  
Certificate fingerprint (SHA1):  
A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
```

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 \(p. 155\)](#).

To verify the signature of an Amazon SNS message when using HTTP query-based requests

1. Extract the name/value pairs from the JSON document in the body of the HTTP POST request that Amazon SNS sent to your endpoint. You'll be using the values of some of the name/value pairs to create the string to sign. When you are verifying the signature of an Amazon SNS message, it is critical that you convert the escaped control characters to their original character representations in the `Message` and `Subject` values. These values must be in their original forms when you use them as part of the string to sign. For information about how to parse the JSON document, see [Step 1: Make sure your endpoint is ready to process Amazon SNS messages \(p. 128\)](#).

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
2012-06-05T04:37:04.321Z
TopicArn
arn:aws:sns:us-east-1:123456789012:s4-MySNSTopic-1G1WEFCOXTCP
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-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:s4-MySNSTopic-1G1WEFCOXTCP&Token=2336412f37fb687f5c51e6e241d09c8058323f60b964268bfe08ce35640228c208a66d3621bd9f7b012918cf
d
c6e58f5f6f8d472f30e7eff06a305f0856e2503e9f13d497638697958f7460626024f8921c0d15c0847b5a405
Timestamp
```



```
2012-06-03T19:25:13.719Z
Token
2336412f37fb687f5d51e6e241d09c8058323f60b964268bfe08ce35640228c208a66d3621bd9f7b012918cf
d
c65b5f67ba472f30e7ef0ca15f70856c270f30971bd9f7313697958f246052424e92101d150047b7a405
TopicArn
arn:aws:sns:us-west-2:123456789012:s4-MySNSTopic-1G1WEFCOXTCP
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 \(p. 153\)](#)

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")) {
    //TODO: 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"))
{
    //TODO: 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());
    //TODO: Process the return value to ensure the endpoint is subscribed.
}
else if (messagetype.equals("UnsubscribeConfirmation")) {
```

```
        //TODO: 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 {
        //TODO: 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());
        InputStream inStream = url.openStream();
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        X509Certificate cert = (X509Certificate)cf.generateCertificate(in
Stream);
        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);
    }
}
```

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 \(p. 153\)](#).

```
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.get
Type().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;
}
```

Invoking Lambda functions using Amazon SNS notifications

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 [Using Amazon SNS Topic Attributes for Message Delivery Status](#).

Prerequisites

To invoke Lambda functions using Amazon SNS notifications, you need the following:

- Lambda function
- Amazon SNS topic

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

Configuring Amazon SNS with Lambda Endpoints with the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left **Navigation** pane, click **Topics**, and then click the topic to which you want to subscribe a Lambda endpoint.
3. Click **Actions** and then click **Subscribe to topic**.
4. In the **Protocol** drop-down box, select **AWS Lambda**.

Amazon Simple Notification Service Developer Guide Configuring Amazon SNS with Lambda Endpoints with the AWS Management Console

5. In the **Endpoint** drop-down box, select the ARN for the Lambda function.
6. In the **Version or Alias** drop-down box, select an available version or alias to use. You can also choose **\$LATEST** to specify the latest version of the Lambda function. If you do not want to specify a version or alias, you can also choose **default**, which is functionally the same as **\$LATEST**. For more information, see [AWS Lambda Function Versioning and Aliases](#).
7. Click **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 on how to create a sample message history store using SNS, Lambda, and Amazon DynamoDB, see the AWS Mobile Development blog [Invoking AWS Lambda functions via Amazon SNS](#).

Using Amazon SNS Topic Attributes for 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 Message Delivery Status Attributes with the AWS Management Console \(p. 162\)](#)
- [Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS Endpoints with the AWS SDKs \(p. 162\)](#)

Configuring Message Delivery Status Attributes with the AWS Management Console

The following steps describe how to use the console to configure the message delivery status attributes for notifications from Amazon SNS to a AWS Lambda endpoint.

To configure message delivery status for notifications from Amazon SNS to a Lambda endpoint:

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In the left **Navigation** pane, click **Topics**, and then click the topic to which you want to receive message delivery status information.
3. Click **Actions** and then click **Delivery status**.
4. Click the **Lambda** check box.
5. On the **Delivery Status** dialog box, click **Create IAM Roles**.

You will then be redirected to the IAM console.

6. Click **Allow** to give Amazon SNS write access to use CloudWatch Logs on your behalf.
7. Switch back to the **Delivery Status** dialog box and enter a number in the **Percentage of Success to Sample (0-100)** field for the percentage of successful messages sent for which you want to receive CloudWatch Logs.

