

---

# **Amazon Simple Notification Service**

**Getting Started Guide**

**API Version 2010-03-31**



# Amazon Web Services

## Amazon Simple Notification Service: Getting Started Guide

Amazon Web Services

Copyright © 2013 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks or registered trademarks of Amazon: Amazon, Amazon.com, Amazon.com Design, Amazon DevPay, Amazon EC2, Amazon Web Services Design, AWS, CloudFront, EC2, Elastic Compute Cloud, Kindle, and Mechanical Turk. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. 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.

## Amazon Simple Notification Service Getting Started Guide

---

Get Started with Amazon SNS .....	1
Sign Up for Amazon SNS .....	2
Create a Topic .....	3
Subscribe to a Topic .....	5
Publish to a Topic .....	7
Clean Up .....	11
Where Do I Go from Here? .....	13
Please Provide Feedback .....	15
Appendix A: Managing Access to Your Amazon SNS Topics .....	16
Overview .....	17
When to Use Access Control .....	17
Key Concepts .....	17
Architectural Overview .....	19
Using the Access Policy Language .....	21
Evaluation Logic .....	22
Example Cases for Amazon SNS Access Control .....	25
How to Write a Policy .....	30
Basic Policy Structure .....	30
Element Descriptions .....	30
Supported Data Types .....	40
Special Information for Amazon SNS Policies .....	41
Controlling User Access to Your AWS Account .....	42
Appendix B: Sending and Receiving SMS Notifications Using Amazon SNS .....	50
Appendix C: Sending Amazon SNS Messages to Amazon SQS Queues .....	59
Sending Amazon SNS messages to an Amazon SQS queue in a different account .....	66
Using an AWS CloudFormation Template to Create a Topic that Sends Messages to Amazon SQS Queues .....	69
Appendix D: Sending Amazon SNS Messages to HTTP/HTTPS Endpoints .....	75
Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints .....	82
Certificate Authorities (CA) Recognized by Amazon SNS for HTTPS Endpoints .....	93
Verifying the Signatures of Amazon SNS Messages .....	98
Using an AWS CloudFormation Template to Set Up a Topic and an HTTP Endpoint .....	100
Example Code for an Amazon SNS Endpoint Java Servlet .....	103
Appendix E: Message and JSON Formats .....	107
About This Guide .....	117

# Get Started with Amazon SNS

---

Amazon Simple Notification Service (Amazon SNS) is a web service that enables applications, end-users, and devices to instantly send and receive notifications from the cloud. You can accomplish these tasks using the AWS Management Console, which is a simple and intuitive web interface. This guide introduces you to Amazon SNS using the AWS Management Console.

This guide walks you through creating a topic, subscribing to the topic, publishing a message to the topic, unsubscribing from the topic, and deleting the topic. In addition there is an appendix that describes how to control access to Amazon SNS topics.

The following video walks you through the example presented in this guide: [Getting Started with Amazon SNS](#)

# Sign Up for Amazon SNS

---

To use Amazon SNS, you need an AWS account. If you don't already have one, you'll be prompted to create one when you sign up for Amazon SNS.

## To sign up for Amazon SNS

1. Go to <http://aws.amazon.com/sns/> and click **Sign Up for Amazon SNS**.
2. Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.

## Create a Topic

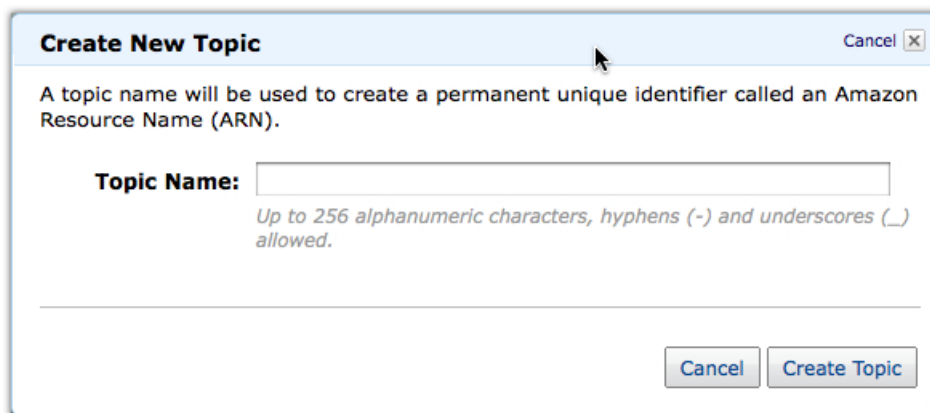
---

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. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Click **Create New Topic**.

The **Create New Topic** dialog box appears.



**Create New Topic** Cancel X

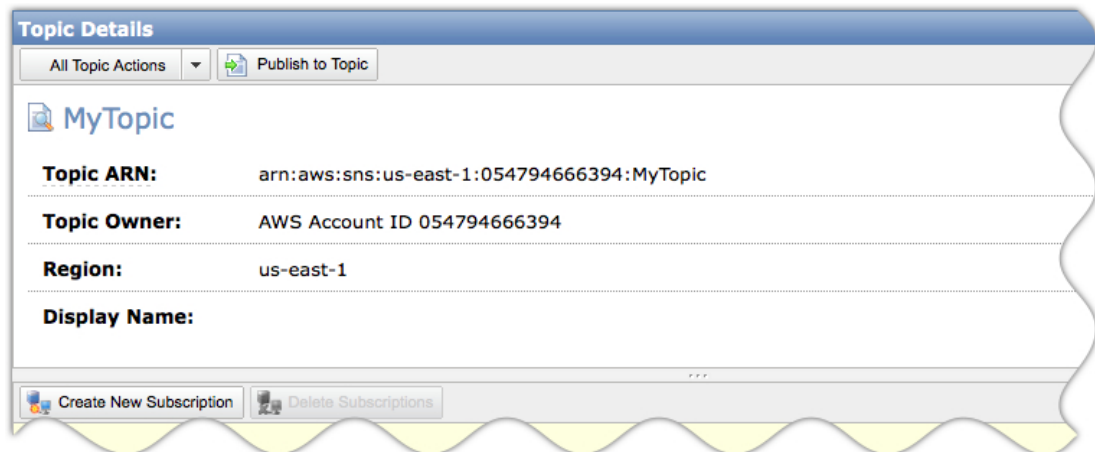
A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN).

**Topic Name:**

*Up to 256 alphanumeric characters, hyphens (-) and underscores (\_) allowed.*

Cancel Create Topic

3. Enter a topic name in the **Topic Name** field.  
The examples that follow use the topic name *MyTopic*.
4. Click **Create Topic**.  
The new topic appears in the **Topic Details** page.



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



# Subscribe to a Topic

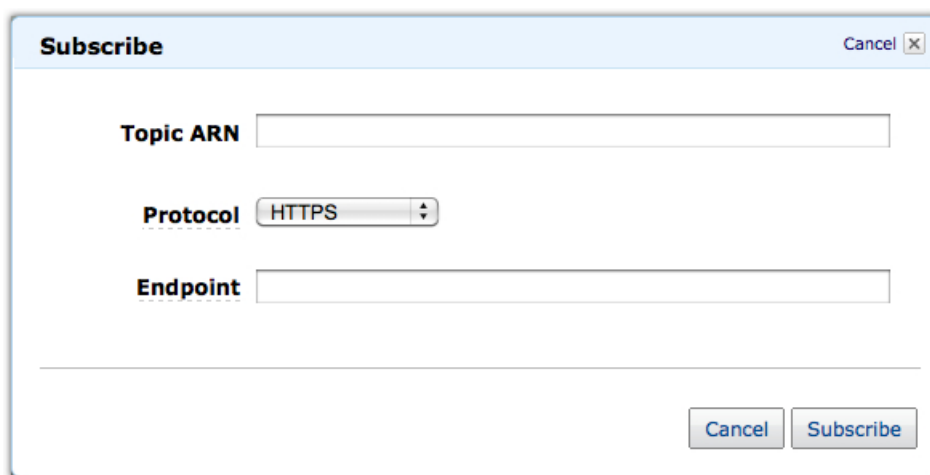
---

To receive messages published to a topic, you have to subscribe an endpoint to that topic. An endpoint is a web server, an email address, or an 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. In the [AWS Management Console](#), click **My Subscriptions** in the **Navigation** pane.  
The **My Subscriptions** page opens.
2. Click the **Create New Subscription** button.  
The **Subscribe** 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-east-1:054794666397:MyTopic`.
4. Select **Email** in the **Protocol** drop-down box.
5. Enter an email address you can use to receive the notification in the **Endpoint** field.

**Important**

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

6. Click **Subscribe**.
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. In the [AWS Management Console](#), click the topic you want to publish to, under **My Topics** in the **Navigation** pane.  
The **Topic Details** page opens.
2. Click the **Publish to Topic** button.  
The **Publish** dialog box appears.

**Publish** Cancel X

**Topic Name:** MyTopic

**Subject:**   
Up to 100 printable ASCII characters (optional)

**Message:**   
Up to 64KB of Unicode text

☒ Use same message body for all protocols  
☐ Use different message body for different protocols

For SMS notifications, it is best to leave the Subject field blank and place your text in the Message field to send a maximum of 140 characters. If the Subject field is not blank, the text in the Subject field will be used as content for the SMS messages.

Cancel Publish Message

3. Enter a subject line for your message in the **Subject** field.
4. Enter a brief message in the **Message** field.
5. Click **Publish Message**.  
A confirmation dialog box appears.
6. Click **Close** to close the confirmation dialog box.

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. Click the topic you want to publish to, under **My Topics** in the **Navigation** pane.  
The **Topic Details** page opens.
3. Click the **Publish to Topic** button.  
The **Publish** dialog box appears.

**Publish** Cancel

**Topic Name:** MyTopic

**Subject:**   
Up to 100 printable ASCII characters (optional)

**Message:**  
Up to 64KB of Unicode text

☒ Use same message body for all protocols  
☐ Use different message body for different protocols

For SMS notifications, it is best to leave the Subject field blank and place your text in the Message field to send a maximum of 140 characters. If the Subject field is not blank, the text in the Subject field will be used as content for the SMS messages.

Cancel Publish Message

## Amazon Simple Notification Service Getting Started Guide

### Create Different Messages for Each Protocol

---

4. Select **Use different message body for different protocols**.

**Publish** Cancel

**Topic Name:** MyTopic

**Subject:**   
*Up to 100 printable ASCII characters (optional)*

**Message:**

```
{  
  "default": "<enter your message here>",  
  "email": "<enter your message here>",  
  "sqs": "<enter your message here>",  
  "sms": "<enter your message here>",  
  "http": "<enter your message here>",  
  "https": "<enter your message here>"  
}
```

  
*Up to 64KB of Unicode text*

☐ Use same message body for all protocols  
☒ Use different message body for different protocols

*For SMS notifications, it is best to leave the Subject field blank and place your text in the Message field to send a maximum of 140 characters. If the Subject field is not blank, the text in the Subject field will be used as content for the SMS messages.*

Cancel Publish Message

5. Enter a subject line for your message in the **Subject** field.
6. Enter a brief message in the **Message** field for each protocol of interest.  
In the following example, messages are specified for the default, email, and sms protocols. Do not delete any protocols from the list.

## Amazon Simple Notification Service Getting Started Guide

### Create Different Messages for Each Protocol

---

**Publish** Cancel X

**Topic Name:** MyTopic

**Subject:**   
*Up to 100 printable ASCII characters (optional)*

**Message:** {  
"default": "<enter your message here>",  
"email": "<enter your message here>",  
"sqs": "<enter your message here>",  
"sms": "<enter your message here>",  
"http": "<enter your message here>",  
"https": "<enter your message here>"  
}  
*Up to 64KB of Unicode text*

☐ Use same message body for all protocols  
☒ Use different message body for different protocols

*For SMS notifications, it is best to leave the Subject field blank and place your text in the Message field to send a maximum of 140 characters. If the Subject field is not blank, the text in the Subject field will be used as content for the SMS messages.*

Cancel Publish Message

7. Click **Publish Message**.  
A confirmation dialog box appears.
8. Click **Close** to close the confirmation dialog box.

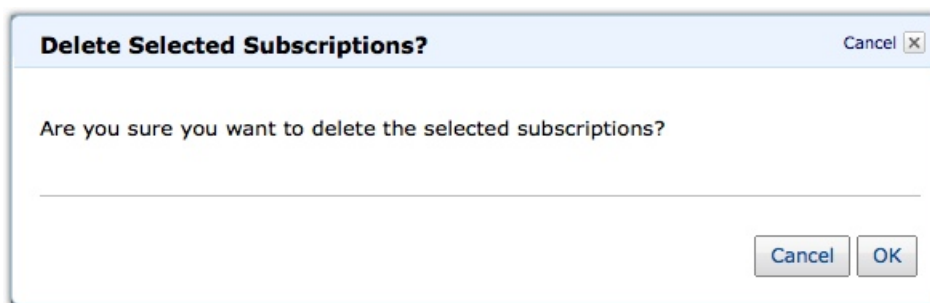
# Clean Up

---

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

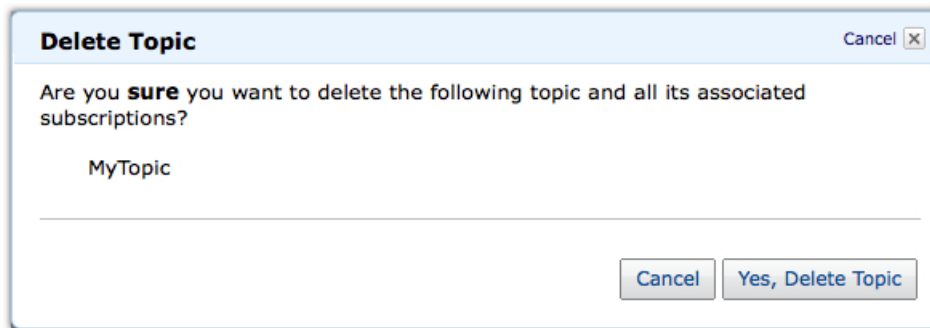
1. In the [AWS Management Console](#), click **My Subscriptions** in the **Navigation** pane.  
The **My Subscriptions** page opens.
2. Click the checkbox next to your topic in the subscription list. This will be the only listing on the page, unless you created more than one subscription.
3. Click the **Delete Subscriptions** button.  
The **Delete Selected Subscriptions?** confirmation dialog box appears.



4. Click **OK**.

## To delete a topic

1. Click the topic you want to delete, under **My Topics** in the **Navigation** pane.  
The **Topic Details** page opens.
2. Click the **All Topic Actions** drop-down list and select **Delete Topic**.  
The **Delete Topic** confirmation dialog box appears.



3. Click **Yes, Delete Topic**.

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

Congratulations! You successfully created, published, subscribed to, and deleted a topic. For more information about Amazon SNS and how to continue, see [Where Do I Go from Here? \(p. 13\)](#).

Your input is important to us to help make our documentation helpful and easy to use. Please take a minute to give us your feedback on how well we were able to help you get started with Amazon SNS. Just click this [Feedback Link](#) link. Thank you.



# Where Do I Go from Here?

---

## Amazon SNS Protocols and Access

Amazon SNS supports other protocols beside email. You can use HTTP, HTTPS, and Amazon Simple Queue Service queues. You have detailed control over what endpoints a topic allows, and who is able to publish to a topic and under what conditions. For more information about access restrictions, see [Appendix A: Managing Access to Your Amazon SNS Topics \(p. 16\)](#).

## Other Ways to Access Amazon SNS

This guide has shown you how to create a topic, subscribe to a topic, publish to a topic, unsubscribe from a topic, and delete a topic using the AWS Management Console. You can continue using Amazon SNS through the console, or try one of the other interfaces.

### Continue Using the Console

The AWS Management Console includes other functions besides just creating and publishing topics. The console has online help to assist you (just click the **Help** button in the console).

### Use the Command Line Interface

To get started with Amazon SNS's Java-based command line interface (CLI), download the CLI at <http://aws.amazon.com/developertools/3688>. The CLI offers a fast way to execute all the Amazon SNS functions without coding to the API or using a library.

### Use an API Library

If you prefer to use Amazon SNS through a programmatic interface, there are libraries and resources available for the following languages:

- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for PHP](#)

## Code Directly to the Web Service API

If you want to write code directly to the Amazon SNS Query API, go to the [Amazon SNS API Reference](#).

## AWS Account and Security Credentials

So far you signed up for the service, got an AWS account and security credentials, and then completed a short exercise covering the essential product functions. Now that you're finished with the exercise, we recommend that you check with an administrator or coworker in your organization to determine if he or she already has an AWS account and security credentials for you to use in future interactions with AWS.

If you're an account owner or administrator and want to know more about AWS Identity and Access Management, go to the product description at <http://aws.amazon.com/iam> or to the technical documentation at [Using AWS Identity and Access Management](#).

# Please Provide Feedback

---

Your input is important to help make our documentation helpful and easy to use. Please tell us about your experience getting started with Amazon SNS by completing our [Getting Started Survey](#).

Thank you.

# Appendix A: Managing Access to Your Amazon SNS Topics

---

## Topics

- [Overview \(p. 17\)](#)
- [How to Write a Policy \(p. 30\)](#)
- [Special Information for Amazon SNS Policies \(p. 41\)](#)
- [Controlling User Access to Your AWS Account \(p. 42\)](#)

Amazon SNS supports other protocols besides 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 access through the use of *access control policies*.

The main portion of this appendix 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 appendix is service-agnostic, there are some Amazon SNS-specific details you need to know. For more information, see [Special Information for Amazon SNS Policies \(p. 41\)](#).

# Overview

## Topics

- [When to Use Access Control](#) (p. 17)
- [Key Concepts](#) (p. 17)
- [Architectural Overview](#) (p. 19)
- [Using the Access Policy Language](#) (p. 21)
- [Evaluation Logic](#) (p. 22)
- [Example Cases for Amazon SNS Access Control](#) (p. 25)

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. 26).
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see [Limiting Subscriptions to HTTPS](#) (p. 26).
- You want to allow Amazon SNS to publish messages to your Amazon SQS queue. For more information, see [Publishing to an SQS Queue](#) (p. 27).

## 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

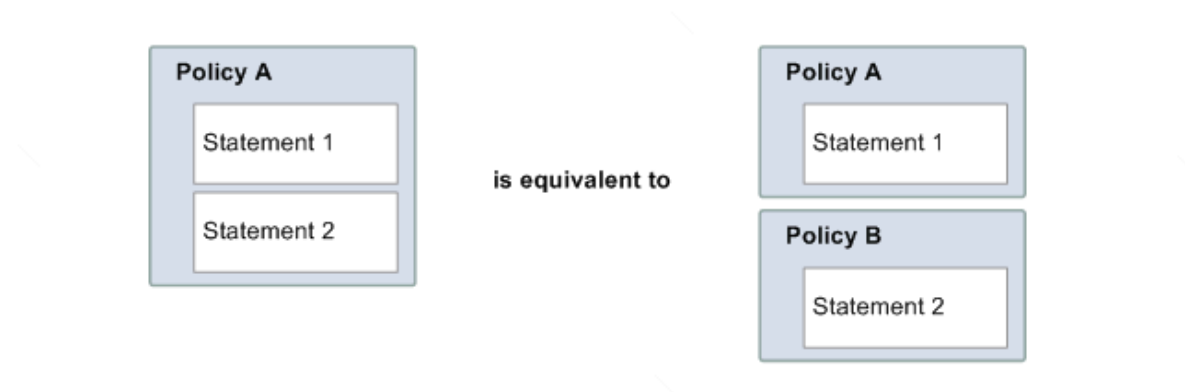
A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *publish* (B) to *TopicA* (C) as long as *she uses the HTTP protocol* (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 about conditions, see [Condition \(p. 34\)](#). For more information about the available keys, see [Available Keys \(p. 36\)](#).

