Developer Guide

Amazon Simple Notification Service

Amazon Simple Notification Service: Developer Guide

Table of Contents

What is Amazon SNS?

Amazon Simple Notification Service (Amazon SNS) is a fully managed service that provides message delivery from publishers (producers) to subscribers (consumers). Publishers communicate asynchronously with subscribers by sending messages to a *topic*, which is a logical access point and communication channel.

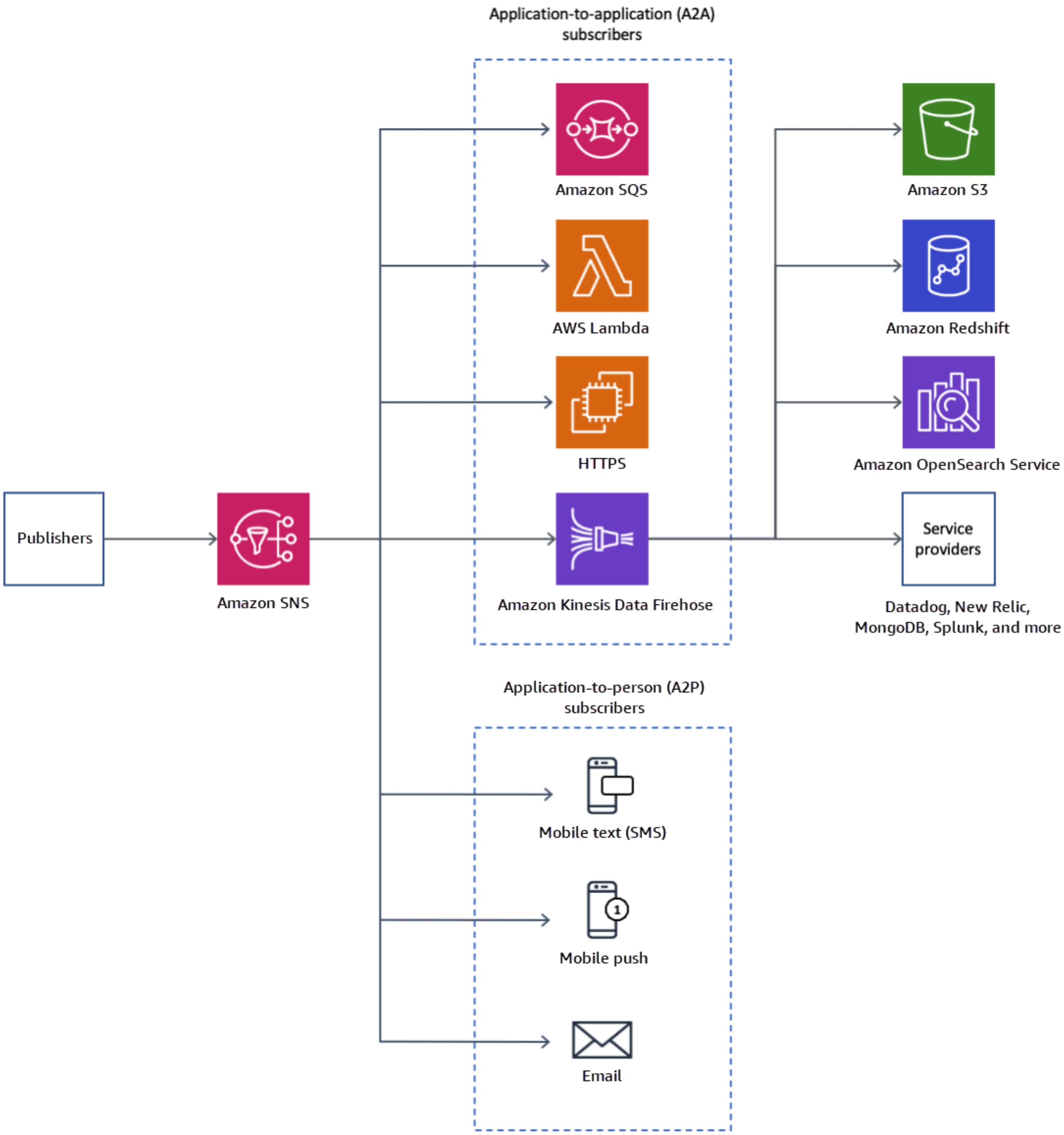
How it works

In SNS, publishers send messages to a topic, which acts as a communication channel. The topic acts as a logical access point, ensuring messages are delivered to multiple subscribers across different platforms.

Subscribers to an SNS topic can receive messages through different endpoints, depending on their use case, such as:

- Amazon SQS
- Lambda
- HTTP(S) endpoints
- Email
- Mobile push notifications
- Mobile text messages (SMS)
- Amazon Data Firehose
- Service providers (For example, Datadog, MongoDB, Splunk)

SNS supports both Application-to-Application (A2A) and Application-to-Person (A2P) messaging, giving flexibility to send messages between different applications or directly to mobile phones, email addresses, and more.


Application-to-application (A2A) subscribers
Amazon SQS
AWS Lambda
HTTPS
Amazon Kinesis Data Firehose
Amazon S3
Amazon Redshift
Amazon OpenSearch Service
Service providers
Datadog, New Relic, MongoDB, Splunk, and more
Publishers
Amazon SNS
Application-to-person (A2P) subscribers
Mobile text (SMS)
Mobile push
Email

Accessing Amazon SNS

You can access and manage Amazon SNS through the console, AWS CLI, or AWS SDKs, depending on your preferred method of interaction. The console offers a graphical interface for basic tasks, while the AWS CLI and SDKs provide advanced configuration and automation capabilities for more complex use cases.

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

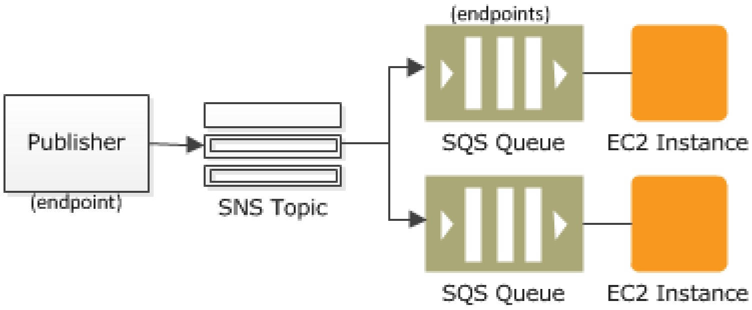
Common Amazon SNS scenarios

Use these common Amazon SNS scenarios to implement scalable, event-driven architectures and ensure reliable, real-time communication between applications and users.

Application integration

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

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



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

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

For more information, see the following resources:

- Fanout to Firehose delivery streams
- Fanout Amazon SNS notifications to Lambda functions for automated processing
- Fanout Amazon SNS notifications to Amazon SQS queues for asynchronous processing
- Fanout Amazon SNS notifications to HTTPS endpoints
- Event-Driven Computing with Amazon SNS and AWS Compute, Storage, Database, and Networking Services

Application alerts

Application and system alerts are notifications that are triggered by predefined thresholds. Amazon SNS can send these notifications to specified users via SMS and email. For example, you can receive immediate notification when an event occurs, such as a specific change to your Amazon EC2 Auto Scaling group, a new file uploaded to an Amazon S3 bucket, or a metric threshold breached in Amazon CloudWatch. For more information, see Setting up Amazon SNS notifications in the *Amazon CloudWatch User Guide*.