Note

After you configure application attributes for message delivery status, all failed message deliveries generate CloudWatch Logs.

8. Finally, click **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](#).

Configuring Message Delivery Status Attributes for Topics Subscribed to Amazon SNS Endpoints with 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.

Java Example to Configure Topic Attributes

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-west-2:123456789012:MyTopic");
final static String attribName = ("LambdaSuccessFeedbackRoleArn");
final static String attribValue = ("arn:aws:iam::123456789012:role/SNSSuccess
Feedback");
```

```
SetTopicAttributesRequest setTopicAttributesRequest = new SetTopicAttributes
Request();
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](#).

Using Amazon SNS Message Attributes

Amazon Simple Notification Service (Amazon SNS) provides support for delivery of *message attributes* to Amazon SQS endpoints. Message attributes allow you to provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) about the message. Message attributes are optional and separate from, but sent along with, the message body to Amazon SQS endpoints. This information can be used by the receiver of the message to help decide how to handle the message without having to first process the message body. Each message can have up to 10 attributes. To specify message attributes, you can use the AWS software development kits (SDKs) or query API.

Important

To use message attributes with Amazon SQS endpoints, you must set the subscription attribute, **Raw Message Delivery**, to True. For more information about Raw Message Delivery, see [Appendix: Large Payload and Raw Message Delivery \(p. 183\)](#)

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

Topics

- [Message Attribute Items and Validation \(p. 164\)](#)
- [Message Attribute Data Types and Validation \(p. 165\)](#)
- [Reserved Message Attributes \(p. 165\)](#)
- [Using Message Attributes with the AWS SDKs \(p. 166\)](#)

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, 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 \(p. 165\)](#) 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 currently 256 KB (262,144 bytes).

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 will validate 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.
- **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 up to 38 digits of precision, and it can be between 10⁻¹²⁸ to 10⁺¹²⁶. 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

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

Push Notification Service	Reserved Message Attribute	Allowed Values
Baidu	AWS.SNS.MOBILE.BAIDU.DeployStatus (optional) AWS.SNS.MOBILE.BAIDU.MessageType (optional) AWS.SNS.MOBILE.BAIDU.MessageKey (optional)	1—development environment. 2—production environment. (default 1) 0—in-app message. 1—alert notification. (default 1) A short message identifier you can attach to your message
MPNS	AWS.SNS.MOBILE.MPNS.Type (required) AWS.SNS.MOBILE.MPNS.NotificationClass (required)	token (for tile notifications), toast, raw realtime, priority, regular

Push Notification Service	Reserved Message Attribute	Allowed Values
WNS	AWS.SNS.MOBILE.WNS.Type (required)	same as X-WNS-Type
	AWS.SNS.MOBILE.WNS.CachePolicy (optional)	same as X-WNS-Cache-Policy
	AWS.SNS.MOBILE.WNS.Group (optional)	same as X-WNS-Group
	AWS.SNS.MOBILE.WNS.Match (optional)	same as X-WNS-Match
	AWS.SNS.MOBILE.WNS.SuppressPopup (optional)	same as X-WNS-SuppressPopup
	AWS.SNS.MOBILE.WNS.Tag (optional)	same as X-WNS-Tag

For more information about using message attributes with Baidu, see [Using Message Attributes for Structuring the Message](#) (p. 66).

Using Message Attributes with the AWS SDKs

The [AWS SDKs](#) provide APIs in several languages for using message attributes with Amazon SNS. Java examples with message attributes are in the AWS sample file `SNSMobilePush.java`, which is included in the [snsmobilepush.zip](#) file.

When setting message attributes for a message, you can use either a *string* value or a *binary* value, but not both *string* and *binary* values.

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

Monitoring Amazon SNS with CloudWatch

Amazon SNS and CloudWatch are integrated so you can collect, view, and analyze metrics for every active Amazon SNS topic. Once you have configured CloudWatch for Amazon SNS, you can gain better insight into the performance of your Amazon SNS topics and Amazon SNS push notifications. 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 \(p. 169\)](#). For more information about Amazon SNS push notifications, see [Amazon SNS Mobile Push Notifications \(p. 36\)](#)

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 (i.e., any API call) on the topic.