## 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 \(p. 22\)](#).

## 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 \(p. 22\)](#).

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 \(p. 22\)](#).

## 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

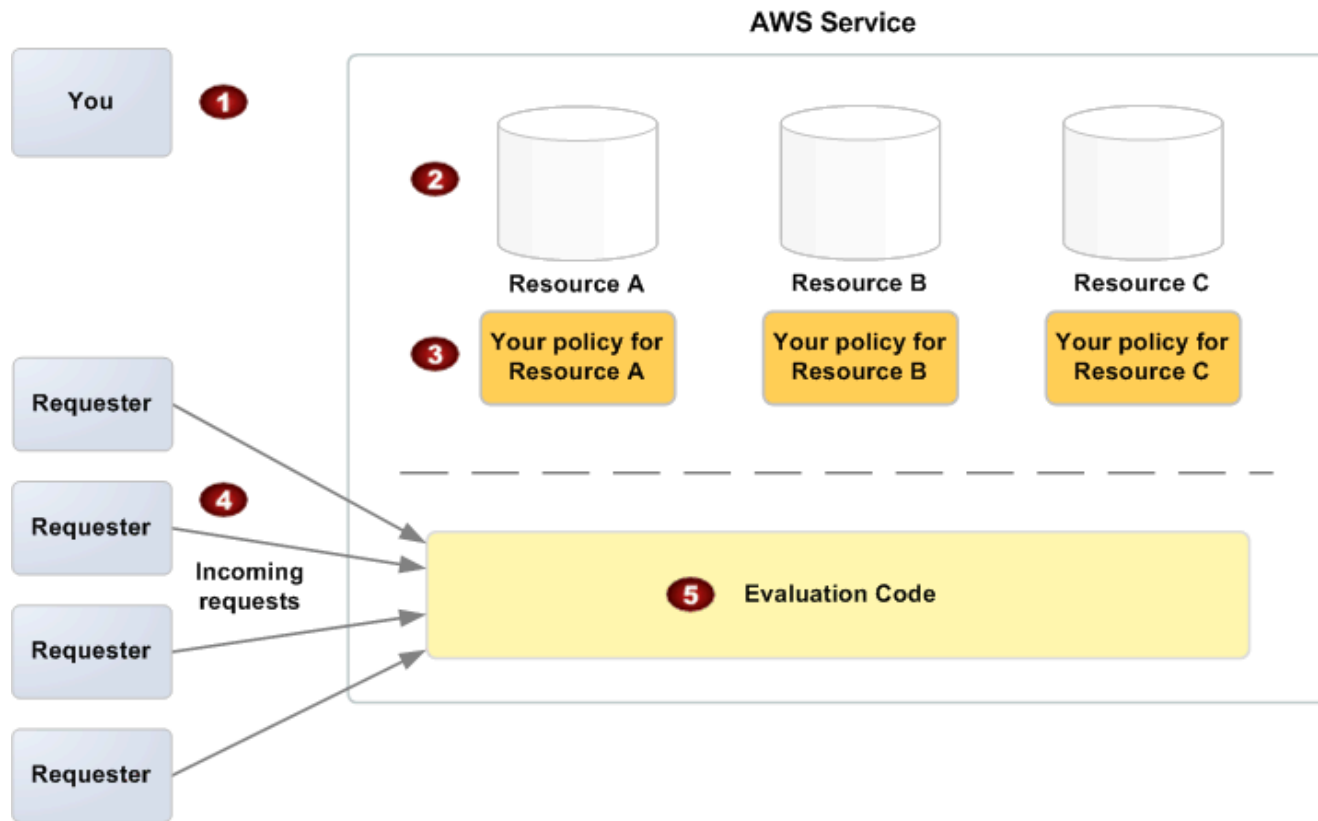
An *explicit deny* results from a statement that has effect=deny, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

## Architectural Overview

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

# Amazon Simple Notification Service Getting Started Guide

## Architectural Overview

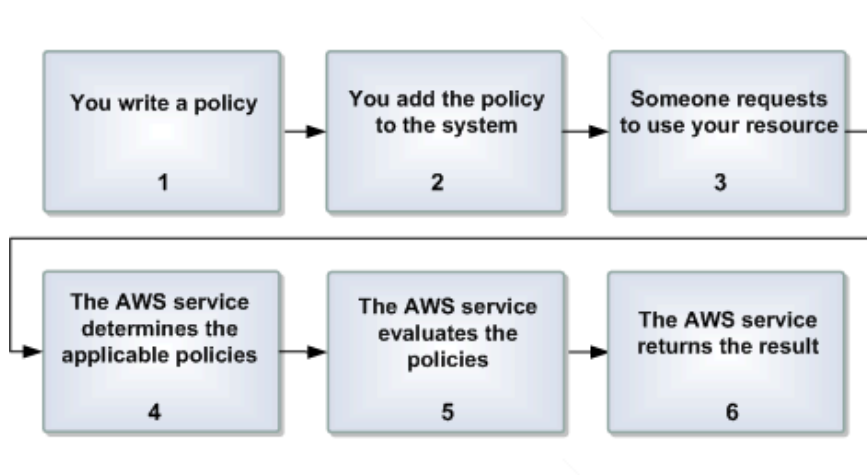


<b>1</b>	You, the resource owner.
<b>2</b>	Your resources (contained within the AWS service; e.g., SQS queues).
<b>3</b>	Your policies. Typically you have one policy per resource, although you could have multiple. The AWS service itself provides an API you use to upload and manage your policies. For information about the content of the policies, see <a href="#">How to Write a Policy (p. 30)</a> .
<b>4</b>	Requesters and their incoming requests to the AWS service.
<b>5</b>	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 <a href="#">Evaluation Logic (p. 22)</a> .



## 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. For more information, see <a href="#">How to Write a Policy (p. 30)</a> .
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 <a href="#">Evaluation Logic (p. 22)</a> .
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

### Related Topics

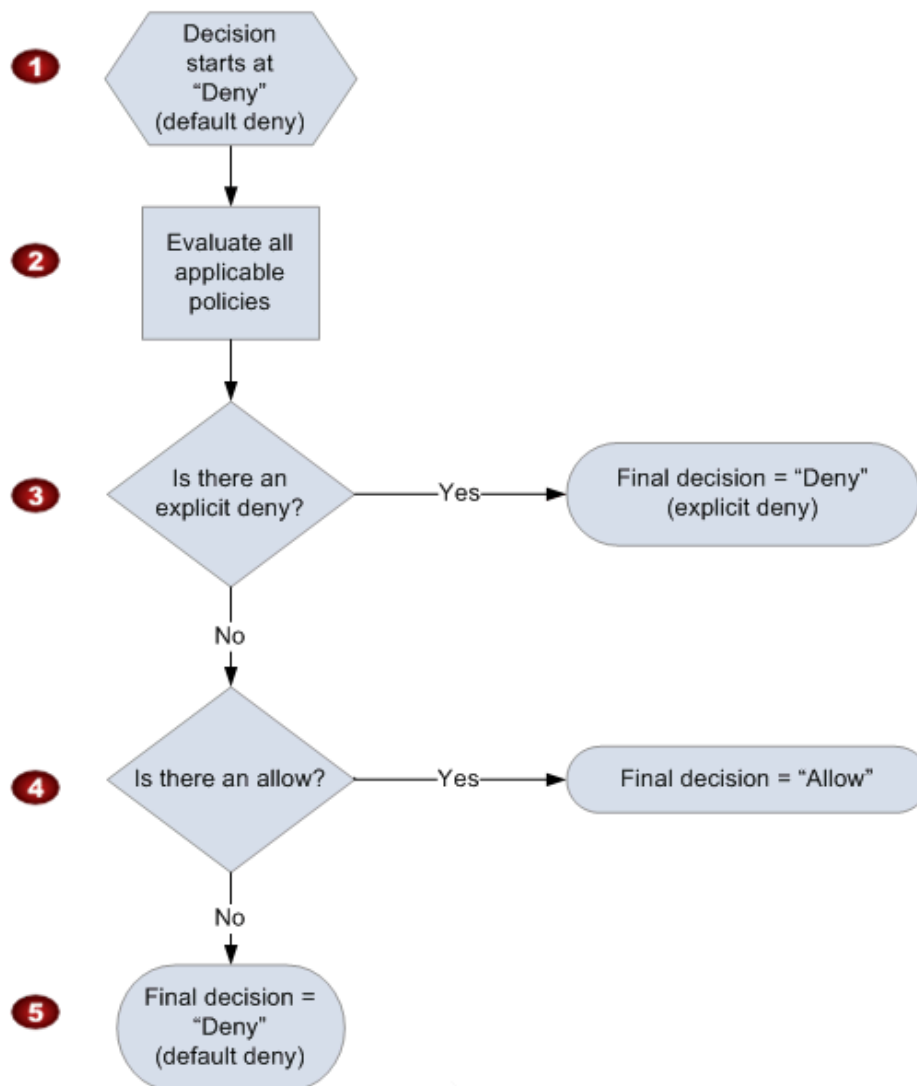
- [Architectural Overview \(p. 19\)](#)

## 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.
---	--

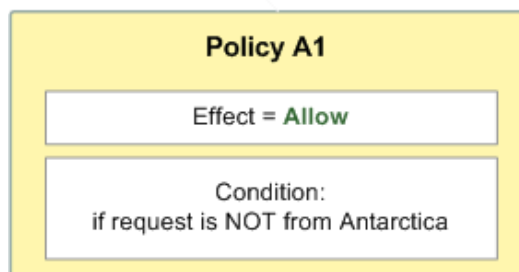
<b>2</b>	The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.
<b>3</b>	In all those policies, the enforcement code looks for 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 <a href="#">Explicit Deny (p. 19)</a> ).
<b>4</b>	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).
<b>5</b>	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a <i>default deny</i> (for more information, see <a href="#">Default Deny (p. 19)</a> )).

## The Interplay of Explicit and Default Denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if a user requests to use Amazon SNS, but the policy on the topic doesn't refer to the user's AWS account at all, then that policy results in a default deny.

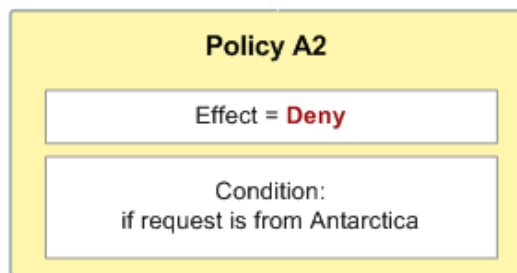
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met,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

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



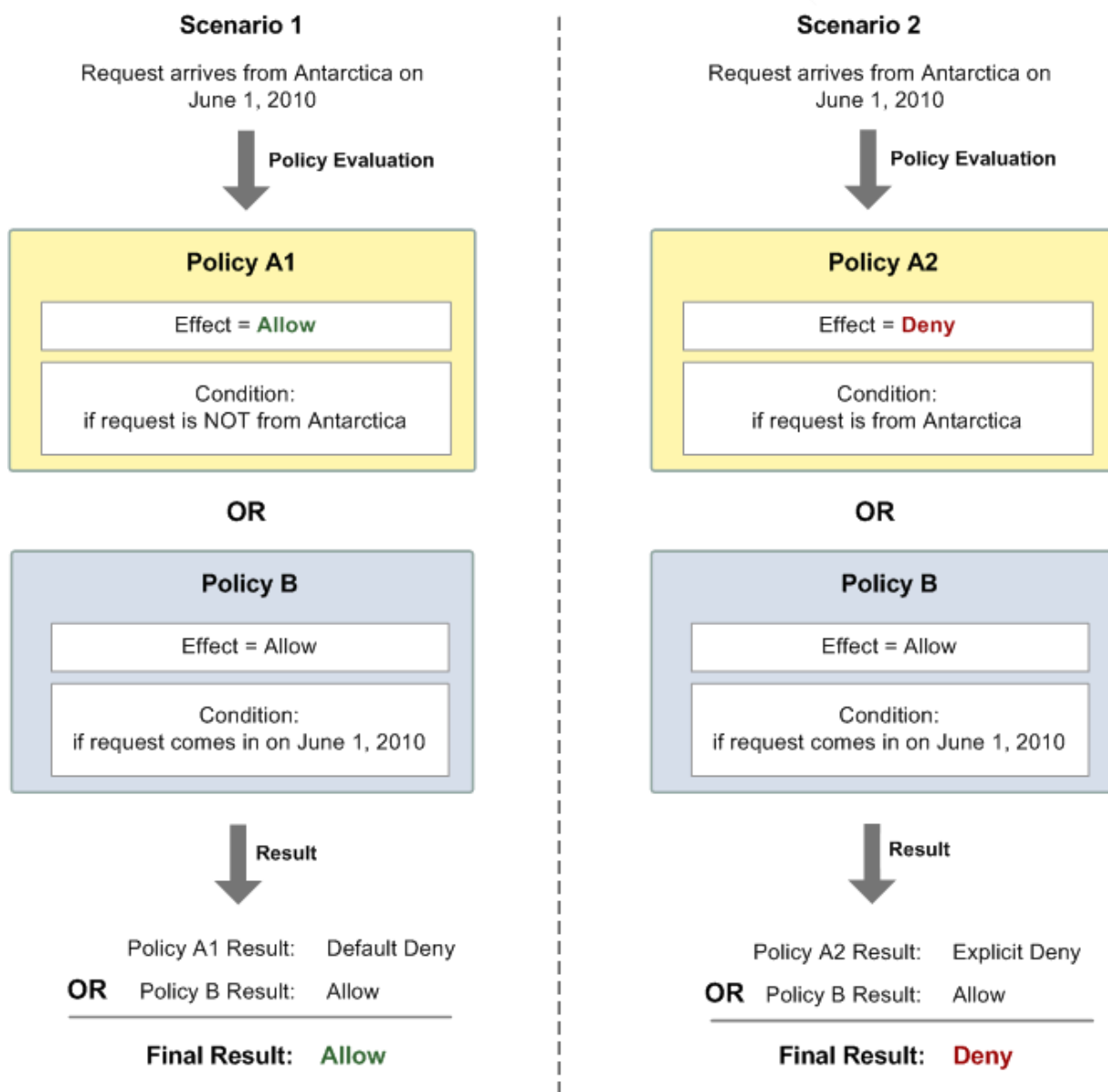
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.



In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy 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 \(p. 26\)](#)
- [Limiting Subscriptions to HTTPS \(p. 26\)](#)

- [Publishing to an SQS Queue \(p. 27\)](#)
- [Allowing Any AWS Resource to Publish to a Topic \(p. 28\)](#)
- [Allowing an Amazon S3 Bucket to Publish to a Topic \(p. 28\)](#)

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": "2008-10-17",
  "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 `1234-5678-9012` 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 `1234-5678-9012` 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": "2008-10-17",
  "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 SQS Queue

In this use case, you want to publish messages from your topic to your SQS queue. Like Amazon SNS, SQS uses Amazon's access control policy language. To allow Amazon SNS to send messages, you'll need to use the SQS action `SetQueueAttributes` to set a policy on the queue.

Again, you'll need to know how to write your own policy because the SQS `AddPermission` action doesn't create policy statements with conditions.

Note that the example presented below is an SQS policy (controlling access to your queue), not an Amazon SNS policy (controlling access to your topic). The actions are 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": "2008-10-17",
  "Id": "MyQueuePolicy",
  "Statement" : [
    {
      "Sid": "Allow-SNS-SendMessage",
      "Effect": "Allow",
      "Principal" : {
        "AWS": "*"
      },
      "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 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., S3 bucket, EC2 instance, or 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": "2008-10-17",
  "Id": "MyAWSPolicy",
  "Statement" : [
    {
      "Sid": "My-statement-id",
      "Effect": "Allow",
      "Principal" : { "AWS": "*" },
      "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. 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 S3 ARN. You could use a similar condition to restrict the permission to a set of S3 buckets, or even to a specific bucket. In this example, the topic owner is 1111-2222-3333 and the S3 owner is 4444-5555-6666. The example states that any S3 bucket owned by 4444-5555-6666 is allowed to publish to `My-Topic`.

```
{
  "Version": "2008-10-17",
  "Id": "MyAWSPolicy",
```



**Amazon Simple Notification Service Getting Started  
Guide  
Example Cases for Amazon SNS Access Control**

---

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

# How to Write a Policy

## Topics

- [Basic Policy Structure](#) (p. 30)
- [Element Descriptions](#) (p. 30)
- [Supported Data Types](#) (p. 40)

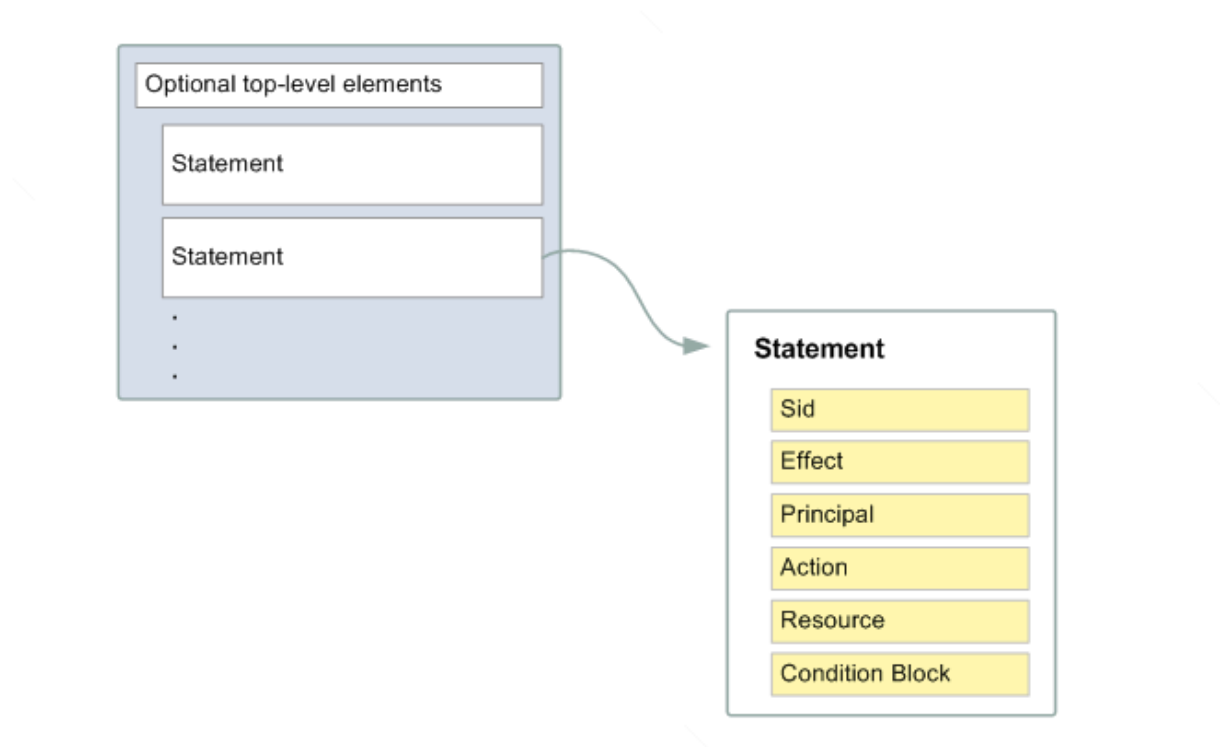
This section describes how to write policies and gives reference information about each policy element.