User notifications

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

Mobile push notifications

Mobile push notifications enable you to send messages directly to mobile apps. For example, you can use Amazon SNS to send update notifications to an app. The notification message can include a link to download and install the update. For more information about using Amazon SNS to send push notification messages, see Sending mobile push notifications with Amazon SNS.

Pricing for Amazon SNS

Amazon SNS has no upfront costs. You pay based on the number of messages that you publish, the number of notifications that you deliver, and any additional API calls for managing topics and subscriptions. Delivery pricing varies by endpoint type. You can get started for free with the Amazon SNS free tier. For information, see Worldwide SMS Pricing.

Amazon SNS features and capabilities

Amazon SNS offers a comprehensive set of features designed to enhance messaging between applications and users. These features enable seamless communication, secure message delivery, and robust message management, ensuring high availability, durability, and flexibility for a wide range of messaging use cases.

Application-to-application messaging

Application-to-application messaging supports subscribers such as delivery streams, Lambda functions, Amazon SQS queues, HTTP/S endpoints, and AWS Event Fork Pipelines. This allows for efficient message delivery in event-driven architectures.

Application-to-person notifications

Application-to-person notifications provide user notifications to subscribers such as mobile applications, mobile phone numbers, and email addresses.

Standard and FIFO topics

FIFO topics ensure strict message ordering, message grouping, and deduplication, allowing FIFO and standard queues to subscribe for message processing. Standard topics are used when message ordering and possible duplication are not critical, supporting all delivery protocols for broader use cases.

Message durability

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

- Published messages are stored across multiple, geographically separated servers and data centers.
- If a subscribed endpoint isn't available, Amazon SNS runs a delivery retry policy.

- To preserve any messages that aren't delivered before the delivery retry policy ends, you can create a dead-letter queue.

Message archiving, replay, and analytics

You can archive messages with Amazon SNS in multiple ways including subscribing Firehose delivery streams to SNS topics, which allows you to send notifications to analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, and more. Additionally, Amazon SNS FIFO topics support message archiving and replay as a no-code, in-place message archive that lets topic owners store (or *archive*) messages within their topic. Topic subscribers can then retrieve (or *replay*) the archived messages back to a subscribed endpoint. For more, see Amazon SNS message archiving and replay for FIFO topics.

Message attributes

Amazon SNS message attributes let you provide any arbitrary metadata about the message.

Message filtering

By default, each subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a filter policy to the topic subscription. A subscriber can also define the filter policy scope to enable payload-based or attribute-based filtering. The default value for the filter policy scope is `MessageAttributes`. When the incoming message attributes match the filter policy attributes, the message is delivered to the subscribed endpoint. Otherwise, the message is filtered out. When the filter policy scope is `MessageBody`, filter policy attributes are matched against the payload. For more information, see *Message filtering*.

Message security

Server-side encryption protects the contents of messages that are stored in Amazon SNS topics, using encryption keys provided by AWS KMS. For more information, see the section called "Securing data with server-side encryption" You can also establish a private connection between Amazon SNS and your virtual private cloud (VPC). for more information, see the section called "Securing traffic with VPC endpoints".

AWS services commonly used with Amazon SNS

Integrate Amazon SNS with multiple AWS Cloud services to boost message handling, improve access control, enable event-driven processing, and automate resources. This integration optimizes performance, strengthens security, and streamlines operations.

Amazon CloudWatch

Amazon CloudWatch provides monitoring and observability for Amazon SNS, helping you track message delivery, detect anomalies, and troubleshoot issues. With CloudWatch, you can:

- **Monitor Amazon SNS metrics** such as the number of messages published, delivered, or failed across topics and subscriptions.
- **Set up CloudWatch Alarms** to trigger automated actions when Amazon SNS metrics exceed predefined thresholds, such as high delivery failures or throttling.
- **Use CloudWatch Logs to capture Amazon SNS delivery status** for messages sent to HTTP/S, Lambda, and Amazon SQS endpoints for debugging and auditing.

For more information, see Monitoring Amazon SNS topics using CloudWatch.

Amazon SQS

Amazon SQS is a fully managed message queuing service that enables secure, durable, and scalable communication between distributed software components. It helps decouple application architecture by buffering messages, ensuring reliable delivery, and preventing system failures due to message loss. Amazon SQS integrates with Amazon SNS in the following ways:

- **Dead-letter queues** – Amazon SNS can route undeliverable messages to an Amazon SQS dead-letter queue for troubleshooting and reprocessing.
- **Topic subscriptions** – You can subscribe an Amazon SQS queue to an Amazon SNS topic, allowing Amazon SNS to fan out messages to multiple consumers using Amazon SQS.
- **FIFO queue support** – Amazon SQS FIFO queues can be subscribed to Amazon SNS FIFO topics, ensuring strict message ordering and exactly-once processing. Standard Amazon SQS queues can also subscribe to Amazon SNS topics but do not guarantee ordered message delivery or deduplication.

AWS CloudFormation

AWS CloudFormation automates the provisioning and management of AWS resources, including Amazon SNS topics and subscriptions, using infrastructure as code (IaC). With AWS CloudFormation, you can:

- **Define Amazon SNS topics, subscriptions, and permissions** in a reusable, version-controlled template.
- **Ensure consistent deployment** of Amazon SNS resources across multiple AWS accounts and Regions.

- **Update or modify Amazon SNS configurations** using change sets without manual intervention.

For more information, see the AWS CloudFormation User Guide.

AWS CloudTrail

CloudTrail provides visibility into API activity for Amazon SNS, helping you monitor and audit access to Amazon SNS topics, subscriptions, and messages. With CloudTrail, you can:

- **Track API calls made to Amazon SNS**, including who accessed or modified topics, subscriptions, and permissions.
- **Detect unauthorized or unexpected activity** by analyzing logs for security and compliance purposes.
- **Integrate with Amazon CloudWatch or AWS Security Hub** to create alerts based on unusual Amazon SNS actions.

For more information, see the Logging AWS SNS API calls using AWS CloudTrail.

AWS Lambda

AWS Lambda is a serverless compute service that automatically runs your code in response to events, eliminating the need to provision or manage servers. It allows you to build event-driven applications that scale automatically and execute in a highly available compute environment.

Amazon SNS integrates with Lambda by allowing you to subscribe a Lambda function to an Amazon SNS topic. When an Amazon SNS topic receives a message, it can trigger the Lambda function, enabling real-time processing, automation, and application logic execution. This integration is commonly used for:

- **Event-driven processing** – Automatically trigger functions in response to Amazon SNS messages.
- **Data transformation** – Modify or filter Amazon SNS messages before forwarding them to other services.
- **Automated workflows** – Process notifications for application alerts, system monitoring, or event orchestration.

AWS Identity and Access Management (IAM)

IAM provides secure access control for AWS resources, allowing you to manage who can access your Amazon SNS topics, what actions they can perform, and under what conditions. With IAM, you can:

- **Authenticate users and services** before they can interact with Amazon SNS topics.
- **Define fine-grained permissions** to specify which Amazon SNS topics users or roles can publish to, subscribe to, or manage.
- **Use identity-based policies** to enforce security best practices, such as restricting access to specific AWS accounts, IP addresses, or conditions.