Note

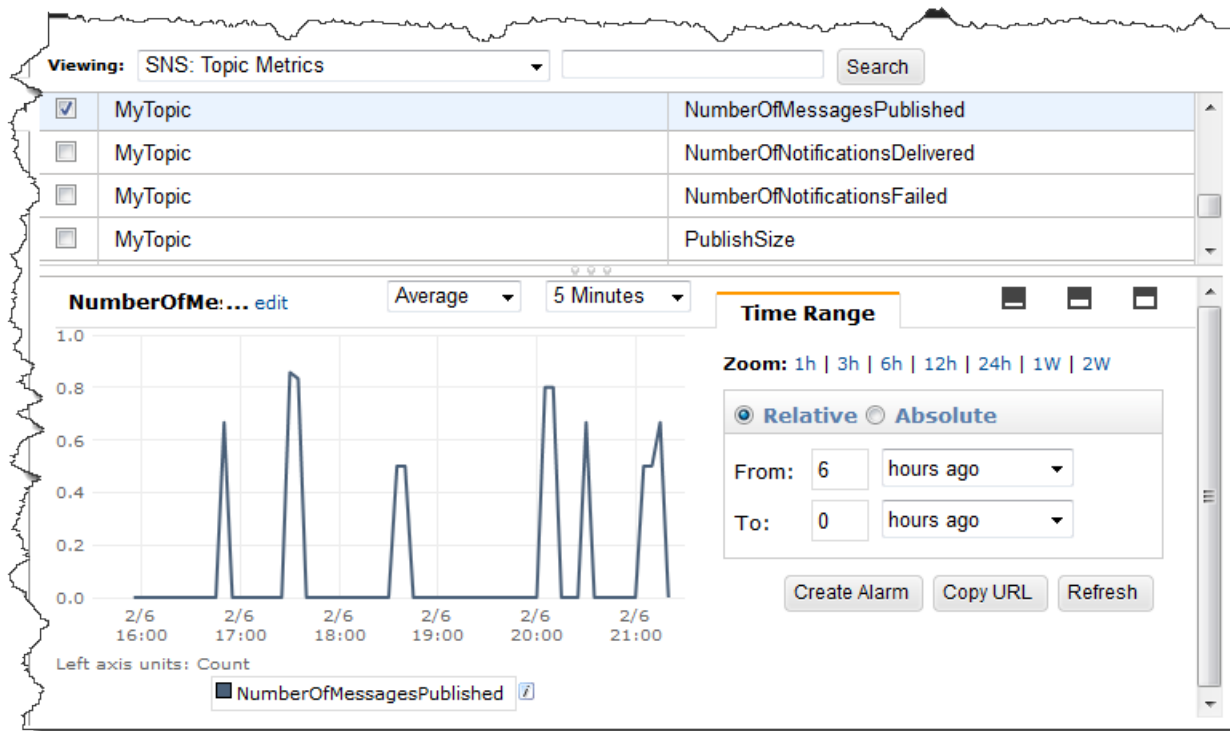
There is no charge for the Amazon SNS metrics reported in CloudWatch; they are provided as part of the Amazon SNS service.

Access 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 these different options.

To view metrics 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. Click **View Metrics**.
3. From the **Viewing** drop-down menu select either **SNS: Topic Metrics**, **SNS: Push Notifications by Application**, **SNS: Push Notifications by Application and Platform**, or **SNS: Push Notifications by Platform** to show the available metrics.
4. Click a specific item to see more detail, such as a graph of the data collected. For example, the following graph of the selected metric, **NumberOfMessagesPublished**, shows the average number of published Amazon SNS messages for a five-minute period throughout the time range of 6 hours.



To access metrics from the CloudWatch CLI

- Call `mon-get-stats`. You can learn more about this and other metrics-related functions in the [Amazon CloudWatch Developer Guide](#).

To access metrics from the CloudWatch API

- Call `GetMetricStatistics`. You can learn more about this and other metrics-related functions in the [Amazon CloudWatch API Reference](#).

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. Click **Alarms**, and then click 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 click **Continue**.
4. Fill in the **Name**, **Description**, **Threshold**, and **Time** values for the metric, and then click **Continue**.
5. Choose **Alarm** as the alarm state. If you want CloudWatch to send you an email when the alarm state is reached, either select a preexisting Amazon SNS topic or click **Create New Email Topic**. If

you click **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. Click **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, click **Create Alarm**.

For more information about using CloudWatch and alarms, see the [CloudWatch Documentation](#).

Amazon SNS Metrics

Amazon SNS sends the following metrics to CloudWatch.