## Basic Policy Structure

Each policy is a JSON document. As illustrated in the following figure, a policy includes:

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

Each statement includes the core information about a single permission. If a policy includes multiple statements, we apply a logical OR across the statements at evaluation time. If multiple policies are applicable to a request, we apply a logical OR across the policies at evaluation time.



The information in a statement is contained within a series of *elements*. For information about these elements, see [Element Descriptions](#) (p. 30).

## Element Descriptions

### Topics

- [Version](#) (p. 31)

- [Id](#) (p. 31)
- [Statement](#) (p. 31)
- [Sid](#) (p. 32)
- [Effect](#) (p. 32)
- [Principal](#) (p. 32)
- [NotPrincipal](#) (p. 32)
- [Action](#) (p. 33)
- [NotAction](#) (p. 33)
- [Resource](#) (p. 33)
- [NotResource](#) (p. 33)
- [Condition](#) (p. 34)

This section describes the elements you can use in a policy and its statements. The elements are listed here in the general order you use them in a policy. The `Id`, `Version`, and `Statement` are top-level policy elements; the rest are statement-level elements. JSON examples are provided.

All elements are optional for the purposes of parsing the policy document itself. The order of the elements doesn't matter (e.g., the `Resource` element can come before the `Action` element). You're not required to specify any `Conditions` in the policy.

## Version

The `Version` is the access policy language version. This is an optional element, and currently the only allowed value is `2008-10-17`.

```
"Version": "2008-10-17"
```

## Id

The `Id` is an optional identifier for the policy. We recommend you use a UUID for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

### Important

The AWS service (e.g., Amazon SNS) implementing the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see [Special Information for Amazon SNS Policies](#) (p. 41).

```
"Id": "cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

## Statement

The `Statement` is the main element for a statement. It can include multiple elements (see the subsequent sections in this guide).

The `Statement` element contains an array of individual statements. Each individual statement is a distinct JSON block enclosed in curly brackets `{ }`.

```
"Statement": [ { ... }, { ... }, { ... } ]
```

## Sid

The `Sid` (statement ID) is an optional identifier you provide for the policy statement. Essentially it is just a sub-ID of the policy document's ID.

### Important

The AWS service (e.g., Amazon SNS) implementing the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see [Special Information for Amazon SNS Policies \(p. 41\)](#).

```
"Sid" : "1"
```

## Effect

The `Effect` is a required element that indicates whether you want the statement to result in an allow or an explicit deny (for more information, see [Explicit Deny \(p. 19\)](#)).

Valid values for `Effect` are `Allow` and `Deny`.

```
"Effect": "Allow"
```

## Principal

The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (\*) to indicate all possible users. You can view your account ID by logging in to your AWS account at <http://aws.amazon.com> and clicking **Account Activity**.

In JSON, you use `"AWS"` as a prefix for the principal's AWS account ID. In the following example, two principals are included in the statement.

```
"Principal": [
  "AWS": "123456789012",
  "AWS": "999999999999"
]
```

## NotPrincipal

The `NotPrincipal` element is useful if you want to make an exception to a list of principals. You could use this, for example, if you want to prevent all AWS accounts except a certain one. The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (\*) to indicate all possible users. You can view your account ID by logging in to your AWS account at <http://aws.amazon.com> and clicking **Account Activity**.

In JSON, you use `"AWS"` as a prefix for the principal's AWS account ID. In the following example, two principals are included in the statement.

```
"Principal": [
  "AWS": "123456789012",
  "AWS": "999999999999"
]
```

## Action

The `Action` is the specific type or types of access allowed or denied (for example, read or write). You can specify multiple values for this element. The values are free-form but must match values the AWS service expects (for more information, see [Special Information for Amazon SNS Policies \(p. 41\)](#)). You can use a wildcard (\*) to give the principal access to all the actions the specific AWS service lets you share with other developers.

```
"Action": [ "sns:Publish", "sns:Subscribe" ]
```

The prefix and the action name are case insensitive. For example, `sns:Subscribe` is equivalent to `SNS:subscribe`.

## NotAction

The `NotAction` element matches everything except the specified action. This is useful if you want to make an exception to a list of actions being allowed or denied. The example below matches any action, except `Publish`.

```
"NotAction": "sns:Publish"
```

## Resource

The `Resource` is the object or objects the policy covers. You specify the resource using the following *Amazon Resource Name (ARN)* format.

```
arn:aws:<vendor>:<region>:<namespace>:<relative-id>
```

Where:

- `vendor` identifies the AWS product (e.g., Amazon SNS)
- `region` is the Region the resource resides in (e.g., `us-east-1`), if any
- `namespace` is the AWS account ID with no hyphens (e.g., `123456789012`)
- `relative-id` is the portion that identifies the specific resource

## NotResource

The `NotResource` element is useful if you want to make an exception to a list of resources. You could use this, for example, if you want your users to be able to access a specific Amazon SNS topic belonging to the AWS account. If the AWS account were to create a new topic for the company, an admin wouldn't have to update the policy with the new topic's name in order to prevent users from being able to use the topic. By default, the users wouldn't be able to use it.

The following example refers to all resources *other* than your company's topic called `my_corporate_topic`. You would use this in a policy with `"Effect": "Deny"` to keep users from accessing any queue besides `my_corporate_topic`.

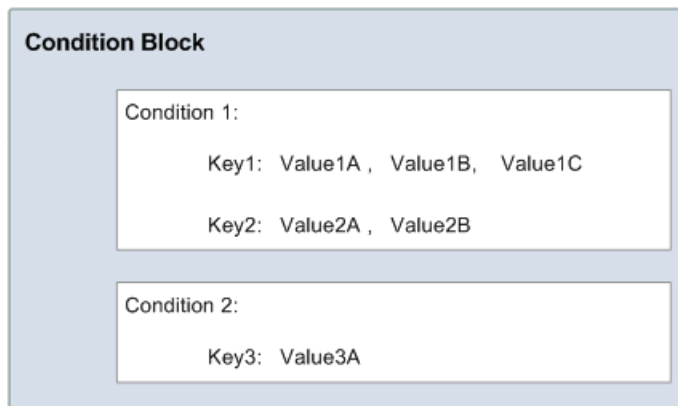
```
"NotResource": "arn:aws:sqs:*:123456789012:my_corporate_topic"
```

## Condition

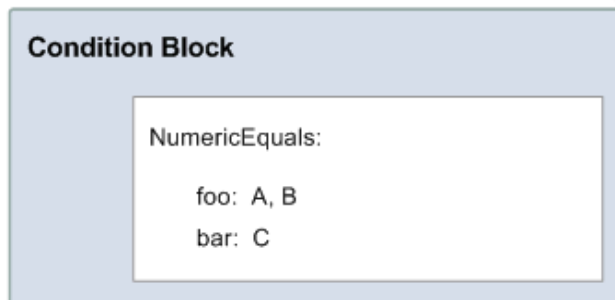
This section describes the `Condition` element and the information you can use inside the element.

### The Condition Block

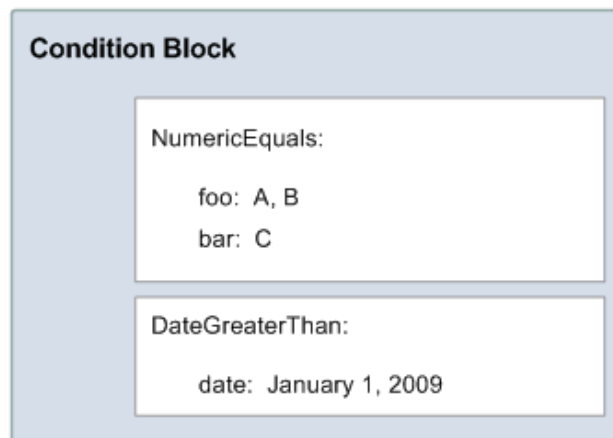
The `Condition` element is the most complex part of the policy statement. We refer to it as the *condition block*, because although it has a single `Condition` element, it can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified for a particular key, all keys can have multiple values.



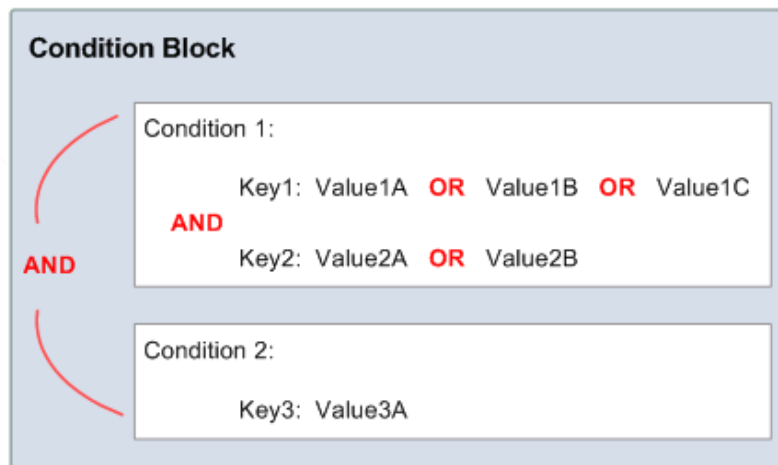
When creating a condition block, you specify the name of each condition, and at least one key-value pair for each condition. AWS defines the conditions and keys you can use (they're listed in the subsequent sections). An example of a condition is `NumericEquals`. Let's say you have a fictional resource, and you want to let John use it only if some particular numeric value *foo* equals either A or B, and another numeric value *bar* equals C. Then you would create a condition block that looks like the following figure.



Let's say you also want to restrict John's access to after January 1, 2009. Then you would add another condition, `DateGreaterThan`, with a date equal to January 1, 2009. The condition block would then look like the following figure.



As illustrated in the following figure, we always apply a logical **AND** to the conditions within a condition block, and to the keys within a condition. We always apply a logical **OR** to the values for a single key. All conditions must be met to return an allow or an explicit deny decision. If a condition isn't met, the result is a default deny.



As mentioned, AWS defines the conditions and keys you can use (for example, one of the keys is `aws:CurrentTime`, which lets you restrict access based on the date and time). The AWS service itself can also define its own service-specific keys. For a list of available keys, see [Available Keys \(p. 36\)](#).

For a concrete example that uses real keys, let's say you want to let John publish to your topic under the following three conditions:

- The time is after 12:00 noon on 8/16/2010
- The time is before 3:00 p.m. on 8/16/2010
- The request comes from an IP address within the 192.168.176.0/24 range or the 192.168.143.0/24 range

Your condition block has three separate conditions, and all three of them must be met for John to have access to your topic.

The following shows what the condition block looks like in your policy.

```
"Condition" : {
  "DateGreaterThan" : {
    "aws:CurrentTime" : "2009-04-16T12:00:00Z"
  },
  "DateLessThan" : {
    "aws:CurrentTime" : "2009-04-16T15:00:00Z"
  },
  "IpAddress" : {
    "aws:SourceIp" : [ "192.168.176.0/24", "192.168.143.0/24" ]
  }
}
```

## Available Keys

AWS provides a set of common keys supported by all AWS services that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`—For date/time conditions (see [Date Conditions \(p. 38\)](#))
- `aws:EpochTime`—The date in epoch or UNIX time, for use with date/time conditions (see [Date Conditions \(p. 38\)](#))
- `aws:MultiFactorAuthAge`—Key that provides a numeric value indicating 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 successfully, this key is not present (see [Existence of Condition Keys \(p. 39\)](#), [Numeric Conditions \(p. 37\)](#) and [Using Multi-Factor Authentication \(MFA\) Devices with AWS](#)).
- `aws:SecureTransport`—Boolean representing whether the request was sent using SSL (see [Boolean Conditions \(p. 38\)](#))
- `aws:SourceArn`—The Amazon Resource Name (ARN) of the source (see [Amazon Resource Name \(ARN\) \(p. 39\)](#))
- `aws:SourceIp`—The requester's IP address, for use with IP address conditions (see [IP Address \(p. 38\)](#))
- `aws:UserAgent`—Information about the requester's client application, for use with string conditions (see [String Conditions \(p. 37\)](#))

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

### Note

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of any service-specific keys you can use, see [Special Information for Amazon SNS Policies \(p. 41\)](#).

## Condition Types

These are the general types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- IP address
- Amazon Resource Name (ARN)



- Existence of condition keys

## String Conditions

String conditions let you constrain using string matching rules. The actual data type you use is a string.

Condition	Description
<code>StringEquals</code>	Strict matching Short version: <code>streq</code>
<code>StringNotEquals</code>	Strict negated matching Short version: <code>strneq</code>
<code>StringEqualsIgnoreCase</code>	Strict matching, ignoring case Short version: <code>streqi</code>
<code>StringNotEqualsIgnoreCase</code>	Strict negated matching, ignoring case Short version: <code>strneqi</code>
<code>StringLike</code>	Loose case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: <code>strl</code>
<code>StringNotLike</code>	Negated loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: <code>strnl</code>

## Numeric Conditions

Numeric conditions let you constrain using numeric matching rules. You can use both whole integers or decimal numbers. Fractional or irrational syntax is not supported.

Condition	Description
<code>NumericEquals</code>	Strict matching Short version: <code>numeq</code>
<code>NumericNotEquals</code>	Strict negated matching Short version: <code>numneq</code>
<code>NumericLessThan</code>	"Less than" matching Short version: <code>numlt</code>
<code>NumericLessThanEquals</code>	"Less than or equals" matching Short version: <code>numlteq</code>
<code>NumericGreaterThan</code>	"Greater than" matching Short version: <code>numgt</code>

**Amazon Simple Notification Service Getting Started  
Guide  
Element Descriptions**

---

Condition	Description
NumericGreaterThanEquals	"Greater than or equals" matching Short version: numgteq

### Date Conditions

Date conditions let you constrain using date and time matching rules. You must specify all date/time values with one of the W3C implementations of the ISO 8601 date formats (for more information, go to <http://www.w3.org/TR/NOTE-datetime>). You use these conditions with the `aws:CurrentTime` key or the `aws:EpochTime` key to restrict access based on request time.

**Note**

Wildcards are not permitted for date conditions.

Condition	Description
DateEquals	Strict matching Short version: dateeq
DateNotEquals	Strict negated matching Short version: dateneq
DateLessThan	A point in time at which a key stops taking effect Short version: dateelt
DateLessThanEquals	A point in time at which a key stops taking effect Short version: dateelteq
DateGreaterThan	A point in time at which a key starts taking effect Short version: dategt
DateGreaterThanEquals	A point in time at which a key starts taking effect Short version: dategteq

### Boolean Conditions

Condition	Description
Bool	Strict Boolean matching

### IP Address

IP address conditions let you constrain based on IP address matching rules. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 10.52.176.0/24). For more information, go to [RFC 4632](#).

Condition	Description
IpAddress	Approval based on the IP address or range
NotIpAddress	Denial based on the IP address or range

## Amazon Resource Name (ARN)

Amazon Resource Name (ARN) conditions let you constrain based on ARN matching rules. The actual data type you use is a string.

Condition	Description
ArnEquals	Strict matching for ARN
ArnNotEquals	Strict negated matching for ARN
ArnLike	Loose case-insensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include a multi-character match wildcard (*) or a single-character match wildcard (?).
ArnNotLike	Negated loose case-insensitive matching of the ARN. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

Following is an example of the kind of policy you need to attach to any queue that you want Amazon SNS to send messages to. It gives Amazon SNS permission to send messages to the queue (or queues) of your choice, but only if the service is sending the messages on behalf of a particular Amazon SNS topic (or topics). You specify the queue in the `Resource` field, and the Amazon SNS topic as the value for the `SourceArn` key.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": "210987654321"
    },
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:01234567891:your_queue_xyz",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:your_special_top
ic_1"
      }
    }
  }]
}
```

## Existence of Condition Keys

Use a `Null` condition to check if a condition key is present at the time of authorization. In the policy statement, use either `true` (the key doesn't exist) or `false` (the key exists and its value is not null). You can use this condition to determine if a user has authenticated with MFA (Multi-Factor Authentication). For example, the following condition states that MFA authentication must exist (be *not null*) for the user to use the Amazon EC2 API.

```
{
  "Statement": [{
    "Action": ["ec2:*"],
    "Effect": "Allow",
    "Resource": ["*"],
    "Condition": {
      "Null": {
        "aws:MultiFactorAuthPresent": true
      }
    }
  }]
}
```

```
    "Condition": {  
      "Null": { "aws:MultiFactorAuthAge": "false" }  
    }  
  ]  
}
```

## Supported Data Types

This section lists the set of data types the access policy language supports. The language doesn't support all types for each policy element (for the supported data types for each element, see [Element Descriptions](#) (p. 30)).

The access policy language supports the following data types:

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to [RFC 4627](#).

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the <a href="#">W3C Profile of ISO 8601</a>
IpAddress	String adhering to <a href="#">RFC 4632</a>
List	Array
Object	Object

## 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 or queue (when writing a policy, don't include statements that cover different topics or queues)
- 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	20 kb
Statements	20
Principals	20
Resource	1 (its 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	This grants permission to add permissions to the topic policy.
sns>DeleteTopic	This grants permission to delete a topic.
sns:GetTopicAttributes	This grants permission to receive all of the topic attributes.
sns:ListSubscriptionsByTopic	This grants permission to retrieve all the subscriptions to a specific topic.
sns:Publish	This grants permission to publish to a topic.
sns:RemovePermission	This grants permission to remove any permissions in the topic policy.
sns:SetTopicAttributes	This grants permission to set a topic's attributes.
sns:Subscribe	This grants permission to subscribe to a topic.

### Note

When you grant other AWS accounts access to your AWS resources, be aware that all AWS accounts can delegate their permissions to users under their accounts. This is known as cross-account access. Cross-account access enables you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, go to [Enabling Cross-Account Access](#) in *Using AWS Identity and Access Management*.

## Amazon SNS Keys

Amazon SNS uses the following service-specific keys. You can use these in policies that restrict access to `Subscribe` requests (like the one in [Limiting Subscriptions to HTTPS \(p. 26\)](#)).