For more information, see Using identity-based policies with Amazon SNS.

AWS Key Management Service (AWS KMS)

AWS KMS enhances the security of Amazon SNS by enabling server-side encryption (SSE) for message confidentiality. With AWS KMS, you can:

- **Encrypt Amazon SNS messages at rest** using AWS-managed or customer-managed encryption keys (CMKs).
- **Control access to Amazon SNS topics** by defining fine-grained key policies that restrict who can publish or subscribe.
- **Ensure compliance with security and regulatory requirements** by auditing key usage through AWS CloudTrail.

For more information, see Managing Amazon SNS encryption keys and costs.

AWS X-Ray

X-Ray provides tracing for Amazon SNS, helping you analyze and debug the flow of messages through your event-driven architecture. With X-Ray, you can:

- **Trace Amazon SNS message delivery** across multiple AWS services, such as Lambda, Amazon SQS, and HTTP/S endpoints.
- **Identify latency bottlenecks** by visualizing how long messages take to be published, delivered, and processed.
- **Detect errors and retries** in Amazon SNS message flows to troubleshoot failed deliveries or slow processing times.

For more information, see Active tracing in Amazon SNS.

Using Amazon SNS with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to Amazon SNS, see Code examples for Amazon SNS using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Create an Amazon SNS topic and publish messages

This topic provides the foundational steps for managing Amazon SNS resources, specifically focusing on topics, subscriptions, and message publishing. First, you will set up the necessary access permissions for Amazon SNS, ensuring that you have the correct permissions to create and manage Amazon SNS resources. Next, you will create a new Amazon SNS topic, which serves as the central hub for managing and delivering messages to subscribers. After creating the topic, you will proceed to create a subscription to this topic, allowing specific endpoints to receive the messages published to it.

Once the topic and subscription are in place, you will publish a message to the topic, observing how Amazon SNS efficiently delivers the message to all subscribed endpoints. Finally, you will learn how to delete both the subscription and the topic, completing the lifecycle of the Amazon SNS resources you've managed. This approach gives you a clear understanding of the fundamental operations in Amazon SNS, equipping you with the practical skills needed to manage messaging workflows using the Amazon SNS console.

Setting up access for Amazon SNS

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

Create an AWS account and an IAM user

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

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Next steps

Now that you're prepared to work with Amazon SNS, get started by:

1. Creating an Amazon SNS topic
2. Creating a subscription to an Amazon SNS topic
3. Publishing an Amazon SNS message
4. Deleting an Amazon SNS topic and subscription

Creating an Amazon SNS topic

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

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

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

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

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

To create a topic using the AWS Management Console

Creating a topic in Amazon SNS establishes the foundation for message distribution, enabling you to publish messages that can be fanned out to multiple subscribers. This step is essential to configure the topic's type, encryption settings, and access policies, ensuring the topic meets the organization’s security, compliance, and operational requirements.

1. Sign in to the Amazon SNS console.
2. Do one of the following:
   - If no topics have ever been created under your AWS account before, read the description of Amazon SNS on the home page.
   - If topics have been created under your AWS account before, on the navigation panel, choose **Topics**.
3. On the **Topics** page, choose **Create topic**.
4. On the **Create topic** page, in the **Details** section, do the following:
   a. For **Type**, choose a topic type (**Standard** or **FIFO**).
   b. Enter a **Name** for the topic. For a FIFO topic, add **.fifo** to the end of the name.
   c. (Optional) Enter a **Display name** for the topic.

> ⚠ **Important**
>
> When subscribing to an email endpoint, the combined character count for the Amazon SNS topic display name and the sending email address (for example, no-reply@sns.amazonaws.com) must not exceed 320 UTF-8 characters. You can use a third party encoding tool to verify the length of the sending address before configuring a display name for your Amazon SNS topic.

   d. (Optional) For a FIFO topic, you can choose **content-based message deduplication** to enable default message deduplication. For more information, see Amazon SNS message deduplication for FIFO topics.

5. (Optional) Expand the **Encryption** section and do the following. For more information, see Securing Amazon SNS data with server-side encryption.

   a. Choose **Enable encryption**.

   b. Specify the AWS KMS key. For more information, see Key terms.

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

> ⚠ **Important**
>
> If you aren't the owner of the KMS, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the KMS on the Amazon SNS console.
> Ask the owner of the KMS to grant you these permissions. For more information, see the AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*.

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

> ⓘ **Note**
>
> Keep the following in mind:
>
> - The first time you use the AWS Management Console to specify the AWS managed KMS for Amazon SNS for a topic, AWS KMS creates the AWS managed KMS for Amazon SNS.

> - Alternatively, the first time you use the `Publish` action on a topic with SSE enabled, AWS KMS creates the AWS managed KMS for Amazon SNS.

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

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

- To use a custom KMS ARN from your AWS account or from another AWS account, enter it into the **KMS key** field.

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

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

7. (Optional) To configure how Amazon SNS retries failed message delivery attempts, expand the **Delivery retry policy (HTTP/S)** section. For more information, see Amazon SNS message delivery retries.
8. (Optional) To configure how Amazon SNS logs the delivery of messages to CloudWatch, expand the **Delivery status logging** section. For more information, see Amazon SNS message delivery status.
9. (Optional) To add metadata tags to the topic, expand the **Tags** section, enter a **Key** and a **Value** (optional) and choose **Add tag**. For more information, see Amazon SNS topic tagging.
10. Choose **Create topic**.

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

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

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

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

To create a topic using an AWS SDK

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

The following code examples show how to use `CreateTopic`.

.NET

SDK for .NET

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

Create a topic with a specific name.

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

    /// <summary>
    /// This example shows how to use Amazon Simple Notification Service
    /// (Amazon SNS) to add a new Amazon SNS topic.
    /// </summary>
    public class CreateSNSTopic
    {
        public static async Task Main()
        {
            string topicName = "ExampleSNSTopic";

            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();
```

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

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

            var response = await client.CreateTopicAsync(request);

            return response.TopicArn;
        }
    }
```

Create a new topic with a name and specific FIFO and de-duplication attributes.

```
    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }
```

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

C++

SDK for C++

Note

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

```
//! Create an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param topicName: An Amazon SNS topic name.
  \param topicARNResult: String to return the Amazon Resource Name (ARN) for the
 topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::createTopic(const Aws::String &topicName,
                              Aws::String &topicARNResult,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::CreateTopicRequest request;
    request.SetName(topicName);

    const Aws::SNS::Model::CreateTopicOutcome outcome =
 snsClient.CreateTopic(request);

    if (outcome.IsSuccess()) {
        topicARNResult = outcome.GetResult().GetTopicArn();
        std::cout << "Successfully created an Amazon SNS topic " << topicName
                  << " with topic ARN '" << topicARNResult
                  << "'." << std::endl;

    }
    else {
        std::cerr << "Error creating topic " << topicName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
        topicARNResult.clear();
    }

    return outcome.IsSuccess();
}
```

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

CLI

AWS CLI

To create an SNS topic

The following `create-topic` example creates an SNS topic named `my-topic`.

```
aws sns create-topic \
    --name my-topic
```

Output:

```
{
    "ResponseMetadata": {
        "RequestId": "1469e8d7-1642-564e-b85d-a19b4b341f83"
    },
    "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
}
```

For more information, see Using the AWS Command Line Interface with Amazon SQS and Amazon SNS in the *AWS Command Line Interface User Guide*.