Metric	Description
NumberOfMessagesPublished	The number of messages published. Units: <i>Count</i> Valid Statistics: Sum
PublishSize	The size of messages published. Units: <i>Bytes</i> Valid Statistics: Minimum, Maximum, Average and Count
NumberOfNotificationsDelivered	The number of messages successfully delivered. Units: <i>Count</i> Valid Statistics: Sum
NumberOfNotificationsFailed	The number of messages that Amazon SNS failed to deliver. This metric is applied after Amazon SNS stops attempting message deliveries to Amazon SQS, email, SMS, or mobile push endpoints. Each delivery attempt to an HTTP or HTTPS endpoint adds 1 to the metric. For all other endpoints, the count increases by 1 when the message is not delivered (regardless of the number of attempts). You can control the number of retries for HTTP endpoints; for more information, see Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints (p. 134) . Units: <i>Count</i> Valid Statistics: Sum, Average

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 GCM.
Application,Platform	Filters on application and platform objects, where the platform objects are for the supported push notification services, such as APNS and GCM.
Platform	Filters on platform objects for the push notification services, such as APNS and GCM.
TopicName	Filters on Amazon SNS topic names.

Logging Amazon Simple Notification Service API Calls By Using AWS CloudTrail

Amazon SNS is integrated with CloudTrail, a service that captures API calls made by or on behalf of Amazon SNS in your AWS account and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls made from the Amazon SNS console or from the Amazon SNS API. Using the information collected by CloudTrail, you can determine what request was made to Amazon SNS, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Amazon SNS Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to Amazon SNS actions are tracked in log files. Amazon SNS records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The following actions are supported:

- [AddPermission](#)
- [ConfirmSubscription](#)
- [CreatePlatformApplication](#)
- [CreatePlatformEndpoint](#)
- [CreateTopic](#)
- [DeleteEndpoint](#)
- [DeletePlatformApplication](#)
- [DeleteTopic](#)
- [GetEndpointAttributes](#)
- [GetPlatformApplicationAttributes](#)
- [GetSubscriptionAttributes](#)
- [GetTopicAttributes](#)

- [ListEndpointsByPlatformApplication](#)
- [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 click 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 log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the `userIdentity` field in the [CloudTrail Event Reference](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see [Configuring Amazon SNS Notifications](#).

You can also aggregate Amazon SNS log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see [Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket](#).

Understanding Amazon SNS Log File Entries

CloudTrail log files contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

The following example shows a CloudTrail log entry for 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"  
},  
]  
}
```

Appendix: Message and JSON Formats

Amazon SNS uses the following formats.

Topics

- [HTTP/HTTPS Headers](#) (p. 175)
- [HTTP/HTTPS Subscription Confirmation JSON Format](#) (p. 176)
- [HTTP/HTTPS Notification JSON Format](#) (p. 178)
- [HTTP/HTTPS Unsubscribe Confirmation JSON Format](#) (p. 179)
- [SetSubscriptionAttributes Delivery Policy JSON Format](#) (p. 181)
- [SetTopicAttributes Delivery Policy JSON Format](#) (p. 181)

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 that Amazon SNS will not 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-
1:123456789012: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" :
"236412f37f687f551e6241d09c805a55730d712f79cc5f698866b92768b6a71a6f3db718542856ad024809eeec29417f1f0260c582af
bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "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.\n\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:MyTop
ic&Token=236412f37f687f551e6241d09c805a55730d712f79cc5f698866b92768b6a71a6f3db718542856ad024809eeec29417f1f0260c582af
bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH+DcEwjAPg8O9mY8dReBSwksfg2S7WKQcikcNK
WLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDVrY0Ad8L10Hs3zH8lmtnP5uvvol
IC1CXGu43obcgFxeL3khZl8IKvO61GWB6jI9b5+gLPoBc1Q=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

You can download the following JSON file to view the definition of the JSON format for a subscription confirmation: <https://sns.us-west-2.amazonaws.com/doc/2010-03-31/SubscriptionConfirmation.json>.

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 that 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" : "EXAMPLEw6JRNwm1LFQL4ICB0bnXrdB8ClRMTQFGBqwLp
GbM78tJ4etTwC5zU703tS6tGpey3ejedNdOJ+1fkIp9F2/LmNVKb5aF1Yq+9rk9ZiPph5YlLmWsD
cyC5T+Sy9/umic5S0UQc2PEtgdpVBahwNodMW4JPwk0kAJJztnc=" ,
"SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem" ,
"UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&Sub
scriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-
413945fb5a96"
}
```

You can download the following JSON file to view the definition of the JSON format for a notification:
<https://sns.us-west-2.amazonaws.com/doc/2010-03-31/Notification.json>.