- **sns:Endpoint**—The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see [String Conditions \(p. 37\)](#)) 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 [String Conditions \(p. 37\)](#)) 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 SNS Policies Together \(p. 42\)](#)
- [Amazon SNS ARNs \(p. 45\)](#)
- [Amazon SNS Actions \(p. 46\)](#)
- [Amazon SNS Keys \(p. 46\)](#)
- [Example Policies for Amazon SNS \(p. 47\)](#)
- [Using Temporary Security Credentials \(p. 49\)](#)

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 create a policy that gives certain users in your organization permission to use the `Publish` action with specific topics in your AWS account.

### 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 \(p. 47\)](#).

## IAM and 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.

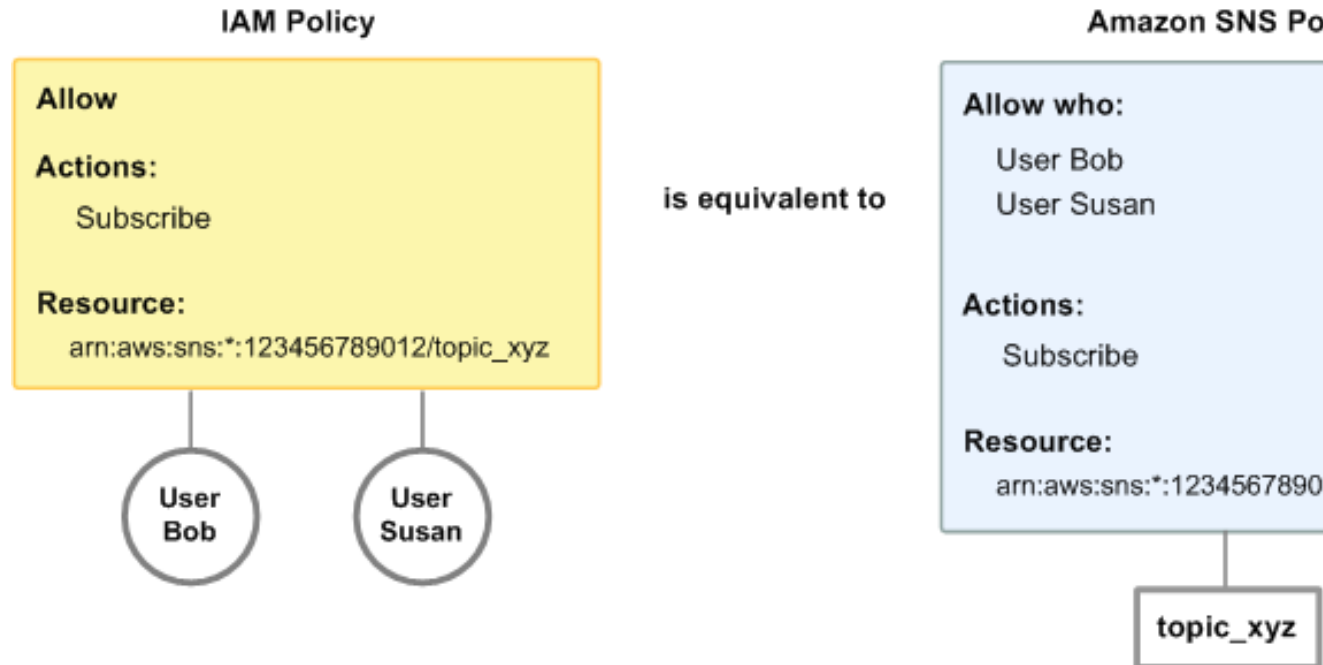
To give your users permissions for your Amazon SNS topics, you can use IAM policies, Amazon SNS policies, or both. For the most part, you can achieve the same results with either. For example, the following diagram shows an IAM policy and an Amazon SNS policy that are equivalent. The IAM policy allows the Amazon SNS `Subscribe` action for the topic called `topic_xyz` in your AWS account. The IAM

## Amazon Simple Notification Service Getting Started Guide

### IAM and SNS Policies Together

---

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.

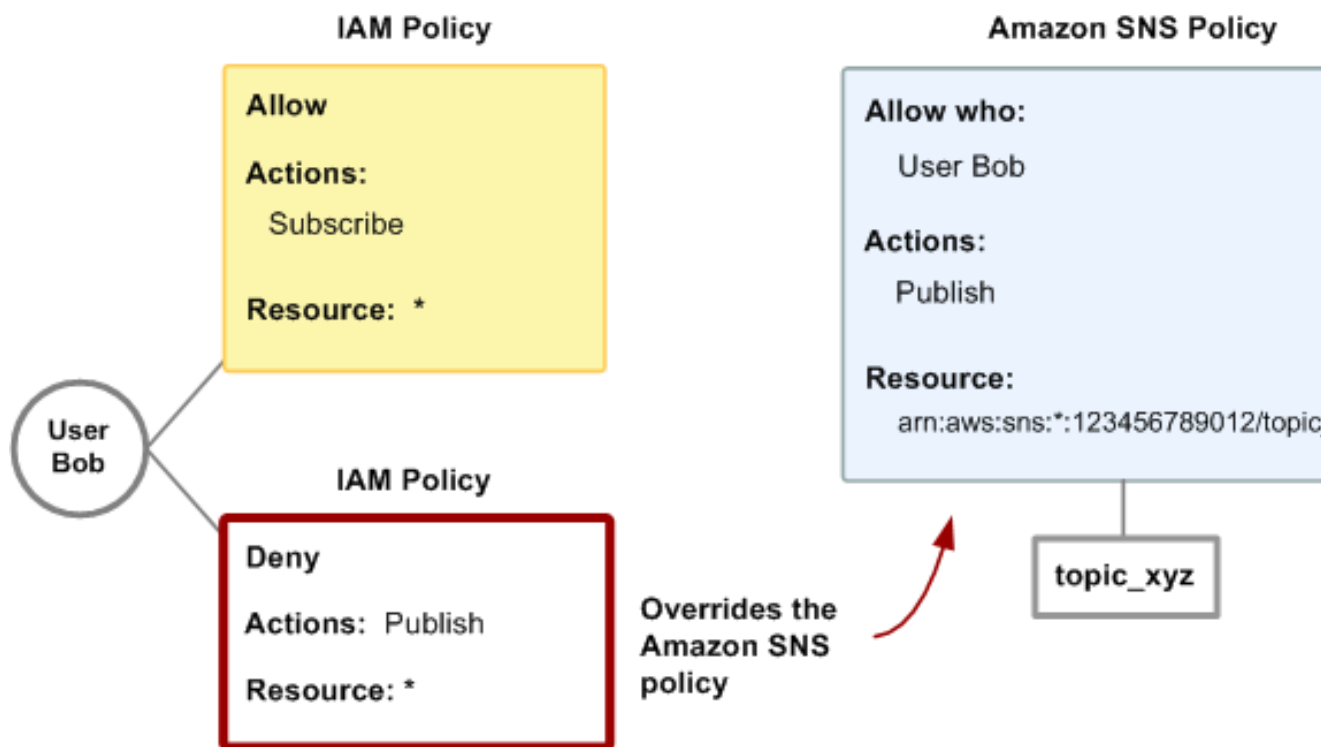


If Bob were to send a request to subscribe to any topic in the AWS account, the IAM policy would allow the action. If Bob were to send a request to publish a message to `topic_xyz`, the Amazon SNS policy would allow the action.



## 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 \(p. 22\)](#)). The following diagram illustrates the concept.



For examples of policies that cover Amazon SNS actions and resources, see [Example Policies for Amazon SNS \(p. 47\)](#). For more information about writing 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 *Using AWS Identity and Access Management*.

### 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. For more information about policy keys, see [Condition \(p. 34\)](#).

### AWS-Wide Policy Keys

- `aws:CurrentTime` (for date/time conditions)
- `aws:EpochTime` (the date in epoch or UNIX time, for use with date/time conditions)
- `aws:SecureTransport` (Boolean representing whether the request was sent using SSL)
- `aws:SourceIp` (the requester's IP address, for use with IP address conditions)
- `aws:UserAgent` (information about the requester's client application, for use with string conditions)

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

For services that use only SSL, such as Amazon RDS and Amazon Route 53, the `aws:SecureTransport` key has no meaning.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

## 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 [String Conditions \(p. 37\)](#)) 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 [String Conditions \(p. 37\)](#)) 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`.

```
{
  "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.

```
{
  "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.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:Subscribe"],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "sns:Protocol": "email",
        "sns:Endpoint": ".*@yourcompany.com"
      }
    }
  }]
}
```

### 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 SNS policy. However, anyone in the partner's company who possesses the AWS account 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 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.

```
{
  "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.

```
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  
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
```

## Appendix B: Sending and Receiving SMS Notifications Using Amazon SNS

---

You can use Amazon Simple Notification Service (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 upper case characters for SMS messages.

You can use SMS notifications in conjunction with other notification types, such as email. For example, if you use Amazon 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 8 KiB 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 8 KiB 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 Region. However, you can publish messages to topics that you create in the US East 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 [Sign Up for Amazon SNS \(p. 2\)](#).
- **Create an Amazon SNS topic**—Create an Amazon SNS topic if you don't have one.  
For more information, see [Create a Topic \(p. 3\)](#).

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**

<a href="#">Task 1: Assign a Topic Display Name (p. 51)</a>
<a href="#">Task 2: Subscribe to a Topic Using the SMS Protocol (p. 53)</a>
<a href="#">Task 3: Publish a Message (p. 55)</a>
<a href="#">Task 4: Cancel SMS Subscriptions (p. 57)</a>

## Task 1: Assign a Topic Display Name

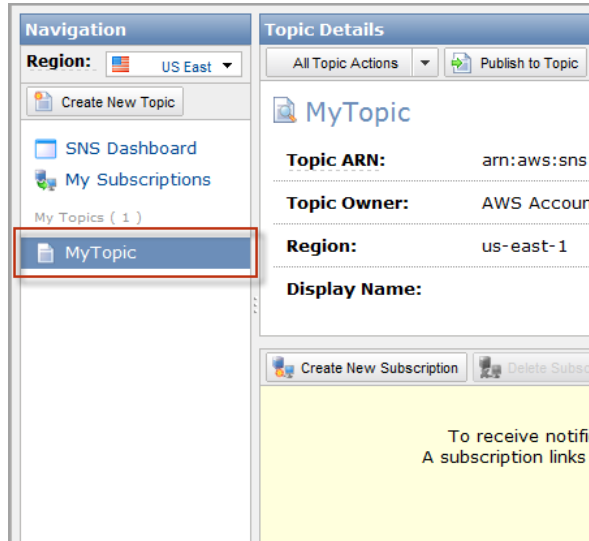
To publish SMS messages for a topic, you must assign the topic a display name.

**To assign a display name to a topic**

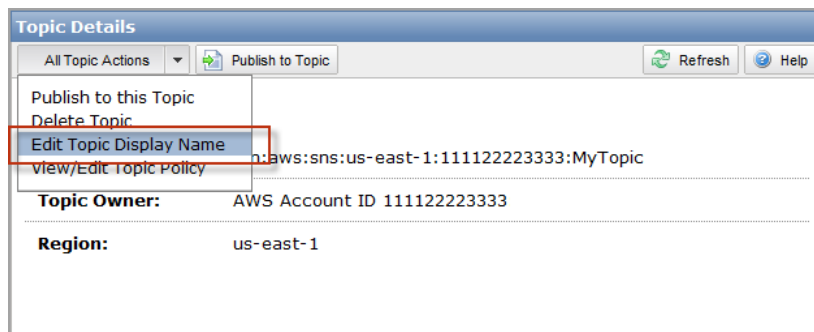
1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select a topic from the Navigation pane.  
The examples that follow use the topic name *MyTopic*.

# Amazon Simple Notification Service Getting Started Guide

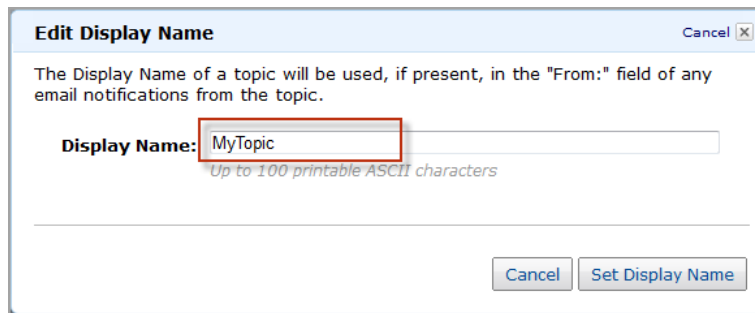
## Task 1: Assign a Topic Display Name



3. Select **Edit Topic Display Name** from the **All Topic Actions** drop-down list.



4. Enter a display name in the **Display Name** box and click **Set Display Name**.

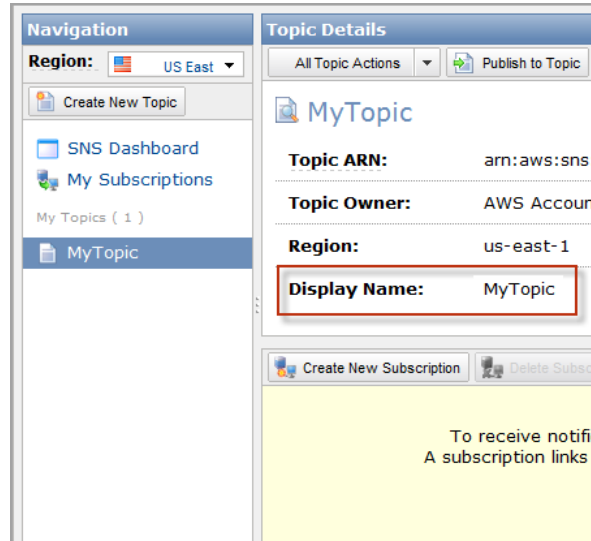


The new topic display name appears in the **Topic Details** page.



Amazon Simple Notification Service Getting Started  
Guide  
Task 2: Subscribe to a Topic Using the SMS Protocol

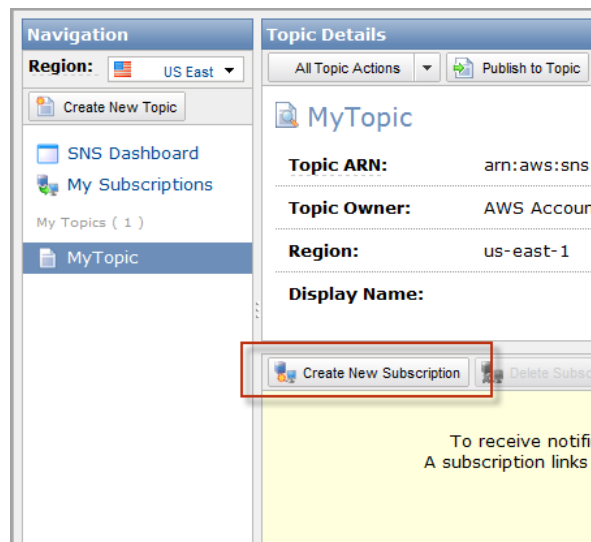
---



## Task 2: Subscribe to a Topic Using the SMS Protocol

To subscribe to an Amazon SNS topic using the SMS protocol

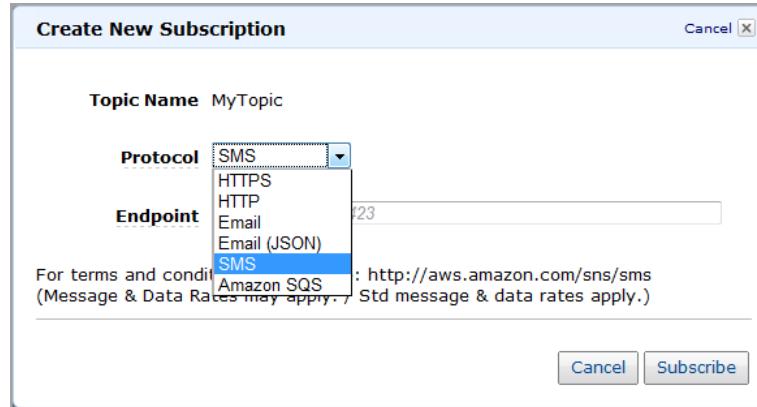
1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select a topic in the **Navigation** pane, and in the **Topic Details** pane click **Create New Subscription**.



3. Select **SMS** from the **Protocol** drop-down list.

**Amazon Simple Notification Service Getting Started  
Guide**  
**Task 2: Subscribe to a Topic Using the SMS Protocol**

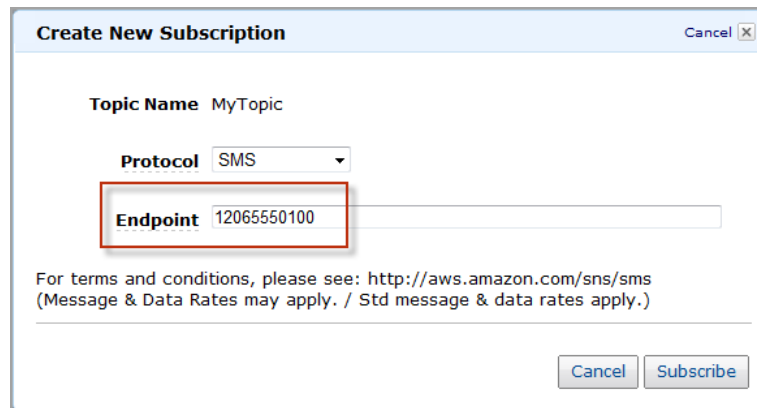
---



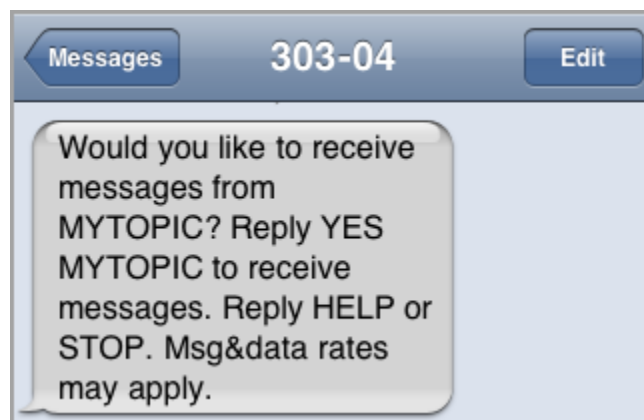
4. Enter the phone number of an SMS-enabled device in the **Endpoint** box and click **Subscribe**.

**Note**

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



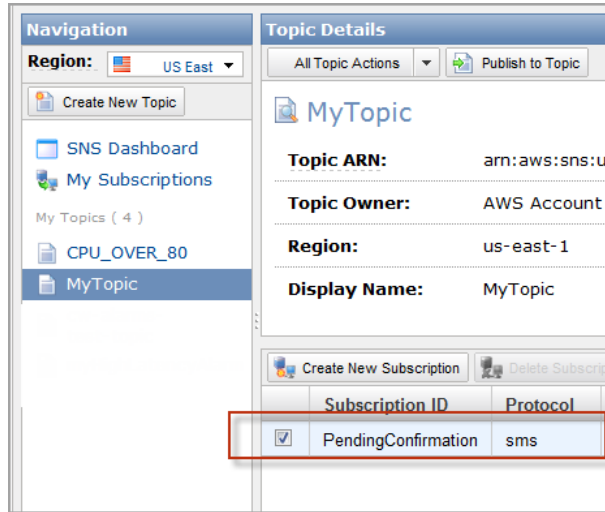
Amazon SNS sends a confirmation text message to the SMS-enabled device associated with the number you entered.