- For API details, see CreateTopic in *AWS CLI Command Reference*.

Go

SDK for Go V2

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

```

import (
 "context"
 "encoding/json"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}
```

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

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicName>

                Where:
                   topicName - The name of the topic to create (for example,
 mytopic).

                """;

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

        String topicName = args[0];
        System.out.println("Creating a topic with name: " + topicName);
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnVal = createSNSTopic(snsClient, topicName);
        System.out.println("The topic ARN is" + arnVal);
        snsClient.close();
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                    .name(topicName)
                    .build();

            result = snsClient.createTopic(request);
            return result.topicArn();

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

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

Kotlin

SDK for Kotlin

Note

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

```
suspend fun createSNSTopic(topicName: String): String {
    val request =
        CreateTopicRequest {
            name = topicName
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}
```

- For API details, see CreateTopic in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

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

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

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

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

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

$topicname = 'myTopic';

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

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

Python

SDK for Python (Boto3)

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


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

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

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

Ruby

SDK for Ruby

Note

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

```
# This class demonstrates how to create an Amazon Simple Notification Service
 (SNS) topic.
class SNSTopicCreator
  # Initializes an SNS client.
  #
  # Utilizes the default AWS configuration for region and credentials.
  def initialize
    @sns_client = Aws::SNS::Client.new
  end

  # Attempts to create an SNS topic with the specified name.
  #
  # @param topic_name [String] The name of the SNS topic to create.
  # @return [Boolean] true if the topic was successfully created, false
 otherwise.
  def create_topic(topic_name)
    @sns_client.create_topic(name: topic_name)
    puts "The topic '#{topic_name}' was successfully created."
    true
  rescue Aws::SNS::Errors::ServiceError => e
    # Handles SNS service errors gracefully.
    puts "Error while creating the topic named '#{topic_name}': #{e.message}"
    false
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_name = 'YourTopicName' # Replace with your topic name
  sns_topic_creator = SNSTopicCreator.new

  puts "Creating the topic '#{topic_name}'..."
  unless sns_topic_creator.create_topic(topic_name)
    puts 'The topic was not created. Stopping program.'
    exit 1
  end
end
```

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

Rust

SDK for Rust

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

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

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

    Ok(())
}
```

- For API details, see CreateTopic in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

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

```
    TRY.
        oo_result = lo_sns->createtopic( iv_name = iv_topic_name ). " oo_result
 is returned for testing purposes. "
        MESSAGE 'SNS topic created' TYPE 'I'.
      CATCH /aws1/cx_snstopiclimitexcdex.
```

```
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreateTopic in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

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

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.createTopic(
            input: CreateTopicInput(name: name)
        )

        guard let arn = output.topicArn else {
            print("No topic ARN returned by Amazon SNS.")
            return
        }
```

- For API details, see CreateTopic in *AWS SDK for Swift API reference*.

Creating a subscription to an Amazon SNS topic

To receive messages published to a topic, you must *subscribe* an endpoint to the topic. When you subscribe an endpoint to a topic, the endpoint begins to receive messages published to the associated topic.

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

To subscribe an endpoint to an Amazon SNS topic

Subscribing an endpoint to an Amazon SNS topic enables message delivery to the specified endpoint, ensuring the right systems or users receive notifications when a message is published to the topic. This step is essential for linking the topic to consumers—whether they are applications, email recipients, or other services—allowing for seamless communication across systems.

1. Sign in to the Amazon SNS console.
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:

    a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a topic. This value is the AWS ARN that was generated when you created the Amazon SNS topic, for example `arn:aws:sns:us-east-2:123456789012:your_topic`.

    b. For **Protocol**, choose an endpoint type. The available endpoint types are:

    - **HTTP/HTTPS**
    - **Email/Email-JSON**
    -
    - **Amazon SQS**

        > **Note**
        >
        > To subscribe to an SNS FIFO topic, choose this option.

    - **AWS Lambda**
    - **Platform application endpoint**
    - **SMS**

c. For **Endpoint**, enter the endpoint value, such as an email address or the ARN of an Amazon SQS queue.

d. Firehose endpoints only: For **Subscription role ARN**, specify the ARN of the IAM role that you created for writing to Firehose delivery streams. For more information, see Prerequisites for subscribing Firehose delivery streams to Amazon SNS topics.

e. (Optional) For Firehose, Amazon SQS, HTTP/S endpoints, you can also enable raw message delivery. For more information, see Amazon SNS raw message delivery.

f. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see Amazon SNS subscription filter policies.

g. (Optional) To enable payload-based filtering, configure `Filter Policy Scope` to `MessageBody`. For more information, see Amazon SNS subscription filter policy scope.

h. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see Amazon SNS dead-letter queues.

i. Choose **Create subscription**.

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

Publishing an Amazon SNS message

After you create an Amazon SNS topic and subscribe an endpoint to it, you can *publish* messages to the topic. When a message is published, Amazon SNS attempts to deliver the message to the subscribed endpoints.

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

1. Sign in to the Amazon SNS console.
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic, and then choose **Publish message**.

   The console opens the **Publish message to topic** page.
4. In the **Message details** section, do the following:

   a. (Optional) Enter a message **Subject**.

b. For a FIFO topic, enter a **Message group ID**. **Message group ID** is required for FIFO topics. Messages in the same message group are delivered in the order that they are published.

c. (Optional)For a standard topic, enter a **Message group ID**. This ID is forwarded to all Amazon SQS standard subscriptions and is not used for, or sent to, any other endpoint types.

d. For a FIFO topic, enter a **Message deduplication ID**. This ID is optional if you enabled the **Content-based message deduplication** setting for the topic.

e. (Optional) For mobile push notifications, enter a **Time to Live (TTL)** value in seconds. This is the amount of time that a push notification service—such as Apple Push Notification Service (APNs) or Firebase Cloud Messaging (FCM)—has to deliver the message to the endpoint.

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

   a. Choose **Identical payload for all delivery protocols**, and then enter a message.

   b. Choose **Custom payload for each delivery protocol**, and then enter a JSON object to define the message to send for each delivery protocol.

      For more information, see Publishing Amazon SNS notifications with platform-specific payloads.

6. In the **Message attributes** section, add any attributes that you want Amazon SNS to match with the subscription attribute `FilterPolicy` to decide whether the subscribed endpoint is interested in the published message.

   a. For **Type**, choose an attribute type, such as **String.Array**.

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

   b. Enter an attribute **Name**, such as `customer_interests`.

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

If the attribute type is **String**, **String.Array**, or **Number**, Amazon SNS evaluates the message attribute against a subscription's filter policy (if present) before sending the message to the subscription given filter policy scope is not explicitly set to `MessageBody`.

For more information, see Amazon SNS message attributes.

7. Choose **Publish message**.

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

To publish a message to a topic using an AWS SDK

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

The following code examples show how to use `Publish`.

.NET

SDK for .NET

Note

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

Publish a message to a topic.

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

/// <summary>
/// This example publishes a message to an Amazon Simple Notification
/// Service (Amazon SNS) topic.
/// </summary>
public class PublishToSNSTopic
{
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-
east-2:000000000000:ExampleSNSTopic";
            string messageText = "This is an example message to publish to the
 ExampleSNSTopic.";

            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            await PublishToTopicAsync(client, topicArn, messageText);
        }

        /// <summary>
        /// Publishes a message to an Amazon SNS topic.
        /// </summary>
        /// <param name="client">The initialized client object used to publish
        /// to the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic.</param>
        /// <param name="messageText">The text of the message.</param>
        public static async Task PublishToTopicAsync(
            IAmazonSimpleNotificationService client,
            string topicArn,
            string messageText)
        {
            var request = new PublishRequest
            {
                TopicArn = topicArn,
                Message = messageText,
            };

            var response = await client.PublishAsync(request);

            Console.WriteLine($"Successfully published message ID:
 {response.MessageId}");
        }
    }
```