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-
1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfc21c8f55.\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" : "2336412f37fb687f5d51e6e241d09c805a5a57b30d712f7948a98bac386edfe3e10314e873973b3e0a3c09119b722dedf2b5e31c59b13edbb26417c19f109351e6f2169efa9085ffe97e10535f4179ac1a03590b0f541f209c190f9ae23219ed6c470453e06c19b5ba9fdd27dab7c7",
  "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.\n\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=2336412f37fb687f5d51e6e241d09c805a5a57b30d712f7948a98bac386edfe3e10314e873973b3e0a3c09119b722dedf2b5e31c59b13edbb26417c19f109351e6f2169efa9085ffe97e10535f4179ac1a03590b0f541f209c190f9ae23219ed6c470453e06c19b5ba9fdd27dab7c7",
  "Timestamp" : "2012-04-26T20:06:41.581Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEHXgJmXqnqsHTlqOck7TIzsnk8zpJJoQbr81eD+8kAHcke3Clc4VPOvdPzo9s/vR9GOznKab6sjGxE8uwqDI9HwpDm81GxSlFGuwCruWeecnt7MdJCNh0XK4XQCbtGoXB762ePJfaSWi9tYwzW65zAFU04WkNBkNsIf60=",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

You can download the following JSON file to view the definition of the JSON format for an unsubscribe confirmation: <https://sns.us-west-2.amazonaws.com/doc/2010-03-31/UnsubscribeConfirmation.json>.

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-1.amazonaws.com/  
?Action=SetSubscriptionAttributes  
&SubscriptionArn=arn%3Aaws%3Asns%3Aus-east-1%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-1.amazonaws.com/  
?Action=SetTopicAttributes  
&TopicArn=arn%3Aaws%3Asns%3Aus-east-1%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*.

Appendix: Large Payload and Raw Message Delivery

With Amazon SNS and Amazon SQS, you now have the ability to send large payload messages that are up to 256KB (262,144 bytes) in size. To send large payloads (messages between 64KB and 256KB), you must use an AWS SDK that supports AWS Signature Version 4 (SigV4) signing. To verify whether SigV4 is supported for an AWS SDK, check the SDK release notes.

In addition to sending large payloads, with Amazon SNS you can now enable raw message delivery for messages delivered to either Amazon SQS endpoints or HTTP/S endpoints. This eliminates the need for the endpoints to process JSON formatting, which is created for the Amazon SNS metadata when raw message delivery is not selected. For example when enabling raw message delivery for an Amazon SQS endpoint, the Amazon SNS metadata is not included and the published message is delivered to the subscribed Amazon SQS endpoint as is. When enabling raw message delivery for HTTP/S endpoints, the messages will contain an additional HTTP header `x-amz-sns-rawdelivery` with a value of `true` to indicate that the message is being published raw instead of with JSON formatting. This enables those endpoints to understand what is being delivered and enables easier transition for subscriptions from JSON to raw delivery.

To enable raw message delivery using one of the AWS SDKs, you must use the `SetSubscriptionAttribute` action and configure the `RawMessageDelivery` attribute with a value of `true`. The default value is `false`.

Enabling Raw Message Delivery with the AWS Management Console

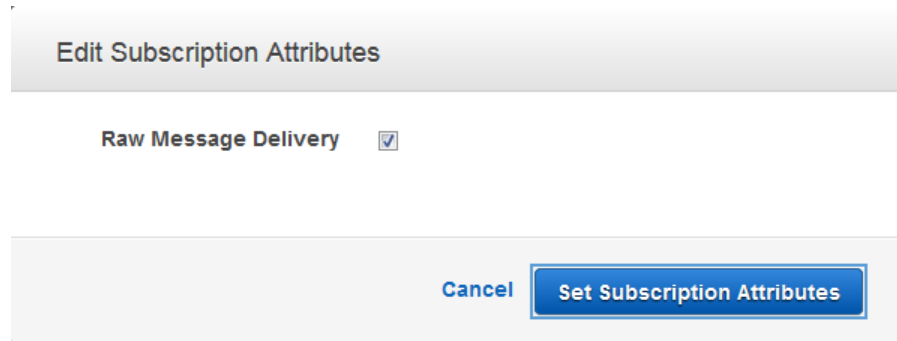
You can enable raw message delivery using the AWS Management Console by setting the **Raw Message Delivery** subscription attribute to a value of `true`.