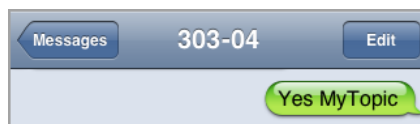
In the AWS Management Console the subscription is listed as **PendingConfirmation** until the SMS-enabled device confirms the subscription.

# Amazon Simple Notification Service Getting Started Guide

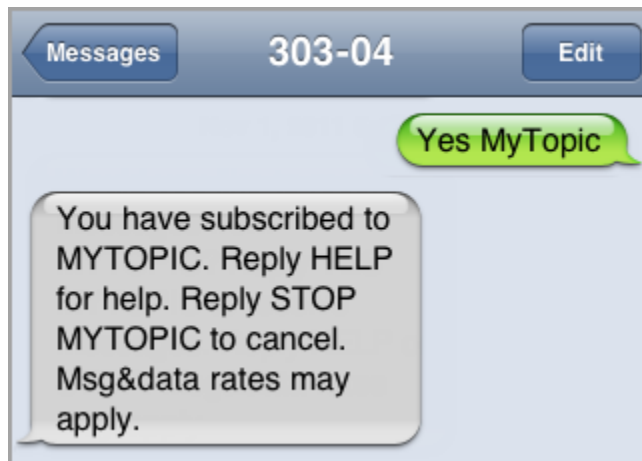
## Task 3: Publish a Message



5. 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. Select a topic from the **Navigation** pane.

## Amazon Simple Notification Service Getting Started Guide

### Task 3: Publish a Message

---

3. Click **Publish to Topic**.

4. Enter text in the **Message** box.

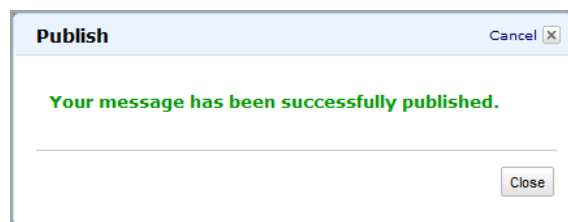
Amazon SNS sends text that you enter in the **Message** box to SMS subscribers unless you also enter text into the **Subject** box. The following example creates a message with the text `Hello World!`



The screenshot shows the 'Publish' dialog box in the Amazon SNS console. The 'Topic Name' is 'MyTopic'. The 'Subject' field is empty, with a hint 'Up to 100 printable ASCII characters (optional)'. The 'Message' field contains 'Hello World!' and has a hint 'Up to 64KB of Unicode text'. There are two radio buttons: 'Use same message body for all protocols' (selected) and 'Use different message body for different protocols'. A note at the bottom states: 'For SMS notifications, it is best to leave the Subject field blank and place your text in the Message field to send a maximum of 140 characters. If the Subject field is not blank, the text in the Subject field will be used as content for the SMS messages.' At the bottom right are 'Cancel' and 'Publish Message' buttons.

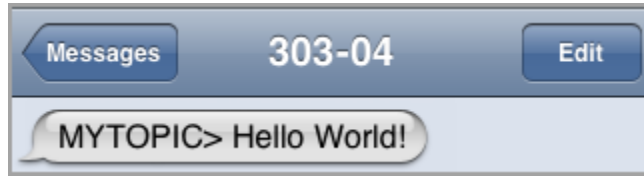
5. Enter text in the **Subject** box if you want to use the **Message** box for messages to email subscribers. If you include text in the **Subject** box, the SMS message will contain the subject text rather than the text from the **Message** box. Any email subscribers, however, will receive both the 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.
6. Click **Publish Message**.

Amazon SNS displays a confirmation dialog box.



The screenshot shows a confirmation dialog box titled 'Publish'. It contains the message 'Your message has been successfully published.' in green text. At the bottom right is a 'Close' button.

The SMS message appears on your SMS-enabled device.

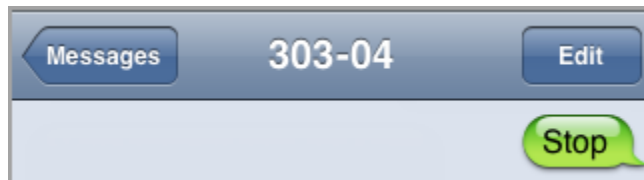


## Task 4: Cancel SMS Subscriptions

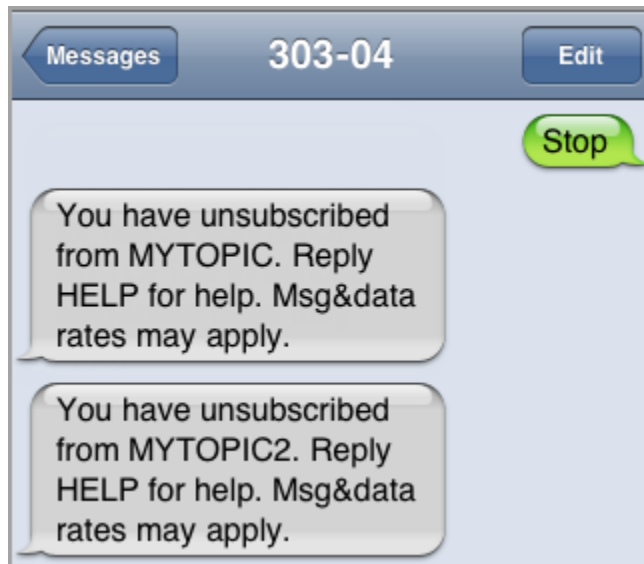
You have several options for canceling SMS subscriptions to a topic. You can stop receiving all SMS messages by replying STOP or QUIT to short code 30304. To cancel your subscription to a specific topic, send an SMS message that contains STOP <TOPICNAME> to short code 30304, where <TOPICNAME> is the 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.

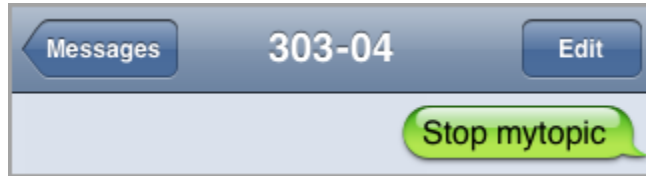


Amazon SNS responds with confirmation messages for each topic.

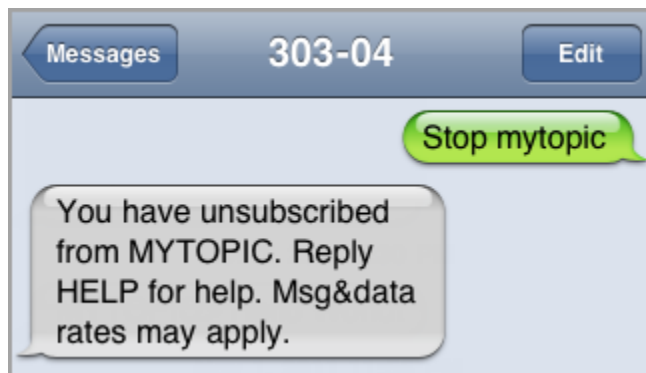


### To stop receiving SMS messages from a specific topic

- 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.



## Appendix C: Sending Amazon SNS Messages to Amazon SQS Queues

---

You can use Amazon Simple Notification Service (SNS) to send messages to one or more Amazon SQS queues. When you subscribe a queue to a topic, you can publish a message to the topic and Amazon SNS sends an SQS message to the subscribed queue. The SQS message contains the subject and message that were published to the topic along with metadata about the message in a JSON document. The SQS message will look similar to the following JSON document.

```
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "TopicArn" : "arn:aws:sns:us-east-1:123456789012:MyTopic",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "Vd4ZCFQnTrFPa37tnVO0FF9Iau3MGzjlJLRfySEoWz4uZHSj6ycK4ph7lZm
dv0NtJ4dC/El9FOGp3VuvchpaTraNHWhhQ/OsNlHVz20zxmF9b88R8GtqjfkB5woZZmz87HiM6CY
DT03l7LMwFT4VU7ELtyaBBafhPTg9O5CnKkg=",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&Sub
scriptionArn=arn:aws:sns:us-east-1:123456789012: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 to.](#) (p. 60)
2. [Give `sqs:SendMessage` permission to the SNS topic so that it can send messages to the queue.](#) (p. 61)

3. [Subscribe the queue to the SNS topic.](#) (p. 62)
4. [Give IAM users or AWS accounts the appropriate permissions to publish to the SNS topic and read messages from the SQS queue.](#) (p. 63)
5. [Test it out by publishing a message to the topic and reading the message from the queue.](#) (p. 65)

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. 66).

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. 69).

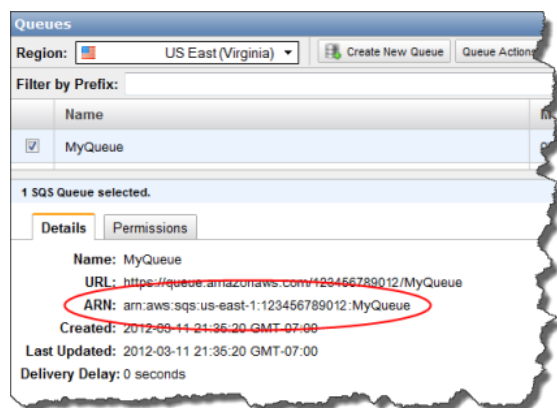
## Step 1. Get the ARN of the queue and the topic.

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.

### 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 SNS topic.



To get the topic ARN, you can use the Amazon SNS console, the [sns-get-topic-attributes](#) command, or the [GetQueueAttributes](#) API.

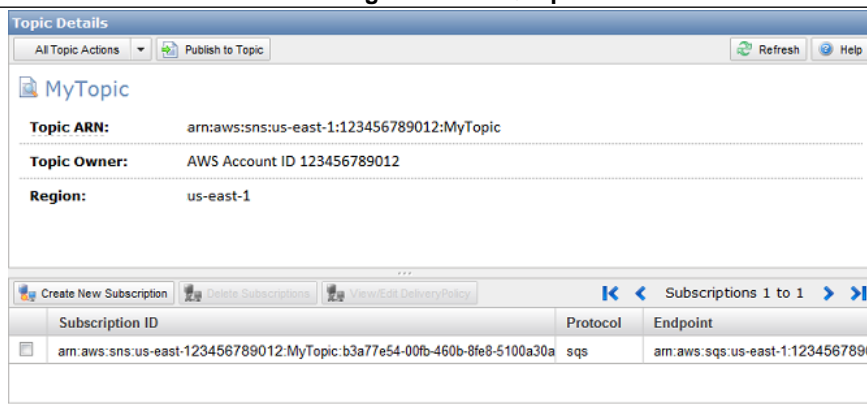
### To get the topic ARN from the Amazon SNS console

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 SNS topic to send messages to the queue.



## Amazon Simple Notification Service Getting Started Guide

### Step 2. Give permission to the SNS topic to send messages to the SQS queue.



## Step 2. Give permission to the SNS topic to send messages to the SQS queue.

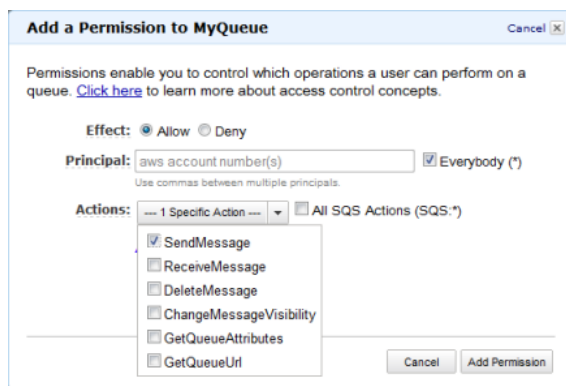
For an SNS topic to be able to send messages to a queue, you must set a policy on the queue that allows the SNS topic to perform the `sqs:SendMessage` action.

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 SNS Getting Started Guide*. For more information, see [Creating a Queue](#) in the *Amazon SQS Getting Started Guide*.

To set a policy on a queue, you can use the Amazon SQS console or the [SetQueueAttributes](#) API. 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.
3. In the **Add a Permission** dialog box, select **Allow** for **Effect**, select **Everybody (\*)** for **Principal**, and select **SendMessage** from the **Actions** drop-down.

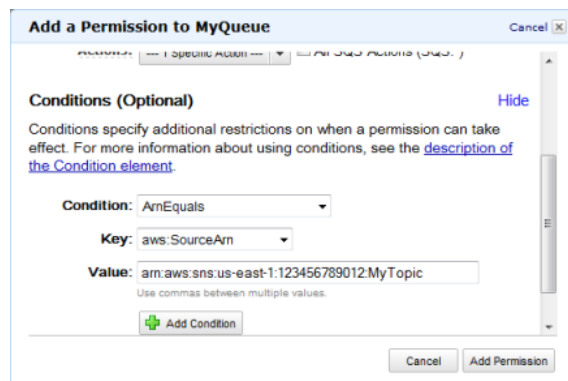


4. Add a condition that allows the action for the topic. Click **Add Conditions (optional)**, select **ArnEquals** for **Condition**, select **aws:SourceArn** for **Key**, and paste in the topic ARN for **Value**. Click **Add**

## Amazon Simple Notification Service Getting Started Guide

### Step 3. Subscribe the queue to the SNS topic.

**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.

```
{
  "Statement": [
    {
      "Sid": "MySQSPolicy001",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "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 SNS topic.

To send messages to a queue through a topic, you must subscribe the queue to the 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. Before you start, make sure you have the ARN for the queue that you want to subscribe.

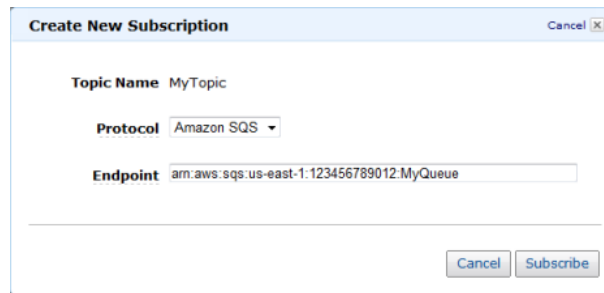
### To subscribe a queue to a topic using the Amazon SNS console

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select the topic from the **Navigation** pane.

## Amazon Simple Notification Service Getting Started Guide

### Step 4. Give users permissions to the appropriate topic and queue actions.

3. Click **Create New 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**.



The screenshot shows a 'Create New Subscription' dialog box. It has a title bar with 'Create New Subscription' and a 'Cancel' button. Inside, there are three fields: 'Topic Name' with the value 'MyTopic', 'Protocol' with a dropdown menu showing 'Amazon SQS', and 'Endpoint' with the value 'arn:aws:sqs:us-east-1:123456789012:MyQueue'. At the bottom right, there are two buttons: 'Cancel' and '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. 66).

## 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 SNS topic and to read/delete messages from the 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 SNS 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. 64). 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. 64). If you want to give permissions to a topic or queue to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

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

## Adding a policy to an IAM user or group

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

```
{
  "Statement": [{
    "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`.

```
{
  "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 `111122223333` permission to perform the `sns:Publish` action on that topic.

```
{
  "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": "2008-10-17",
  "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.

**Amazon Simple Notification Service Getting Started  
Guide**  
**Sending Amazon SNS messages to an Amazon SQS  
queue in a different account**

---

**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-east-1:123456789012:MyTopic",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "Vd4ZCFQnTrFPa37tnVO0FF9Iau3MGzjlJLRfySEoWz4uZHSj6ycK4ph71Zm
dv0NtJ4dC/El9FOGp3VuvchpaTraNHWhhQ/OsN1HVz20zxmF9b88R8GtqjKb5woZZmz87HiM6CY
DTo3l7LMwFT4VU7ELtyaBBafhPTg9O5CnKkg=",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotification
Service-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsub
scribe&SubscriptionArn=arn:aws:sns:us-east-1:123456789012: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 [Appendix C: Sending Amazon SNS Messages to Amazon SQS Queues \(p. 59\)](#)). 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 \(p. 66\)](#)
- [User Who Does Not Own the Queue Creates Subscription \(p. 68\)](#)

## 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

## Amazon Simple Notification Service Getting Started Guide

### Queue Owner Creates Subscription

---

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`.

```
{
  "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 SNS console with credentials for account `111122223333` to subscribe to the topic.

#### To add an 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 SNS topic to send messages to the SQS queue.](#) (p. 61).
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 root AWS account `111122223333` to issue the `sns-subscribe` command or call the [Subscribe](#) API 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.

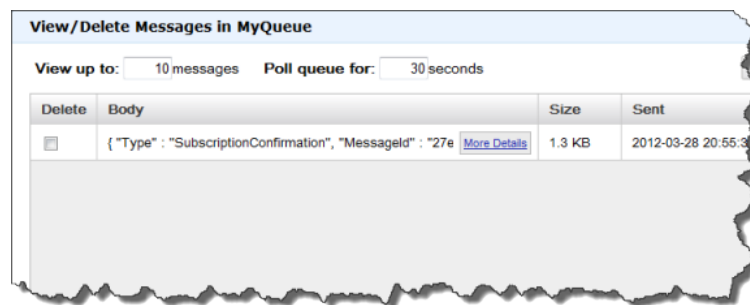
### Important

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 SNS topic to send messages to the SQS queue.](#) (p. 61).

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.