Publish a message to a topic with group, duplication, and attribute options.

```
    /// <summary>
```

```
   /// Publish messages using user settings.
   /// </summary>
   /// <returns>Async task.</returns>
   public static async Task PublishMessages()
   {
       Console.WriteLine("Now we can publish messages.");

       var keepSendingMessages = true;
       string? deduplicationId = null;
       string? toneAttribute = null;
       while (keepSendingMessages)
       {
           Console.WriteLine();
           var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

           if (_useFifoTopic)
           {
               Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                 "\r\nAll messages within the same group will be
received in the order " +
                                 "they were published.");

               Console.WriteLine();
               var messageGroupId = GetUserResponse("Enter a message group ID
for this message:", "1");

               if (!_useContentBasedDeduplication)
               {
                   Console.WriteLine("Because you are not using content-based
deduplication, " +
                                     "you must enter a deduplication ID.");

                   Console.WriteLine("Enter a deduplication ID for this
message.");
                   deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
               }

               if (GetYesNoResponse("Add an attribute to this message?"))
               {
                   Console.WriteLine("Enter a number for an attribute.");
                   for (int i = 0; i < _tones.Length; i++)
```

```
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
 messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?",
 false);
        }
    }
```

Apply the user's selections to the publish action.

```
    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
 message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await
_amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
```

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

C++

SDK for C++

Note

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

```
//! Send a message to an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param message: The message to publish.
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::publishToTopic(const Aws::String &message,
                                 const Aws::String &topicARN,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with id '"
                  << outcome.GetResult().GetMessageId() << "'." << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

Publish a message with an attribute.

```
        static const Aws::String TONE_ATTRIBUTE("tone");
        static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
 "serious",
                                                       "sincere"};

        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::SNS::SNSClient snsClient(clientConfiguration);

        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);

        if (filteringMessages && askYesNoQuestion(
                "Add an attribute to this message? (y/n) ")) {
            for (size_t i = 0; i < TONES.size(); ++i) {
                std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
            }
            int selection = askQuestionForIntRange(
                    "Enter a number for an attribute. ",
                    1, static_cast<int>(TONES.size()));
            Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
            messageAttributeValue.SetDataType("String");
            messageAttributeValue.SetStringValue(TONES[selection - 1]);
            request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
        }

        Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

        if (outcome.IsSuccess()) {
            std::cout << "Your message was successfully published." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Publish. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }
```

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

CLI

AWS CLI

Example 1: To publish a message to a topic

The following publish example publishes the specified message to the specified SNS topic. The message comes from a text file, which enables you to include line breaks.

```
aws sns publish \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic" \
    --message file://message.txt
```

Contents of message.txt:

```
Hello World
Second Line
```

Output:

```
{
    "MessageId": "123a45b6-7890-12c3-45d6-111122223333"
}
```

Example 2: To publish an SMS message to a phone number

The following publish example publishes the message Hello world! to the phone number +1-555-555-0100.

```
aws sns publish \
    --message "Hello world!" \
    --phone-number +1-555-555-0100
```

Output:

```
{
    "MessageId": "123a45b6-7890-12c3-45d6-333322221111"
}
```

- For API details, see Publish in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

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

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
```

```
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

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

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <message> <topicArn>

                Where:
                   message - The message text to send.
                   topicArn - The ARN of the topic to publish.
                """;

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

        String message = args[0];
        String topicArn = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTopic(snsClient, message, topicArn);
        snsClient.close();
    }

    public static void pubTopic(SnsClient snsClient, String message, String
 topicArn) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .topicArn(topicArn)
                    .build();

            PublishResponse result = snsClient.publish(request);
            System.out
```

```
                    .println(result.messageId() + " Message sent. Status is " +
 result.sdkHttpResponse().statusCode());

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Publish a message to a topic with group, duplication, and attribute options.

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );
    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

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

Kotlin

SDK for Kotlin

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

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

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

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

PHP

SDK for PHP

Note

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

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

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



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

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

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

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

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows publishing a message with a single MessageAttribute declared inline.

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
 @{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='Strin
 StringValue ='AnyCity'}}
```

Example 2: This example shows publishing a message with multiple MessageAttributes declared in advance.

```
$cityAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- For API details, see Publish in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows publishing a message with a single MessageAttribute declared inline.

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
 @{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='Strin
 StringValue ='AnyCity'}}
```

Example 2: This example shows publishing a message with multiple MessageAttributes declared in advance.

```
$cityAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- For API details, see Publish in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

Note

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

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def publish_message(topic, message, attributes):
        """
        Publishes a message, with attributes, to a topic. Subscriptions can be
 filtered
        based on message attributes so that a subscription receives messages only
        when specified attributes are present.

        :param topic: The topic to publish to.
        :param message: The message to publish.
        :param attributes: The key-value attributes to attach to the message.
 Values
                           must be either `str` or `bytes`.
        :return: The ID of the message.
        """
        try:
            att_dict = {}
            for key, value in attributes.items():
                if isinstance(value, str):
                    att_dict[key] = {"DataType": "String", "StringValue": value}
                elif isinstance(value, bytes):
                    att_dict[key] = {"DataType": "Binary", "BinaryValue": value}
            response = topic.publish(Message=message, MessageAttributes=att_dict)
            message_id = response["MessageId"]
            logger.info(
                "Published message with attributes %s to topic %s.",
                attributes,
                topic.arn,
            )
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id
```

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource



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

        :param topic: The topic to publish to.
        :param subject: The subject of the message.
        :param default_message: The default version of the message. This version
 is
```

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

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

Ruby

SDK for Ruby

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

```
# Service class for sending messages using Amazon Simple Notification Service
 (SNS)
class SnsMessageSender
  # Initializes the SnsMessageSender with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Sends a message to a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param message [String] The message to send
  # @return [Boolean] true if message was successfully sent, false otherwise
  def send_message(topic_arn, message)
    @sns_client.publish(topic_arn: topic_arn, message: message)
    @logger.info("Message sent successfully to #{topic_arn}.")
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while sending the message: #{e.message}")
    false
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'SNS_TOPIC_ARN' # Should be replaced with a real topic ARN
  message = 'MESSAGE'         # Should be replaced with the actual message
 content

  sns_client = Aws::SNS::Client.new
  message_sender = SnsMessageSender.new(sns_client)