To enable raw message delivery with the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select a topic that is subscribed to either an Amazon SQS endpoint or an HTTP/S endpoint and then click the topic ARN.

The **Topic Details** page appears.

3. Select the **Subscription ID** and then click the Other subscription actions drop-down box.
4. Click **Edit subscription attributes**, select **Raw Message Delivery**, and then click **Set subscription attributes**.



Edit Subscription Attributes

Raw Message Delivery

Cancel **Set Subscription Attributes**

Document History

The following table describes the important changes to the documentation since the last release of the *Amazon SNS Developer Guide*.

- **API version:** 2010-03-31
- **Latest documentation update:** September 23, 2015

Change	Description	Date Changed
Platform endpoints and device tokens	Added a topic about how to create a platform endpoint and manage device tokens for Amazon SNS mobile push notification. For more information, see Create a Platform Endpoint and Manage Device Tokens (p. 83) .	September 23, 2015
Application event notifications	Added a topic about how to trigger notifications when certain application events occur. For more information, see Application Event Notifications (p. 93) .	September 23, 2015
New support for VoIP and Mac OS X push notifications	Added new topics about sending push notification messages to VoIP and Mac OS X apps using Apple Push Notification Service. For more information, see Getting Started with Apple Push Notification Service (p. 44) .	June 15, 2015
Invoking AWS Lambda functions	Added a topic on how to invoke Lambda functions using Amazon SNS notifications. For more information, see Invoking Lambda functions using Amazon SNS notifications (p. 159) .	April 09, 2015
Using Amazon SNS topic attributes for message delivery status	Added a topic on using Amazon SNS topic attributes for message delivery status. For more information, see Using Amazon SNS Topic Attributes for Message Delivery Status (p. 161) .	April 09, 2015
Support to log the delivery status of push notification messages	Added a topic on using Amazon SNS application attributes for message delivery status. For more information, see Using Amazon SNS Application Attributes for Message Delivery Status (p. 90) .	February 05, 2015

Change	Description	Date Changed
Support for AWS CloudTrail with Amazon Simple Notification Service	Added a topic on logging Amazon SNS API calls by using CloudTrail. For more information, see Logging Amazon Simple Notification Service API Calls By Using AWS CloudTrail (p. 171) .	October 09, 2014
Amazon SNS mobile push high-level steps	Added a topic about the high-level steps you must perform to use Amazon SNS mobile push. This information should help you gain a better understanding of the steps involved when using the Amazon SNS mobile push APIs. For more information, see Amazon SNS Mobile Push High Level Steps (p. 38) .	October 09, 2014
Support for authenticated messages with Microsoft Push Notification Service for Windows Phone	Updated a topic on how to send authenticated messages with MPNS. For more information, see Getting Started with MPNS (p. 72) .	August 19, 2014
Support for setting a Time To Live (TTL) message attribute for mobile push notification messages	Added a topic on how to specify expiration metadata for a mobile push notification message. For more information, see Using the Amazon SNS Time To Live (TTL) Message Attribute for Mobile Push Notifications (p. 95) .	July 10, 2014
Support for Baidu Cloud Push, Microsoft Push Notification Service for Windows Phone, and Windows Push Notification Services	Added topics on how to use Baidu, MPNS, and WNS, with Amazon SNS to send push notification messages to mobile devices. For more information, see Getting Started with Baidu Cloud Push (p. 50) , Getting Started with MPNS (p. 72) , and Getting Started with WNS (p. 75) .	June 12, 2014
Message attributes	Message attributes allow you to provide structured metadata items about a message. For more information, see Using Amazon SNS Message Attributes (p. 164) .	June 12, 2014
Amazon SNS samples in Java	Added a section about using the AWS SDK for Java with Amazon SNS. Examples in this section show how to create a new Amazon SNS client, set the Amazon SNS endpoint to use, and create a new topic. In addition, examples are provided on how to subscribe to, publish to, and delete a topic. For more information, see Using the AWS SDK for Java with Amazon SNS (p. 9) .	April 23, 2014
Mobile push notifications	Added a topic about how to create and send custom platform-specific payloads in messages to mobile devices. For more information, see Send Custom Platform-Specific Payloads in Messages to Mobile Devices (p. 88) .	December 17, 2013
Mobile push notifications	Added support to send notification messages directly to apps on mobile devices. For more information, see Amazon SNS Mobile Push Notifications (p. 36) .	August 13, 2013
Initial Release	This is the first release of the <i>Amazon SNS Developer Guide</i> .	May 1, 2013