### 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**.



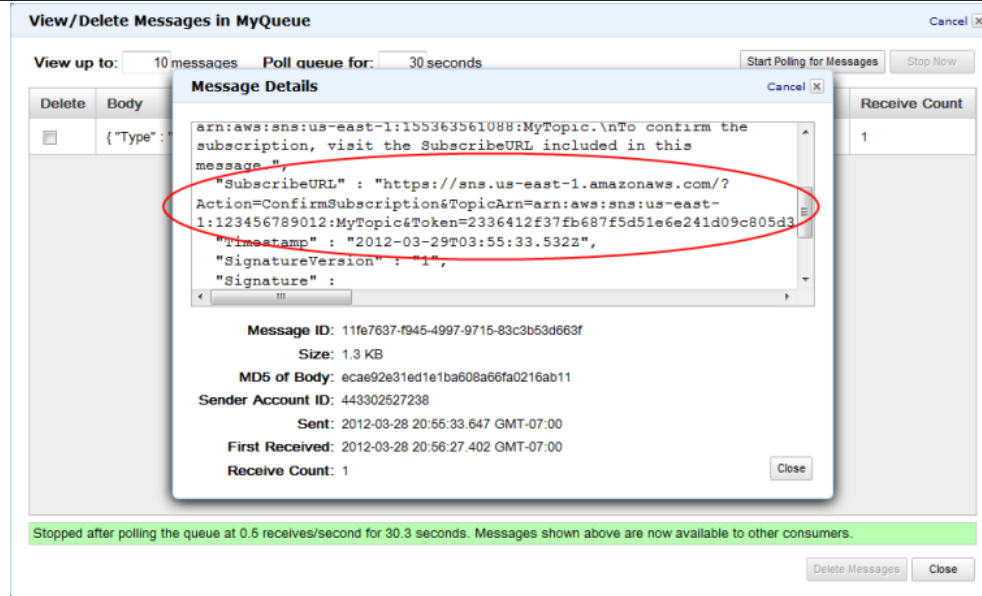
5. In the text box, find the `SubscribeURL` value and copy the URL. The URL will look similar to the following URL.

```
https://sns.us-east-1.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-1:123456789012:MyTopic&SubscriptionId=27e23136f5621885f488cf02f881c72641e5210451a56390095148367024344f27e990278320345f822720902
```



## Amazon Simple Notification Service Getting Started Guide

### Using an AWS CloudFormation Template to Create a Topic that Sends Messages to Amazon SQS Queues



6. 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-east-1:123456789012: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 SNS topic that can send messages to two 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 SQS queues (MyQueue1 and MyQueue2). MyPublishTopicGroup is an IAM group whose members have permission to publish to MySNSTopic using the [Publish](#) API 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 that 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 SQS queues using the [ReceiveMessage](#) and [DeleteMessage](#) APIs. AddUserToMyQueueGroup adds MyQueueUser to the MyRDMessageQueueGroup so that that 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 SNS topic that can send messages
to two SQS queues with appropriate permissions for one IAM user to publish to
the topic and another to read messages from the queues. MySNSTopic is set up
to publish to two subscribed endpoints, which are two SQS queues (MyQueue1 and
MyQueue2). MyPublishUser is an IAM user that can publish to MySNSTopic using
the Publish API. MyTopicPolicy assigns that permission to MyPublishUser.
MyQueueUser is an IAM user that can read messages from the two SQS queues.
MyQueuePolicy assigns those permissions to MyQueueUser. It also assigns permis
sion 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",
```

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

```

        "MinLength": "1",
        "MaxLength": "41",
        "AllowedPattern" : "[a-zA-Z0-9]*",
        "ConstraintDescription" : "must contain only alphanumeric characters."
    }
},

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

```

**Amazon Simple Notification Service Getting Started  
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": { "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "sqs:DeleteMessage",
              "sqs:ReceiveMessage"
            ],
            "Resource": [
              { "Fn::GetAtt" : [ "MyQueue1", "Arn" ] },
              { "Fn::GetAtt" : [ "MyQueue2", "Arn" ] }
            ]
          }
        ]
      }
    ]
  }
},
},
"AddUserToMyQueueGroup" : {
  "Type" : "AWS::IAM::UserToGroupAddition",
  "Properties" : {
    "GroupName": { "Ref" : "MyRDMessageQueueGroup" },
    "Users" : [{ "Ref" : "MyQueueUser" }]
  }
},
"MyQueuePolicy" : {
  "Type" : "AWS::SQS::QueuePolicy",

```

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

```
"Properties" : {
  "PolicyDocument": {
    "Id": "MyQueuePolicy",
    "Statement" : [
      {
        "Sid": "Allow-SendMessage-To-Both-Queues-From-SNS-Topic",
        "Effect": "Allow",
        "Principal" : { "AWS" : "*" },
        "Action": [ "sqs:SendMessage" ],
        "Resource": "*",
        "Condition": {
          "ArnEquals": {
            "aws:SourceArn": { "Ref" : "MySNSTopic" }
          }
        }
      }
    ]
  },
  "Queues" : [ { "Ref" : "MyQueue1" }, { "Ref" : "MyQueue2" } ]
},
"Outputs" : {
  "MySNSTopicTopicARN" : {
    "Value" : { "Ref" : "MySNSTopic" }
  },
  "MyQueue1Info" : {
    "Value" : { "Fn::Join" : [
      " ",
      [
        "ARN:",
        { "Fn::GetAtt" : [ "MyQueue1", "Arn" ] },
        "URL:",
        { "Ref" : "MyQueue1" }
      ]
    ] }
  },
  "MyQueue2Info" : {
    "Value" : { "Fn::Join" : [
      " ",
      [
        "ARN:",
        { "Fn::GetAtt" : [ "MyQueue2", "Arn" ] },
        "URL:",
        { "Ref" : "MyQueue2" }
      ]
    ] }
  },
  "MyPublishUserInfo" : {
    "Value" : { "Fn::Join" : [
      " ",
      [
        "ARN:",
        { "Fn::GetAtt" : [ "MyPublishUser", "Arn" ] },
        "Access Key:",
        { "Ref" : "MyPublishUserKey" },
        "Secret Key:",
        { "Fn::GetAtt" : [ "MyPublishUserKey", "SecretAccessKey" ] }
      ]
    ] }
  }
}
```

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

---

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

## Appendix D: 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. 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 \(p. 108\)](#) and [HTTP/HTTPS Notification JSON Format \(p. 111\)](#).

```
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-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 761
Content-Type: text/plain; charset=UTF-8
Host: ec2-50-17-44-49.compute-1.amazonaws.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "da41e39f-ea4d-435a-b922-c6aae3915ebe",
  "TopicArn" : "arn:aws:sns:us-east-1:123456789012:MyTopic",
  "Subject" : "test",
  "Message" : "test message",
  "Timestamp" : "2012-04-25T21:49:25.719Z",
  "SignatureVersion" : "1",
  "Signature" : "R3j/tNolDMXvB8r9R83tGoNn0ecwd5UjllzsvS
vbItzfamPn2nk5HVS7XnOn/49IkxDKz8Yr1H2qJXj2iZB0Zo2071c4qQk1fMUDI3LG
pij7RCW7AW9vYYsSqIKRnFS94ilu7NFhUzLiieYr4BKHpdtmdD6c0esKEYBpabxDSc=",
```

## Amazon Simple Notification Service Getting Started Guide

### Step 1: Make sure your endpoint is ready to process Amazon SNS messages

```
"SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
"UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfc21c8f55"
}
```

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

[Step 1: Make sure your endpoint is ready to process Amazon SNS messages \(p. 76\)](#)

[Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic \(p. 79\)](#)

[Step 3: Confirm the subscription \(p. 80\)](#)

[Step 4: Set the delivery retry policy for the subscription \(Optional\) \(p. 80\)](#)

[Step 5: Give users permissions to publish to the topic \(Optional\) \(p. 80\)](#)

[Step 6: Send messages to the HTTP/HTTPS endpoint \(p. 82\)](#)

To see an AWS CloudFormation template that creates a topic that sends messages to an HTTP endpoint, see [Using an AWS CloudFormation Template to Set Up a Topic and an HTTP Endpoint \(p. 100\)](#).

## 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 a subscription confirmation request to the endpoint. 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. After the subscription is confirmed, Amazon SNS will send notifications to the endpoint when a publish action is performed on the subscribed topic.

### To set up your endpoint to process subscription confirmation and notification messages

1. Your code should read the HTTP headers of the HTTP POST requests that Amazon SNS sends to your endpoint. Your code should look for the header field `x-amz-sns-message-type`, which tells you the type of message that Amazon SNS has sent to you. By looking at the header, you can determine the message type without having to parse the body of the HTTP request. There are two types that you need to handle: `SubscriptionConfirmation` and `Notification`. The `UnsubscribeConfirmation` message is used only when the subscription is deleted from the topic.

For details about the HTTP header, see [HTTP/HTTPS Headers \(p. 108\)](#). 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
```



## Amazon Simple Notification Service Getting Started Guide

### Step 1: Make sure your endpoint is ready to process Amazon SNS messages

```
x-amz-sns-topic-arn: arn:aws:sns:us-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012:MyTopic:2bcf
bf39-05c3-41de-beaa-fcfcc21c8f55
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" :
    "2336412f371687551e624d0980555130712f794c5f6a886692768160a74ba6f3bd71854256ad0242809ccc29417f1f02869c582f
    bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "TopicArn" : "arn:aws:sns:us-east-1:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
east-1:123456789012:MyTopic.\nTo confirm the subscription, visit the Sub
scribeURL included in this message.",
  "SubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=ConfirmSub
scription&TopicArn=arn:aws:sns:us-east-1:123456789012:MyTop
ic&Token=2336412f371687551e624d0980555130712f794c5f6a886692768160a74ba6f3bd71854256ad0242809ccc29417f1f02869c582f
bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "skvXQIEpH+DcEwjAPg809mY8dReBSwksfg2S7WKQcicnK
WLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDVrY0Ad8L10Hs3zH81mtnPk5uvvol
IC1cXGu43obcgFxeL3khZl8IKvO6lGWB6jI9b5+gLPoBclQ=",
  "SigningCertURL" : "https://sns.us-east-1.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. Optionally, you can 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. 98).
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:

## Amazon Simple Notification Service Getting Started Guide

### Step 1: Make sure your endpoint is ready to process Amazon SNS 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 \(p. 109\)](#). 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-east-1:123456789012: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 the Token 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 \(p. 108\)](#). 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-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012: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-east-1:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
  "SignatureVersion" : "1",
  "Signature" : "duKygmBw6JRNwm1LFQL4ICB0bnXrdB8ClRMTQFGBqwLpGbm78tJ4etTwC5zU7O3tS6tGpey3ejedNdOJ+1fkIp9F2/LmNVKb5aF1Yq+9rk9ZiPph5Y1LmWsDcyC5T+Sy9/umic5S0UQc2PEtgdgdpVBahwNODMW4JPwk0kAJJztnc=" ,
```

## Amazon Simple Notification Service Getting Started Guide

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

```
"SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
"UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

5. Make sure that your endpoint responds to the HTTP POST message from Amazon SNS with the appropriate status code. The connection will time out in 15 seconds. If your endpoint does not respond before the connection times out or if your endpoint returns a status code outside the range of 200-4xx, Amazon SNS will consider the delivery of the message as a failed attempt. For more details on what is considered a failed delivery attempt, see [Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints \(p. 82\)](#).
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, Amazon SNS 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 3 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 last delivery attempt. If you don't like the default delivery policy, you can set a different delivery policy on the endpoint. For more information about delivery policies, see [Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints \(p. 82\)](#). To be clear, Amazon SNS attempts to retry only after a delivery attempt has failed. For details on what Amazon SNS considers a failed delivery, see [Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints \(p. 82\)](#). 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 \(p. 93\)](#).
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 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. 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.

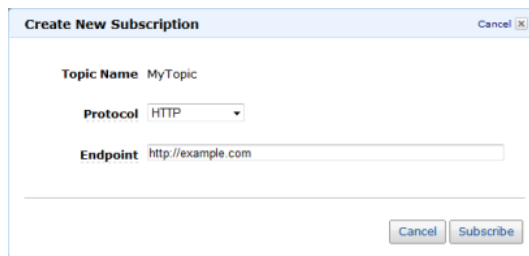
### To subscribe an HTTP or HTTPS endpoint to a topic using the Amazon SNS console

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select the topic from the **Navigation** pane.
3. Click **Create New Subscription**, select **HTTP** or **HTTPS** for **Protocol**, paste in the URL for the endpoint that you want the topic to send messages to for **Endpoint**, and click **Subscribe**.

## Amazon Simple Notification Service Getting Started Guide

### Step 3: Confirm the subscription

---



4. 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 \(p. 76\)](#) 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 3 retries with a delay between failed attempts set at 20 seconds. As discussed in [Step 1 \(p. 76\)](#), 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. For details about how delivery policies work and how to set them, see [Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints \(p. 82\)](#).

## 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:

## Amazon Simple Notification Service Getting Started Guide

### Step 5: Give users permissions to publish to the topic (Optional)

- 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, you should use the second method.

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

```
{
  "Statement": [{
    "Sid": "AllowPublishToMyTopic",
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"
  }]
}
```