  @logger.info('Sending message.')
  unless message_sender.send_message(topic_arn, message)
    @logger.error('Message sending failed. Stopping program.')
    exit 1
  end
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see Publish in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

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

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

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

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

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

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

    Ok(())
}
```

- For API details, see Publish in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

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

```
   TRY.
       oo_result = lo_sns->publish(              " oo_result is returned for
testing purposes. "
         iv_topicarn = iv_topic_arn
         iv_message = iv_message ).
       MESSAGE 'Message published to SNS topic.' TYPE 'I'.
     CATCH /aws1/cx_snsnotfoundexception.
       MESSAGE 'Topic does not exist.' TYPE 'E'.
   ENDTRY.
```

- For API details, see Publish in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

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

```
import AWSSNS
```

```
        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.publish(
            input: PublishInput(
                message: message,
                topicArn: arn
            )
        )

        guard let messageId = output.messageId else {
            print("No message ID received from Amazon SNS.")
            return
        }

        print("Published message with ID \(messageId)")
```

- For API details, see Publish in *AWS SDK for Swift API reference*.

Publishing large messages with Amazon SNS and Amazon S3

To publish large Amazon SNS messages, you can use the Amazon SNS Extended Client Library for Java, or the Amazon SNS Extended Client Library for Python. These libraries are useful for messages that are larger than the current maximum of 256 KB, with a maximum of 2 GB. Both libraries save the actual payload to an Amazon S3 bucket, and publish the reference of the stored Amazon S3 object to the Amazon SNS topic. Subscribed Amazon SQS queues can use the Amazon SQS Extended Client Library for Java to de-reference and retrieve payloads from Amazon S3. Other endpoints such as Lambda can use the Payload Offloading Java Common Library for AWS to de-reference and retrieve the payload.

> **Note**
>
> The Amazon SNS Extended Client Libraries are compatible with both standard and FIFO topics.

Amazon SNS Extended Client Library for Java

Prerequisites

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

- An AWS SDK. The example on this page uses the AWS Java SDK. To install and set up the SDK, see Set up the AWS SDK for Java in the *AWS SDK for Java Developer Guide*.
- An AWS account with the proper credentials. To create an AWS account, navigate to the AWS home page, and then choose **Create an AWS Account**. Follow the instructions.

  For information about credentials, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.
- Java 8 or better.
- The Amazon SNS Extended Client Library for Java (also available from Maven).

Configuring message storage

The Amazon SNS Extended Client library uses the Payload Offloading Java Common Library for AWS for message storage and retrieval. You can configure the following Amazon S3 message storage options:

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

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

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

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

The following code example shows how to:

- Create a sample topic and queue.

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

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

SDK for Java 1.x

Note

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

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

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

public class Example {

        public static void main(String[] args) {
                final String BUCKET_NAME = "extended-client-bucket";
                final String TOPIC_NAME = "extended-client-topic";
```

```
               final String QUEUE_NAME = "extended-client-queue";
               final Regions region = Regions.DEFAULT_REGION;

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

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

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

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

subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
               subscriptionAttributesRequest.setAttributeValue("TRUE");
               snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);
```

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

 .withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
                final AmazonSNSExtendedClient snsExtendedClient = new
 AmazonSNSExtendedClient(snsClient,
                                snsExtendedClientConfiguration);

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

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

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

Other endpoint protocols

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

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

```
[
  "software.amazon.payloadoffloading.PayloadS3Pointer",
  {
    "s3BucketName": "extended-client-bucket",
    "s3Key": "xxxx-xxxxx-xxxxx-xxxxxx"
  }
]
```

Amazon SNS Extended Client Library for Python

Prerequisites

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

- An AWS SDK. The example on this page uses AWS Python SDK Boto3. To install and set up the SDK, see the *AWS SDK for Python* documentation.
- An AWS account with the proper credentials. To create an AWS account, navigate to the AWS home page, and then choose **Create an AWS Account**. Follow the instructions.

  For information about credentials, see Credentials in the *AWS SDK for Python Developer Guide*.
- Python 3.x (or later) and pip.
- The Amazon SNS Extended Client Library for Python (also available from PyPI).

Configuring message storage

The below attributes are available on Boto3 Amazon SNS Client, Topic, and PlatformEndpoint objects to configure the Amazon S3 message storage options.

- **`large_payload_support`** – The Amazon S3 bucket name that will store large messages.
- **`use_legacy_attribute`** – If `True`, then all published messages use the Legacy reserved message attribute (`SQSLargePayloadSize`) instead of the current reserved message attribute (`ExtendedPayloadSize`).

- **`message_size_threshold`** – The threshold for storing the message in the large messages bucket. Cannot be less than `0`, or greater than `262144`. The default is `262144`.
- **`always_through_s3`** – If `True`, then all messages are stored in Amazon S3. The default is `False`.
- **`s3_client`** – The Boto3 Amazon S3 `client` object to use to store objects to Amazon S3. Use this if you want to control the Amazon S3 client (for example, custom Amazon S3 config or credentials). Defaults to `boto3.client("s3")` on first use if not previously set.

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

The following code example shows how to:

- Create a sample Amazon SNS topic and Amazon SQS queue.
- Attach the policy to the Amazon SQS queue to receive the message from Amazon SNS topic.
- Subscribe the queue to receive messages from the topic.
- Publish a test message using the Amazon SNS extended client, Topic resource, and PlatformEndpoint resource.
- The message payload is stored in Amazon S3, and the reference to it is published.
- Print the published message from the queue along with the original message retrieved from Amazon S3.

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

```
import boto3
from sns_extended_client import SNSExtendedClientSession
from json import loads

s3_extended_payload_bucket = "extended-client-bucket-store"  # S3 bucket with the
 given bucket name is a resource which is created and accessible with the given AWS
 credentials
TOPIC_NAME = "---TOPIC-NAME---"
QUEUE_NAME = "---QUEUE-NAME---"

def allow_sns_to_write_to_sqs(topicarn, queuearn):
    policy_document = """{{
        "Version":"2012-10-17",
        "Statement":[
            {{
            "Sid":"MyPolicy",
            "Effect":"Allow",
            "Principal" : {{"AWS" : "*"}},
            "Action":"SQS:SendMessage",
            "Resource": "{}",
            "Condition":{{
                "ArnEquals":{{
                "aws:SourceArn": "{}"
                }}
            }}
            }}
        ]
        }}""".format(queuearn, topicarn)

    return policy_document

def get_msg_from_s3(body,sns_extended_client):
    """Handy Helper to fetch message from S3"""
    json_msg = loads(body)
    s3_object = sns_extended_client.s3_client.get_object(
        Bucket=json_msg[1].get("s3BucketName"), Key=json_msg[1].get("s3Key")
    )
    msg = s3_object.get("Body").read().decode()
    return msg


def fetch_and_print_from_sqs(sqs, queue_url,sns_extended_client):
    sqs_msg = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=['All'],
        MessageAttributeNames=['All'],
        VisibilityTimeout=0,
        WaitTimeSeconds=0,
        MaxNumberOfMessages=1
    ).get("Messages")[0]

    message_body = sqs_msg.get("Body")
    print("Published Message: {}".format(message_body))
    print("Message Stored in S3 Bucket is:
 {}\n".format(get_msg_from_s3(message_body,sns_extended_client)))

    # Delete the Processed Message
    sqs.delete_message(
        QueueUrl=queue_url,
        ReceiptHandle=sqs_msg['ReceiptHandle']
    )


sns_extended_client = boto3.client("sns", region_name="us-east-1")
create_topic_response = sns_extended_client.create_topic(Name=TOPIC_NAME)
sns_topic_arn = create_topic_response.get("TopicArn")

# create and subscribe an sqs queue to the sns client
sqs = boto3.client("sqs",region_name="us-east-1")
demo_queue_url = sqs.create_queue(QueueName=QUEUE_NAME).get("QueueUrl")
sqs_queue_arn = sqs.get_queue_attributes(
    QueueUrl=demo_queue_url, AttributeNames=["QueueArn"]
)["Attributes"].get("QueueArn")

# Adding policy to SQS queue such that SNS topic can send msg to SQS queue
policy_json = allow_sns_to_write_to_sqs(sns_topic_arn, sqs_queue_arn)
response = sqs.set_queue_attributes(
    QueueUrl = demo_queue_url,
    Attributes = {
        'Policy' : policy_json
    }
)

# Set the RawMessageDelivery subscription attribute to TRUE if you want to use
# SQSExtendedClient to help with retrieving msg from S3
sns_extended_client.subscribe(TopicArn=sns_topic_arn, Protocol="sqs",
Endpoint=sqs_queue_arn
, Attributes={"RawMessageDelivery":"true"}
)

sns_extended_client.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of sns_extended_client to use 'us-east-1' region
sns_extended_client.s3_client = boto3.client("s3", region_name="us-east-1")


# Below is the example that all the messages will be sent to the S3 bucket
sns_extended_client.always_through_s3 = True
sns_extended_client.publish(
    TopicArn=sns_topic_arn, Message="This message should be published to S3"
)
print("\n\nPublished using SNS extended client:")
```

```
fetch_and_print_from_sqs(sqs, demo_queue_url,sns_extended_client)  # Prints message
 stored in s3

# Below is the example that all the messages larger than 32 bytes will be sent to the
 S3 bucket
print("\nUsing decreased message size threshold:")

sns_extended_client.always_through_s3 = False
sns_extended_client.message_size_threshold = 32
sns_extended_client.publish(
    TopicArn=sns_topic_arn,
    Message="This message should be published to S3 as it exceeds the limit of the 32
 bytes",
)

fetch_and_print_from_sqs(sqs, demo_queue_url,sns_extended_client)  # Prints message
 stored in s3


# Below is the example to publish message using the SNS.Topic resource
sns_extended_client_resource = SNSExtendedClientSession().resource(
    "sns", region_name="us-east-1"
)

topic = sns_extended_client_resource.Topic(sns_topic_arn)
topic.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of topic to use 'us-east-1' region
topic.s3_client = boto3.client("s3", region_name="us-east-1")

topic.always_through_s3 = True
# Can Set custom S3 Keys to be used to store objects in S3
topic.publish(
    Message="This message should be published to S3 using the topic resource",
    MessageAttributes={
        "S3Key": {
            "DataType": "String",
            "StringValue": "347c11c4-a22c-42e4-a6a2-9b5af5b76587",
        }
    },
)
print("\nPublished using Topic Resource:")
fetch_and_print_from_sqs(sqs, demo_queue_url,topic)
```

```
# Below is the example to publish message using the SNS.PlatformEndpoint resource
sns_extended_client_resource = SNSExtendedClientSession().resource(
    "sns", region_name="us-east-1"
)

platform_endpoint = sns_extended_client_resource.PlatformEndpoint(sns_topic_arn)
platform_endpoint.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of platform_endpoint to use 'us-east-1' region
platform_endpoint.s3_client = boto3.client("s3", region_name="us-east-1")

platform_endpoint.always_through_s3 = True
# Can Set custom S3 Keys to be used to store objects in S3
platform_endpoint.publish(
    Message="This message should be published to S3 using the PlatformEndpoint
 resource",
    MessageAttributes={
        "S3Key": {
            "DataType": "String",
            "StringValue": "247c11c4-a22c-42e4-a6a2-9b5af5b76587",
        }
    },
)
print("\nPublished using PlatformEndpoint Resource:")
fetch_and_print_from_sqs(sqs, demo_queue_url,platform_endpoint)
```

Output

```
Published using SNS extended client:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
 {"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3

Using decreased message size threshold:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
 {"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 as it exceeds
 the limit of the 32 bytes

Published using Topic Resource:
```

```
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
 {"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 using the topic
 resource

Published using PlatformEndpoint Resource:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
 {"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 using the
 PlatformEndpoint resource
```

Amazon SNS message attributes

Amazon SNS supports delivery of message attributes, which let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) about the message. For SQS subscriptions, a maximum of 10 message attributes can be sent when Raw Message Delivery is enabled. To send more than 10 message attributes, Raw Message Delivery must be disabled. Messages with more than 10 message attributes directed towards Raw Message Delivery enabled Amazon SQS subscriptions will be discarded as client side errors.

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

For information about sending messages with attributes using the AWS Management Console or the AWS SDK for Java, see the To publish messages to Amazon SNS topics using the AWS Management Console tutorial.

Note

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

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

You can also use message attributes to make your messages filterable using subscription filter policies. You can apply filter policies to topic subscriptions. When a filter policy is applied with filter policy scope set to `MessageAttributes` (default), a subscription receives only those messages that have attributes that the policy accepts. For more information, see Amazon SNS message filtering.

> **Note**
>
> When message attributes are used for filtering, the value must be a valid JSON string. Doing this ensures that the message is delivered to a subscription with message attributes filtering enabled.

Message attribute items and validation

Each message attribute consists of the following items:

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

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

Message attribute data types and validation

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

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

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

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

- **String.Array** – An array, formatted as a string, that can contain multiple values. The values can be strings, numbers, or the keywords `true`, `false`, and `null`. A String.Array of number or boolean type does not require quotes. Multiple String.Array values are separated by commas.

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

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

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

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

Reserved message attributes for mobile push notifications

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

Push notification service	Reserved message attribute
ADM	`AWS.SNS.MOBILE.ADM.TTL`
APNs[1]	`AWS.SNS.MOBILE.APNS_MDM.TTL`
	`AWS.SNS.MOBILE.APNS_MDM_SANDBOX.TTL`
	`AWS.SNS.MOBILE.APNS_PASSBOOK.TTL`
	`AWS.SNS.MOBILE.APNS_PASSBOOK_SANDBOX.TTL`
	`AWS.SNS.MOBILE.APNS_SANDBOX.TTL`
	`AWS.SNS.MOBILE.APNS_VOIP.TTL`
	`AWS.SNS.MOBILE.APNS_VOIP_SANDBOX.TTL`
	`AWS.SNS.MOBILE.APNS.COLLAPSE_ID`
	`AWS.SNS.MOBILE.APNS.PRIORITY`
	`AWS.SNS.MOBILE.APNS.PUSH_TYPE`
	`AWS.SNS.MOBILE.APNS.TOPIC`
	`AWS.SNS.MOBILE.APNS.TTL`
Baidu	`AWS.SNS.MOBILE.BAIDU.DeployStatus`
	`AWS.SNS.MOBILE.BAIDU.MessageKey`
	`AWS.SNS.MOBILE.BAIDU.MessageType`
	`AWS.SNS.MOBILE.BAIDU.TTL`
FCM	`AWS.SNS.MOBILE.FCM.TTL`
	`AWS.SNS.MOBILE.GCM.TTL`
macOS	`AWS.SNS.MOBILE.MACOS_SANDBOX.TTL`