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.

```
{
  "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 \(p. 76\)](#), 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 **Navigation** pane, select the topic and click **Publish to Topic**.
3. In the **Subject** box, enter a subject (for example, Testing publish to my endpoint) 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 \(p. 84\)](#)
- [Setting the Maximum Receive Rate \(p. 85\)](#)
- [Immediate Retry Phase \(p. 88\)](#)
- [Pre-Backoff Phase \(p. 89\)](#)
- [Backoff Phase \(p. 90\)](#)
- [Post-Backoff Phase \(p. 92\)](#)

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.

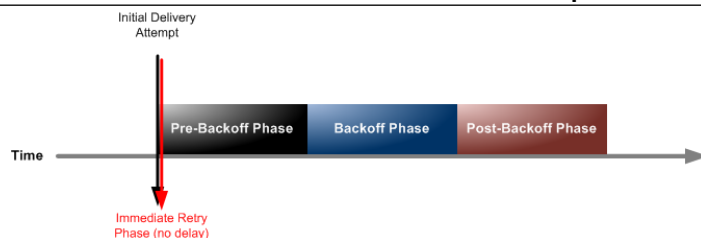
Amazon SNS only attempts a retry after a failed delivery attempt. Amazon SNS considers the following situations as a failed delivery attempt.

- 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.

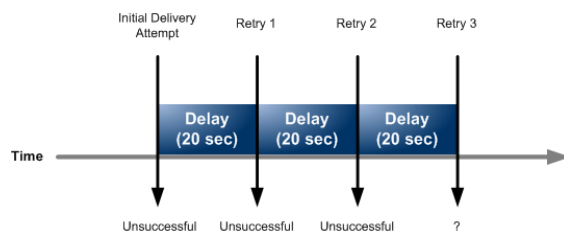
# Amazon Simple Notification Service Getting Started Guide

## Setting Amazon SNS Delivery Retry Policies for HTTP/HTTPS Endpoints

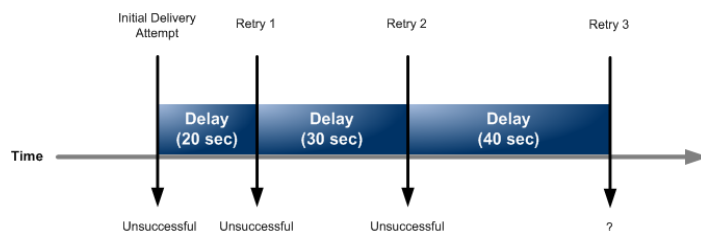


1. **Immediate Retry Phase (p. 88)**—Also called the *no delay phase*, this phase occurs immediately after the initial delivery attempt. The value you set for **Retries with no delay** determines the number of retries immediately after the initial delivery attempt. There is no delay between retries in this phase.
2. **Pre-Backoff Phase (p. 89)**—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. 90)**—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. 92)**—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.





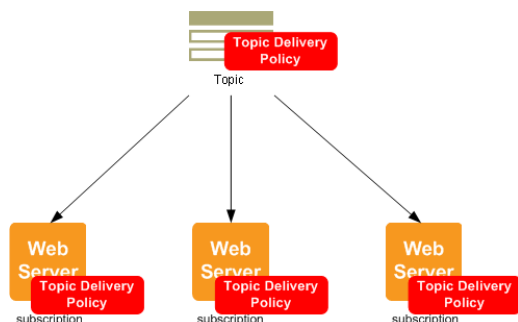
The maximum lifetime of a message in the system is one hour. This one hour limit cannot be extended by a delivery policy.

**Note**

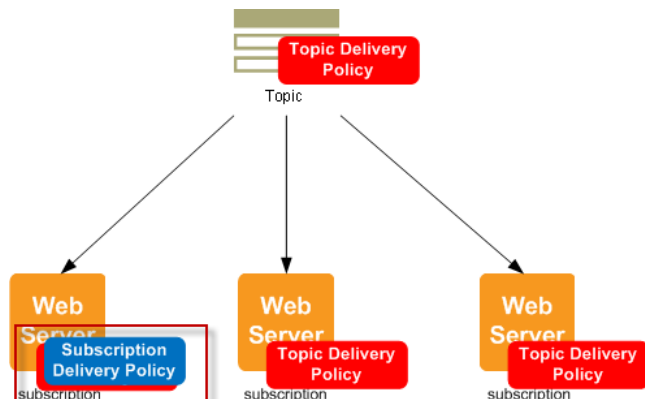
Only HTTP and HTTPS subscription types are supported by delivery policies. Support for other Amazon SNS subscription types (e.g., email, SQS, and SMS) is not currently available.

## 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.

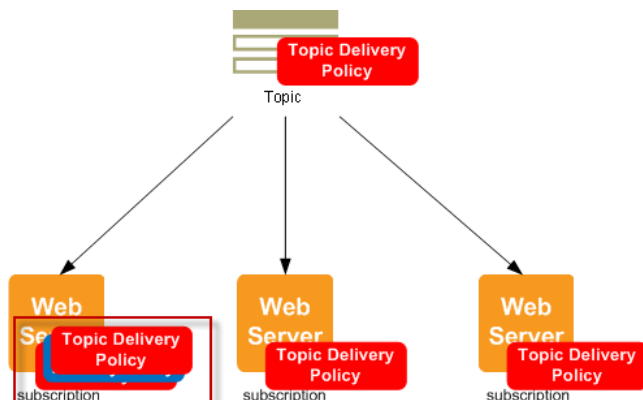


You can also apply delivery policies to individual subscriptions. If you assign a delivery policy to a subscription, the subscription-level policy takes precedence over the topic-level delivery policy. In the following diagram, one subscription has a subscription-level delivery policy whereas the two 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.





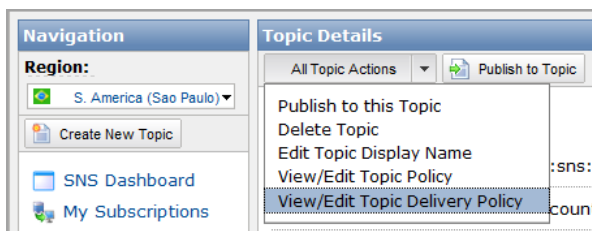
## Setting the Maximum Receive Rate

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 **View/Edit Topic Delivery Policy** dialog box. For more information, see [To set the maximum receive rate for a topic \(p. 85\)](#).
- To set a maximum receive rate that applies to a specific subscription, apply the setting at the subscription level using the **View/Edit Subscription Delivery Policy** dialog box. For more information, see [To set the maximum receive rate for a subscription \(p. 86\)](#).

### 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. Select a topic in the **Navigation** pane, and in the **Topic Details** pane select **View/Edit Topic Delivery Policy** from the **All Topic Actions** list.



3. Type an integer value (e.g., 2) in the **Maximum receive rate** box.

## Amazon Simple Notification Service Getting Started Guide

### Setting the Maximum Receive Rate

**View/Edit Topic Delivery Policy** Cancel

**Basic View** **Advanced View**

**Delivery Policy**

**Allow these delivery policies to this topic:**

Number of retries:   
Between 0 - 100

Retries with no delay:   
Between (0 - number of retries)

Minimum delay:   
In seconds. Between 0 - maximum delay

Minimum delay retries:   
Between (0 - number of retries)

Maximum delay:   
In seconds. Between minimum delay - 1 hr

Maximum delay retries:   
Between (0 - number of retries)

**Maximum receive rate:**   
Receives per second. >= 1

4. Click **Update Delivery Policy** to save your changes.

**View/Edit Topic Delivery Policy** Cancel

**Basic View** **Advanced View**

**Delivery Policy**

**Allow these delivery policies to this topic:**

Number of retries:   
Between 0 - 100

Retries with no delay:   
Between (0 - number of retries)

Minimum delay:   
In seconds. Between 0 - maximum delay

Minimum delay retries:   
Between (0 - number of retries)

Maximum delay:   
In seconds. Between minimum delay - 1 hr

Maximum delay retries:   
Between (0 - number of retries)

Maximum receive rate:   
Receives per second. >= 1

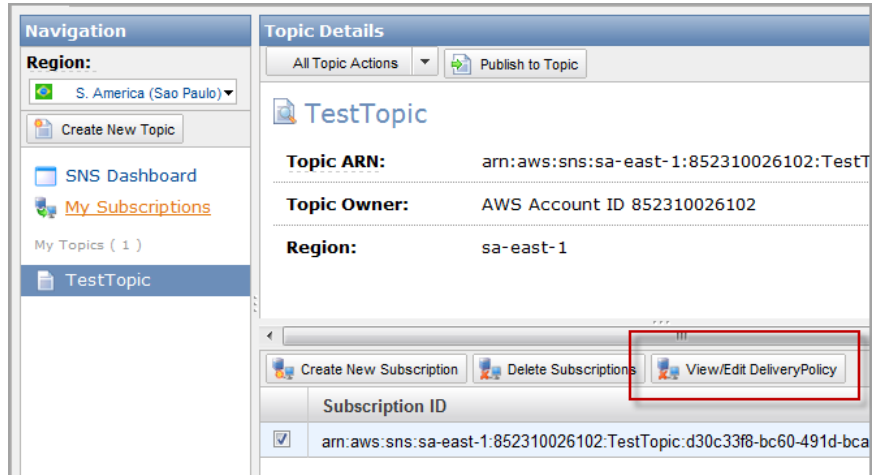
Cancel **Update Delivery Policy**

#### To set the maximum receive rate for a subscription

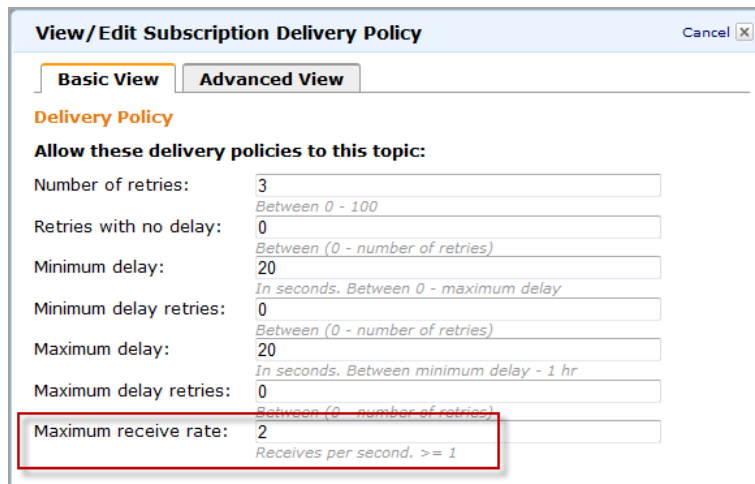
1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. Select a topic in the **Navigation** pane.
3. In the **Topic Details** pane, select a subscription and click **View/Edit Delivery Policy**.

## Amazon Simple Notification Service Getting Started Guide

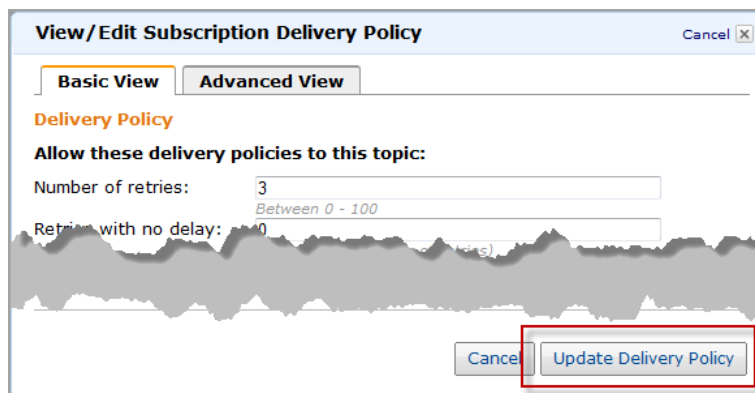
### Setting the Maximum Receive Rate



4. Type an integer value (e.g., 2) in the **Maximum receive rate** box.



5. Click **Update Delivery Policy** to save your changes.

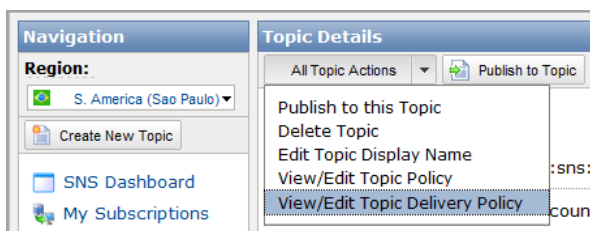


## Immediate Retry Phase

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. Select a topic in the **Navigation** pane, and in the **Topic Details** pane select **View/Edit Topic Delivery Policy** from the **All Topic Actions** list.



3. Type an integer value in the **Retries with no delay** box.

A screenshot of the 'View/Edit Topic Delivery Policy' dialog box. The 'Basic View' tab is selected. Under the 'Delivery Policy' section, it says 'Allow these delivery policies to this topic:'. There are several input fields: 'Number of retries' (value 3), 'Retries with no delay' (value 0, highlighted with a red box), 'Minimum delay' (value 20), 'Minimum delay retries' (value 0), 'Maximum delay' (value 20), 'Maximum delay retries' (value 0), and 'Maximum receive rate' (empty). Each field has a small text hint below it. The 'Retries with no delay' field is highlighted with a red rectangular box.

4. Click **Update Delivery Policy** to save your changes.

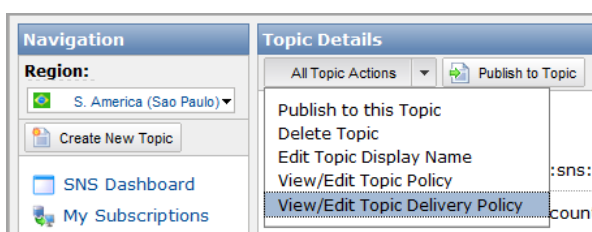
A screenshot of the 'View/Edit Topic Delivery Policy' dialog box, similar to the previous one. The 'Retries with no delay' field is still highlighted with a red box. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Update Delivery Policy'. The 'Update Delivery Policy' button is highlighted with a red rectangular box.

## 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. Select a topic in the **Navigation** pane, and in the **Topic Details** pane select **View/Edit Topic Delivery Policy** from the **All Topic Actions** list.



3. Type an integer value in the **Minimum delay retries** box.

A screenshot of the 'View/Edit Topic Delivery Policy' dialog box. It has a 'Cancel' button in the top right. There are two tabs: 'Basic View' (selected) and 'Advanced View'. Under 'Delivery Policy', it says 'Allow these delivery policies to this topic:'. Below this are several input fields with their current values and ranges: 'Number of retries:' with value 3 and range 'Between 0 - 100'; 'Retries with no delay:' with value 0 and range 'Between (0 - number of retries)'; 'Minimum delay:' with value 20 and range 'In seconds. Between 0 - maximum delay'; 'Minimum delay retries:' with value 0 and range 'Between (0 - number of retries)'; 'Maximum delay:' with value 20 and range 'In seconds. Between minimum delay - 1 hr'; 'Maximum delay retries:' with value 0 and range 'Between (0 - number of retries)'; and 'Maximum receive rate:' with an empty field and range 'Receives per second. >= 1'. The 'Minimum delay retries' field is highlighted with a red rectangular box.

4. Type an integer value in the **Minimum delay** box 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**.

## Amazon Simple Notification Service Getting Started Guide

### Backoff Phase

**View/Edit Topic Delivery Policy** Cancel X

**Basic View** **Advanced View**

**Delivery Policy**

**Allow these delivery policies to this topic:**

Number of retries: 3  
Between 0 - 100

Retries with no delay: 0  
Between (0 - number of retries)

**Minimum delay:** 20  
In seconds. Between 0 - maximum delay

Minimum delay retries: 0  
Between (0 - number of retries)

Maximum delay: 20  
In seconds. Between minimum delay - 1 hr

Maximum delay retries: 0  
Between (0 - number of retries)

Maximum receive rate:   
Receives per second. >= 1

5. Click **Update Delivery Policy** to save your changes.

**View/Edit Topic Delivery Policy** Cancel X

**Basic View** **Advanced View**

**Delivery Policy**

**Allow these delivery policies to this topic:**

Number of retries: 3  
Between 0 - 100

Retries with no delay: 0  
Between (0 - number of retries)

Cancel Update Delivery Policy

## 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.

The following screen shot shows the Amazon SNS console fields that pertain to the backoff phase.

## Amazon Simple Notification Service Getting Started Guide

### Backoff Phase

**View/Edit Topic Delivery Policy** Cancel

**Basic View** **Advanced View**

**Delivery Policy**

**Allow these delivery policies to this topic:**

Number of retries: 3  
Between 0 - 100

Retries with no delay: 0  
Between (0 - number of retries)

Minimum delay: 20  
In seconds. Between 0 - maximum delay

Minimum delay retries: 0  
Between (0 - number of retries)

Maximum delay: 20  
In seconds. Between minimum delay - 1 hr

Maximum delay retries: 0  
Between (0 - number of retries)

Maximum receive rate:   
Receives per second. >= 1

Retry backoff function: Linear

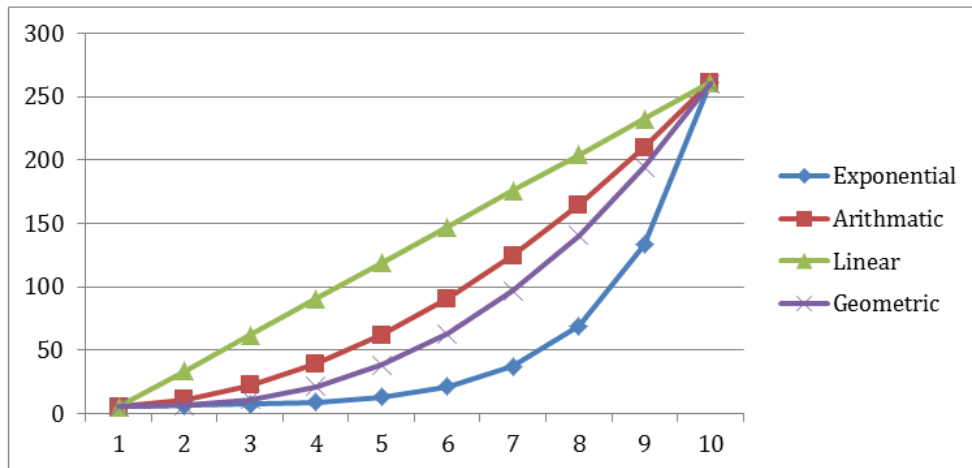
Subscription Override: ☐ Unselect to allow subscription overrides.

Cancel Update Delivery Policy

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.

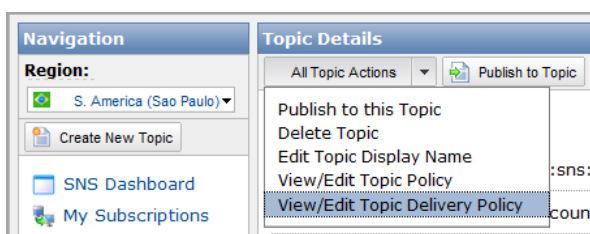


## Post-Backoff Phase

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. Select a topic in the **Navigation** pane, and in the **Topic Details** pane select **View/Edit Topic Delivery Policy** from the **All Topic Actions** list.



3. Type an integer value in the **Maximum delay retries** box.

A screenshot of the 'View/Edit Topic Delivery Policy' dialog box in the AWS SNS console, showing the 'Basic View' tab. The dialog has a title bar with 'View/Edit Topic Delivery Policy' and a 'Cancel' button. Below the title bar are two tabs: 'Basic View' (selected) and 'Advanced View'. Under the 'Basic View' tab, the section 'Delivery Policy' is shown. It contains the text 'Allow these delivery policies to this topic:' followed by several input fields with their respective ranges: 'Number of retries:' (3, range 'Between 0 - 100'), 'Retries with no delay:' (0, range 'Between (0 - number of retries)'), 'Minimum delay:' (20, range 'In seconds. Between 0 - maximum delay'), 'Minimum delay retries:' (0, range 'Between (0 - number of retries)'), 'Maximum delay:' (20, range 'In seconds. Between minimum delay - 1 hr'), 'Maximum delay retries:' (0, range 'Between (0 - number of retries)'), and 'Maximum receive rate:' (empty, range 'Receives per second. >= 1'). The 'Maximum delay retries' input field is highlighted with a red rectangular box.

4. Type an integer value in the **Maximum delay** box 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**.



**Amazon Simple Notification Service Getting Started  
Guide  
Certificate Authorities (CA) Recognized by Amazon SNS  
for HTTPS Endpoints**

The screenshot shows the 'View/Edit Topic Delivery Policy' dialog box with the 'Basic View' tab selected. The 'Delivery Policy' section is active. The 'Allow these delivery policies to this topic:' section contains several input fields. The 'Maximum delay' field, which has a value of '20' and a tooltip 'In seconds. Between minimum delay - 1 hr', is highlighted with a red rectangular box. Other fields include 'Number of retries' (3), 'Retries with no delay' (0), 'Minimum delay' (20), 'Minimum delay retries' (0), 'Maximum delay retries' (0), and 'Maximum receive rate' (empty).

5. Click **Update Delivery Policy** to save your changes.

This screenshot shows the same 'View/Edit Topic Delivery Policy' dialog box, but with the 'Update Delivery Policy' button highlighted by a red rectangular box. The button is located at the bottom right of the dialog, next to a 'Cancel' button. The 'Maximum delay' field is still visible but not highlighted.

## 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.

```
digicertassuredidrootca, Jan 7, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):  
87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72  
trustcenterclass2caii, Jan 7, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):  
CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23  
thawtepremiumserverca, Dec 2, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46  
swissignsilverg2ca, Aug 13, 2008, trustedCertEntry,
```

**Amazon Simple Notification Service Getting Started  
Guide  
Certificate Authorities (CA) Recognized by Amazon SNS  
for HTTPS Endpoints**

Certificate fingerprint (MD5):  
E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13  
swisssignplatinumg2ca, Aug 13, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):  
C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6  
equifaxsecureebusinesscal, Jul 18, 2003, trustedCertEntry,  
Certificate fingerprint (MD5):  
64:9C:EF:2E:44:FC:C6:8F:52:07:D0:51:73:8F:CB:3D  
thawteserverca, Dec 2, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2  
utnuserfirstclientauthemailca, May 2, 2006, trustedCertEntry,  
Certificate fingerprint (MD5):  
D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7  
thawtepersonalfreemailca, Dec 2, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65  
utnuserfirsthardwareca, May 2, 2006, trustedCertEntry,  
Certificate fingerprint (MD5):  
4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39  
entrustevca, Apr 28, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4  
certumca, Feb 10, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9  
addtrustclasslca, May 2, 2006, trustedCertEntry,  
Certificate fingerprint (MD5):  
1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC  
entrustrootcag2, Jun 22, 2010, trustedCertEntry,  
Certificate fingerprint (MD5):  
4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2  
equifaxsecureca, Jul 18, 2003, trustedCertEntry,  
Certificate fingerprint (MD5):  
67:CB:9D:C0:13:24:8A:82:9B:B2:17:1E:D1:1B:EC:D4  
quovadisrootca3, Jun 9, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF  
quovadisrootca2, Jun 9, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B  
digicerthighassuranceevrootca, Jan 7, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):  
D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A  
secomvalicertclasslca, May 1, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):  
65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB  
equifaxsecureglobalebusinesscal, Jul 18, 2003, trustedCertEntry,  
Certificate fingerprint (MD5):  
8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC  
geotrustuniversalca, Dec 3, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48  
thawteprimaryrootcag3, Nov 24, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):  
FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31  
verisignclass3ca, Dec 2, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

**Amazon Simple Notification Service Getting Started  
Guide  
Certificate Authorities (CA) Recognized by Amazon SNS  
for HTTPS Endpoints**

EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4  
deutschetelekomrootca2, Nov 6, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):

74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08  
utnuserfirstobjectca, May 2, 2006, trustedCertEntry,  
Certificate fingerprint (MD5):

A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9  
geotrustprimaryca, Nov 24, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF  
verisignclass1ca, Dec 2, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E  
baltimorecodesigningca, May 10, 2002, trustedCertEntry,  
Certificate fingerprint (MD5):

90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22  
baltimorecybertrustca, May 10, 2002, trustedCertEntry,  
Certificate fingerprint (MD5):

AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4  
starfieldclass2ca, Jan 20, 2005, trustedCertEntry,  
Certificate fingerprint (MD5):

32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24  
camerfirmachamberscommerceca, Oct 10, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):

B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84  
ttelesecglobalrootclass3ca, Feb 10, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF  
verisignclass3g5ca, Nov 24, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C  
trustcenteruniversalcai, Jan 7, 2008, trustedCertEntry,  
Certificate fingerprint (MD5):

45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C  
ttelesecglobalrootclass2ca, Feb 10, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A  
verisignclass3g3ca, Mar 25, 2004, trustedCertEntry,  
Certificate fingerprint (MD5):

CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09  
certumtrustednetworkca, Feb 10, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78  
certplusclass3pprimaryca, May 27, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB  
verisignclass3g2ca, Mar 25, 2004, trustedCertEntry,  
Certificate fingerprint (MD5):

A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9  
globalsignr3ca, Aug 17, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28  
utndatacorpsgcca, May 2, 2006, trustedCertEntry,  
Certificate fingerprint (MD5):

B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06  
secomscrootca2, Aug 17, 2009, trustedCertEntry,  
Certificate fingerprint (MD5):

6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43

**Amazon Simple Notification Service Getting Started  
Guide  
Certificate Authorities (CA) Recognized by Amazon SNS  
for HTTPS Endpoints**

```

secomscrootcal, May 1, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
gtecybertrustglobalca, May 10, 2002, trustedCertEntry,
Certificate fingerprint (MD5):
CA:3D:D3:68:F1:03:5C:D0:32:FA:B8:2B:59:E8:5A:DB
verisignuniversalrootca, Nov 24, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
trustcenterclass4caii, Jan 7, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0
globalsignr2ca, Aug 1, 2007, trustedCertEntry,
Certificate fingerprint (MD5):
94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
certplusclass2primaryca, May 27, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
digicertglobalrootca, Jan 7, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
globalsignca, Mar 26, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
thawteprimaryrootca, Nov 24, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
geotrustglobalca, Jul 18, 2003, trustedCertEntry,
Certificate fingerprint (MD5):
F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
soneraclass2ca, Mar 28, 2006, trustedCertEntry,
Certificate fingerprint (MD5):
A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
verisightsaca, Aug 13, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
7F:66:7A:71:D3:EB:69:78:20:9A:51:14:9D:83:DA:20
quovadisrootca, Jun 9, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
soneraclass1ca, Mar 28, 2006, trustedCertEntry,
Certificate fingerprint (MD5):
33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
valicertclass2ca, Jan 20, 2005, trustedCertEntry,
Certificate fingerprint (MD5):
A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
comodoaaaca, May 2, 2006, trustedCertEntry,
Certificate fingerprint (MD5):
49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
aolrootca2, Mar 26, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
keynectisrootca, Jun 8, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26
addtrustqualifiedca, May 2, 2006, trustedCertEntry,
Certificate fingerprint (MD5):
27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
aolrootcal, Jan 17, 2008, trustedCertEntry,

```

**Amazon Simple Notification Service Getting Started  
Guide  
Certificate Authorities (CA) Recognized by Amazon SNS  
for HTTPS Endpoints**

```

Certificate fingerprint (MD5):
14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
  verisignclass2g3ca, Mar 25, 2004, trustedCertEntry,
Certificate fingerprint (MD5):
F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
  addtrustexternalca, May 2, 2006, trustedCertEntry,
Certificate fingerprint (MD5):
1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  verisignclass2g2ca, Mar 25, 2004, trustedCertEntry,
Certificate fingerprint (MD5):
2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
  geotrustprimarycag3, Nov 24, 2009, trustedCertEntry,
Certificate fingerprint (MD5):
B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  swisssigngoldg2ca, Aug 13, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  entrust2048ca, Jun 22, 2010, trustedCertEntry,
Certificate fingerprint (MD5):
EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  gtecybertrust5ca, May 10, 2002, trustedCertEntry,
Certificate fingerprint (MD5):
7D:6C:86:E4:FC:4D:D1:0B:00:BA:22:BB:4E:7C:6A:8E
  camerfirmachambersignca, Oct 10, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  camerfirmachambersca, Oct 10, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  godaddyclass2ca, Jan 20, 2005, trustedCertEntry,
Certificate fingerprint (MD5):
91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
  entrustsslca, Jan 9, 2003, trustedCertEntry,
Certificate fingerprint (MD5):
DF:F2:80:73:CC:F1:E6:61:73:FC:F5:42:E9:C5:7C:EE
  verisignclass1g3ca, Mar 25, 2004, trustedCertEntry,
Certificate fingerprint (MD5):
B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
  secomevrootcal, May 1, 2008, trustedCertEntry,
Certificate fingerprint (MD5):
22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  verisignclass1g2ca, Mar 25, 2004, trustedCertEntry,
Certificate fingerprint (MD5):
DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
  godaddysecurecertificationauthority, Jul 19, 2010, trustedCertEntry,
Certificate fingerprint (MD5):
D5:DF:85:B7:9A:52:87:D1:8C:D5:0F:90:23:2D:B5:34

```

# Verifying the Signatures of Amazon SNS Messages

Optionally, you can 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.

For example code for a Java servlet that handles Amazon SNS messages and also verifies the signature, see [Example Code for an Amazon SNS Endpoint Java Servlet \(p. 103\)](#).

## To verify the signature of an Amazon SNS message

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 \(p. 76\)](#).

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 uses signature version 1. This section provides the steps for creating a signature using that version.

2. Get the X509 certificate that Amazon SNS used to sign the message. The *Message* value that is the location of the X509 certificate used to create 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 *Message* is used as the key for signing the message. You'll need this when you sign the message.
4. Determine the type of the message. The format of the string to sign depends on the type of message. The *Type* value specifies the message type.
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.

## Amazon Simple Notification Service Getting Started Guide

### Verifying the Signatures of Amazon SNS Messages

---

```
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.

```
MessageId
3d891288-136d-417f-bc05-901c108273ee
SubscribeURL
https://sns.us-east-1.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-1:123456789012:s4-MySNSTopic-1G1WEFCOXTCP&Token=2336412f37fb687f5d51e6e241d09c8058323f60b964268bfe08ce35640228c208a66d3621bd9f7b012918cfd
Timestamp
2012-06-03T19:25:13.719Z
Token
2336412f37fb687f5d51e6e241d09c8058323f60b964268bfe08ce35640228c208a66d3621bd9f7b012918cfd
TopicArn
arn:aws:sns:us-east-1:123456789012: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. Calculate the signature and compare the signature that you calculated with the signature originally sent with the message. Call a hash function with the SHA1withRSA algorithm to sign the string (from

step 3) to sign using the public key (from step 5). Some libraries such as the `java.security.Signature` class enable you to calculate a signature and compare it against an existing signature.

## Using an AWS CloudFormation Template to Set Up a Topic and an HTTP Endpoint

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 create an HTTP endpoint on a running AWS Elastic Beanstalk application and create a topic that subscribes that endpoint.

For more information about deploying AWS resources using an AWS CloudFormation template, see [Get Started](#) in the *AWS CloudFormation User Guide*. For more information about using AWS Elastic Beanstalk to deploy applications, see [What Is AWS Elastic Beanstalk and Why Do I Need It?](#) in the *AWS Elastic Beanstalk User Guide*.

The example template deploys an AWS Elastic Beanstalk application that serves as an HTTP endpoint that can handle subscription confirmation and notification messages from an Amazon SNS topic and creates a topic that subscribes that HTTP endpoint. The template gives appropriate permissions for members of an IAM group to publish to the topic and also creates an IAM user that is added to the group.

You can download this template

([https://s3.amazonaws.com/cloudformation-templates-us-east-1/sns\\_httpendpoint.template](https://s3.amazonaws.com/cloudformation-templates-us-east-1/sns_httpendpoint.template)) from the [AWS CloudFormation Sample Templates](#) page.

The template takes care of the setup steps for you by deploying AWS Elastic Beanstalk application `MyEndpointApplication` that serves as an HTTP endpoint that can handle subscription confirmation and notification messages from an Amazon SNS topic. The `MyEndpointApplication` application contains an application version `Initial Version` that specifies the web application archive (WAR) file containing the Java servlet that handles Amazon SNS HTTP requests. For more information about the example servlet (`MySnsHttpServlet`), see [Example Code for an Amazon SNS Endpoint Java Servlet \(p. 103\)](#). To see the WAR file deployed in the application, see <https://s3.amazonaws.com/cloudformation-examples/sns-http-example.war>) The application also contains a configuration template `DefaultConfiguration` that specifies that the application should run in an environment using the 32bit Amazon Linux running Tomcat 7 container and that the environment created by the template should assign the key pair specified by the input parameter `KeyName` to the Amazon EC2 instances deployed in the environment. The `MyEndpointEnvironment` environment deploys `MyEndPointApplication` using the application version `Initial Version` and the configuration template `DefaultConfiguration`.

`MySNSTopic` is set up to publish to one subscribed endpoint, which is the HTTP endpoint implemented with the `MyEndpointApplication`. The `Endpoint` property is a URL constructed from the DNS address of the load balancer for `MyEndpointApplication` and the url-pattern for the servlet installed from the application's WAR file. `MyPublishTopicGroup` is an IAM group whose members have permission to publish to `MySNSTopic` using the [Publish](#) API or `sns-publish` command. The template creates the IAM user `MyPublishUser` gives it a login profile and access keys. The user who creates a stack with this template specifies the password for the login profile as input parameters. The template creates access keys for the IAM users with `MyPublishUserKey`. `AddUserToMyPublishTopicGroup` adds `MyPublishUser` to the `MyPublishTopicGroup` so that that user will have the permissions assigned to the group.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "This sample template creates an HTTP endpoint using AWS
Elastic Beanstalk, creates an Amazon SNS topic, and subscribes the HTTP endpoint
```



**Amazon Simple Notification Service Getting Started  
Guide  
Using an AWS CloudFormation Template to Set Up a  
Topic and an HTTP Endpoint**

```
to that topic. **WARNING** This template creates one or more Amazon EC2 in-
stances. You will be billed for the AWS resources used if you create a stack
from this template.",

  "Parameters" : {
    "KeyName" : {
      "Description" : "Name of an existing EC2 KeyPair to enable SSH access to
the Amazon EC2 instance(s) in the environment deployed for the AWS Elastic
Beanstalk application in this template.",
      "Type" : "String"
    },
    "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."
    }
  },

  "Resources" : {
    "MySNSTopic" : {
      "Type" : "AWS::SNS::Topic",
      "Properties" : {
        "Subscription" : [ {
          "Endpoint" : {
            "Fn::Join" : [ "/",
              [ "http/", { "Fn::GetAtt" : [ "MyEndpointEnvironment", "EndpointURL" ] } ], "myendpoint" ] },
          "Protocol" : "http"
        } ]
      }
    },
    "MyEndpointApplication" : {
      "Type" : "AWS::ElasticBeanstalk::Application",
      "Properties" : {
        "Description" : "HTTP endpoint to receive messages from Amazon SNS
subscription.",
        "ApplicationVersions" : [{
          "VersionLabel" : "Initial Version",
          "Description" : "Version 1.0",
          "SourceBundle" : {
            "S3Bucket" : "cloudformation-examples",
            "S3Key" : "sns-http-example.war"
          }
        } ],
        "ConfigurationTemplates" : [{
          "TemplateName" : "DefaultConfiguration",
          "Description" : "Default Configuration Version 1.0 - with SSH access",

          "SolutionStackName" : "32bit Amazon Linux running Tomcat 7",
          "OptionSettings" : [{
            "Namespace" : "aws:autoscaling:launchconfiguration",
            "OptionName" : "EC2KeyName",
            "Value" : { "Ref" : "KeyName" }
          } ]
        } ]
      }
    }
  }
}
```

## Amazon Simple Notification Service Getting Started Guide

### Using an AWS CloudFormation Template to Set Up a Topic and an HTTP Endpoint

```

    }
  },
  "MyEndpointEnvironment" : {
    "Type" : "AWS::ElasticBeanstalk::Environment",
    "Properties" : {
      "ApplicationName" : { "Ref" : "MyEndpointApplication" },
      "Description" : "AWS Elastic Beanstalk Environment running HTTP endpoint
for Amazon SNS subscription.",
      "TemplateName" : "DefaultConfiguration",
      "VersionLabel" : "Initial Version"
    }
  },
  "MyPublishUser" : {
    "Type" : "AWS::IAM::User",
    "Properties" : {
      "LoginProfile": {
        "Password": { "Ref" : "MyPublishUserPassword" }
      }
    }
  },
  "MyPublishUserKey" : {
    "Type" : "AWS::IAM::AccessKey",
    "Properties" : {
      "UserName" : { "Ref" : "MyPublishUser" }
    }
  },
  "MyPublishTopicGroup" : {
    "Type" : "AWS::IAM::Group",
    "Properties" : {
      "Policies": [
        {
          "PolicyName": "MyTopicGroupPolicy",
          "PolicyDocument": { "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "sns:Publish"
              ],
              "Resource": { "Ref" : "MySNSTopic" }
            }
          ] }
        ]
      }
    }
  },
  "AddUserToMyPublishTopicGroup" : {
    "Type" : "AWS::IAM::UserToGroupAddition",
    "Properties" : {
      "GroupName": { "Ref" : "MyPublishTopicGroup" },
      "Users" : [ { "Ref" : "MyPublishUser" } ]
    }
  },
  "Outputs" : {
    "MySNSTopicTopicARN" : {
      "Description" : "ARN for MySNSTopic.",
      "Value" : { "Ref" : "MySNSTopic" }
    }
  }
}

```

```
    },
    "MyPublishUserInfo" : {
      "Description" : "Information about MyPublishUser.",
      "Value" : { "Fn::Join" : [
        " ",
        [
          "ARN:",
          { "Fn::GetAtt" : [ "MyPublishUser", "Arn" ] },
          "Access Key:",
          { "Ref" : "MyPublishUserKey" },
          "Secret Key:",
          { "Fn::GetAtt" : [ "MyPublishUserKey", "SecretAccessKey" ] }
        ]
      ] }
    },
    "URL" : {
      "Description" : "URL of the HTTP endpoint hosted on AWS Elastic Beanstalk  
and subscribed to topic.",
      "Value" : { "Fn::Join" : [ "/", [ "http://", { "Fn::GetAtt" : [ "MyEndpointEn  
vironment", "EndpointURL" ] }, "myendpoint" ] ] }
    }
  }
}
```

## Example Code for an Amazon SNS Endpoint Java Servlet

The following code snippets are from an example Java servlet that processes Amazon SNS HTTP POST requests. You can deploy this servlet as an AWS Elastic Beanstalk application and subscribe a topic to it using the AWS CloudFormation template in [Using an AWS CloudFormation Template to Set Up a Topic and an HTTP Endpoint \(p. 100\)](#). You can view the code for the servlet in the src directory of the WAR file at <https://s3.amazonaws.com/cloudformation-examples/sns-http-example.war>.

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());
    }
}
```

**Amazon Simple Notification Service Getting Started  
Guide**  
**Example Code for an Amazon SNS Endpoint Java Servlet**

---

```
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.
```

**Amazon Simple Notification Service Getting Started  
Guide  
Example Code for an Amazon SNS Endpoint Java Servlet**

---

```
        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 \(p. 98\)](#).

```
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";
}
```

**Amazon Simple Notification Service Getting Started  
Guide  
Example Code for an Amazon SNS Endpoint Java Servlet**

---

```
        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;
    }
}
```

# Appendix E: Message and JSON Formats

---

Amazon SNS uses the following formats.

## Topics

- [HTTP/HTTPS Headers](#) (p. 108)
- [HTTP/HTTPS Subscription Confirmation JSON Format](#) (p. 109)
- [HTTP/HTTPS Notification JSON Format](#) (p. 111)
- [HTTP/HTTPS Unsubscribe Confirmation JSON Format](#) (p. 113)
- [SetSubscriptionAttributes Delivery Policy JSON Format](#) (p. 115)
- [SetTopicAttributes Delivery Policy JSON Format](#) (p. 116)

## 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 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 `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-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012: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.\n\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-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1336
```

**Amazon Simple Notification Service Getting Started  
Guide**  
**HTTP/HTTPS Subscription Confirmation JSON Format**

---

```
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" :
"236412f37b687551e6241d09c80555730d712f794c5f698866392768b6a71a6f3bd718542856ad0242809ee29417f1f0260c582f
bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "TopicArn" : "arn:aws:sns:us-east-1:123456789012:MyTopic",
  "Message" : "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.",
  "SubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-east-1:123456789012:MyTopic&Token=236412f37b687551e6241d09c80555730d712f794c5f698866392768b6a71a6f3bd718542856ad0242809ee29417f1f0260c582f
bacc99c583a916b9981dd2728f4ae6fdb82efd087cc3b7849e05798d2d2785c03b0879594eeac82c01f235d0e717736",

  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "skvXQIEpH+DcEwjAPg809mY8dReBSwksfg2S7WKQcikcNK
WLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDVrY0Ad8L10Hs3zH8lmtnP5uvvol
IC1CXGu43obcgFxeL3khZl8IKvO6lGWB6ji9b5+gLPoBclQ=",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

You can download the following JSON file to view the definition of the JSON format for a subscription confirmation: <https://sns.us-east-1.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-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012: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-east-1:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
```

**Amazon Simple Notification Service Getting Started  
Guide  
HTTP/HTTPS Notification JSON Format**

---

```
"SignatureVersion" : "1",
"Signature" : "duKygmBw6JRNwm1LFQL4ICB0bnXrdB8ClRMTQFGBqwLp
GbM78tJ4etTwC5zU703tS6tGpey3ejedNdOJ+1fkIp9F2/LmNVKb5aF1Yq+9rk9ZiPph5YlLmWsD
cyC5T+Sy9/umic5S0UQc2PEtgdpVBahwNodMW4JPwk0kAJJztnc=" ,
"SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem" ,
"UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&Sub
scriptionArn=arn:aws:sns:us-east-1:123456789012: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-east-1.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-fcfcc21c8f55.\n\nTo cancel this operation and restore the subscription, visit the SubscribeURL included in this message.
```

## MessageId

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

## Signature

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

## SignatureVersion

Version of the Amazon SNS signature used.

## SigningCertURL

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

## SubscribeURL

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

## Timestamp

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

## Token

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

## TopicArn

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

## Type

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

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

```
POST / HTTP/1.1
x-amz-sns-message-type: UnsubscribeConfirmation
x-amz-sns-message-id: 47138184-6831-46b8-8f7c-afc488602d7d
x-amz-sns-topic-arn: arn:aws:sns:us-east-1:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-east-1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1399
Content-Type: text/plain; charset=UTF-8
```

**Amazon Simple Notification Service Getting Started  
Guide  
HTTP/HTTPS Unsubscribe Confirmation JSON Format**

---

```
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "UnsubscribeConfirmation",
  "MessageId" : "47138184-6831-46b8-8f7c-afc488602d7d",
  "Token" : "2336412f37fb687f5d51e6e241d09c805a5a57b30d712f7948a98bac386ed
fe3e10314e873973b3e0a3c09119b722dedf2b5e31c59b13ed
b26417c19f109351e6f2169efa9085ffe97e10535f4179ac1a03590b0f541f209c190f9ae23219ed6c470453e06c19b5a9fdd27dab7c7",
  "TopicArn" : "arn:aws:sns:us-east-1:123456789012:MyTopic",
  "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-east-
1:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this
operation and restore the subscription, visit the SubscribeURL included in this
message.",
  "SubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=ConfirmSubscrip
tion&TopicArn=arn:aws:sns:us-east-1:123456789012:MyTop
ic&Token=2336412f37fb687f5d51e6e241d09c805a5a57b30d712f7948a98bac386ed
fe3e10314e873973b3e0a3c09119b722dedf2b5e31c59b13ed
b26417c19f109351e6f2169efa9085ffe97e10535f4179ac1a03590b0f541f209c190f9ae23219ed6c470453e06c19b5a9fdd27dab7c7",
  "Timestamp" : "2012-04-26T20:06:41.581Z",
  "SignatureVersion" : "1",
  "Signature" : "e8VAeRNHXgJmXqnqsHTlqOck7TIZsnk8zpJJJoQbr8leD+8kAHcke3ClC4VPOvd
pZo9s/vR9GOznKab6sjGxE8uwqDI9HwpDm8lGxSlFGuwCruWeecnt7MdJCNh0XK4XQCbtGoXB762ePJ
faSwi9tYwzW65zAFU04WkNBkNsIf60=",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationSer
vice-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

You can download the following JSON file to view the definition of the JSON format for an unsubscribe confirmation: <https://sns.us-east-1.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-Top  
ic%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:aws:sns:us-east-1:123456789012:My-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*.



# About This Guide

---

This is the *Amazon Simple Notification Service Getting Started Guide*. It was last updated on January 09, 2013.

Amazon Simple Notification Service is often referred to within this guide as "Amazon SNS." All copyrights and legal protections still apply.