Push notification service	Reserved message attribute
	`AWS.SNS.MOBILE.MACOS.TTL`
MPNS	`AWS.SNS.MOBILE.MPNS.NotificationClass`
	`AWS.SNS.MOBILE.MPNS.TTL`
	`AWS.SNS.MOBILE.MPNS.Type`
WNS	`AWS.SNS.MOBILE.WNS.CachePolicy`
	`AWS.SNS.MOBILE.WNS.Group`
	`AWS.SNS.MOBILE.WNS.Match`
	`AWS.SNS.MOBILE.WNS.SuppressPopup`
	`AWS.SNS.MOBILE.WNS.Tag`
	`AWS.SNS.MOBILE.WNS.TTL`
	`AWS.SNS.MOBILE.WNS.Type`

[1] Apple will reject Amazon SNS notifications if message attributes do not meet their requirements. For additional details, see Sending Notification Requests to APNs on the Apple Developer website.

Amazon SNS message batching

What is message batching?

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

Note

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

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

How does message batching work?

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

Examples

Publishing a batch of 10 messages to a Standard topic

```
// Imports
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishBatchRequest;
import software.amazon.awssdk.services.sns.model.PublishBatchRequestEntry;
import software.amazon.awssdk.services.sns.model.PublishBatchResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToTopic(SnsClient snsClient, String topicArn, int
 batchSize) {
    try {
        // Validate the batch size
        if (batchSize > MAX_BATCH_SIZE) {
            throw new IllegalArgumentException("Batch size cannot exceed " +
 MAX_BATCH_SIZE);
        }

        // Create the batch entries
        List<PublishBatchRequestEntry> entries = IntStream.range(0, batchSize)
                .mapToObj(i -> PublishBatchRequestEntry.builder()
                        .id("id" + i)
                        .message("message" + i)
                        .build())
                .collect(Collectors.toList());

        // Build the batch request
        PublishBatchRequest request = PublishBatchRequest.builder()
                .topicArn(topicArn)
                .publishBatchRequestEntries(entries)
                .build();

        // Publish the batch request
        PublishBatchResponse response = snsClient.publishBatch(request);

        // Handle successful messages
        response.successful().forEach(success -> {
            System.out.println("Successful Batch Id: " + success.id());
            System.out.println("Message Id: " + success.messageId());
        });

        // Handle failed messages
        response.failed().forEach(failure -> {
            System.err.println("Failed Batch Id: " + failure.id());
            System.err.println("Error Code: " + failure.code());
            System.err.println("Sender Fault: " + failure.senderFault());
            System.err.println("Error Message: " + failure.message());
        });

    } catch (SnsException e) {
        // Log and handle exceptions
        System.err.println("SNS Exception: " + e.awsErrorDetails().errorMessage());
    } catch (IllegalArgumentException e) {
        System.err.println("Validation Error: " + e.getMessage());
    }
}
```

Publishing a batch of 10 messages to a FIFO topic

```
// Imports
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishBatchRequest;
import software.amazon.awssdk.services.sns.model.PublishBatchRequestEntry;
import software.amazon.awssdk.services.sns.model.PublishBatchResponse;
import software.amazon.awssdk.services.sns.model.BatchResultErrorEntry;
import software.amazon.awssdk.services.sns.model.SnsException;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToFifoTopic(SnsClient snsClient, String topicArn) {
    try {
        // Create the batch entries to send
        List<PublishBatchRequestEntry> entries = IntStream.range(0, MAX_BATCH_SIZE)
                .mapToObj(i -> PublishBatchRequestEntry.builder()
                        .id("id" + i)
                        .message("message" + i)
                        .messageGroupId("groupId")
                        .messageDeduplicationId("deduplicationId" + i)
                        .build())
                .collect(Collectors.toList());

        // Create the batch request
        PublishBatchRequest request = PublishBatchRequest.builder()
                .topicArn(topicArn)
                .publishBatchRequestEntries(entries)
                .build();

        // Publish the batch request
        PublishBatchResponse response = snsClient.publishBatch(request);

        // Handle the successfully sent messages
        response.successful().forEach(success -> {
            System.out.println("Batch Id for successful message: " + success.id());
            System.out.println("Message Id for successful message: " +
 success.messageId());
```

```
            System.out.println("Sequence Number for successful message: " +
 success.sequenceNumber());
        });

        // Handle the failed messages
        response.failed().forEach(failure -> {
            System.err.println("Batch Id for failed message: " + failure.id());
            System.err.println("Error Code for failed message: " + failure.code());
            System.err.println("Sender Fault for failed message: " +
 failure.senderFault());
            System.err.println("Failure Message for failed message: " +
 failure.message());
        });

    } catch (SnsException e) {
        // Handle any exceptions from the request
        System.err.println("SNS Exception: " + e.awsErrorDetails().errorMessage());
    }
}
```

Deleting an Amazon SNS topic and subscription

When a topic is deleted, its associated subscriptions are deleted asynchronously. While customers can still access these subscriptions, the subscriptions are no longer associated with the topic–even if you recreate the topic using the same name. If a publisher attempts to publish a message to the deleted topic, the publisher will receive an error message indicating that the topic doesn't exist. Similarly, any attempt to subscribe to the deleted topic will also result in an error message. You can't delete a subscription that's pending confirmation. Amazon SNS automatically deletes unconfirmed subscriptions after 48 hours.

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

Deleting an Amazon SNS topic or subscription ensures efficient resource management, preventing unnecessary usage and keeping the Amazon SNS console organized. This step helps avoid potential costs from idle resources and streamlines administration by removing topics or subscriptions that are no longer needed.

To delete a topic using the AWS Management Console

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

   The console deletes the topic.

To delete a subscription using the AWS Management Console

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

   The console deletes the subscription.

To delete a subscription and topic using an AWS SDK

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

The following code examples show how to use `DeleteTopic`.

.NET

SDK for .NET

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

Delete a topic by its topic ARN.

```
    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

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

C++

SDK for C++

Note

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

```
//! Delete an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);
    const Aws::SNS::Model::DeleteTopicOutcome outcome =
 snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting topic " << topicARN << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

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

CLI

AWS CLI

To delete an SNS topic

The following `delete-topic` example deletes the specified SNS topic.

```
aws sns delete-topic \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

This command produces no output.

- For API details, see DeleteTopic in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

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

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

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

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:     <topicArn>

                Where:
                   topicArn - The ARN of the topic to delete.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        System.out.println("Deleting a topic with name: " + topicArn);
        deleteSNSTopic(snsClient, topicArn);
        snsClient.close();
    }

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

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

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

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

Kotlin

SDK for Kotlin

Note

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

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

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

- For API details, see DeleteTopic in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

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

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

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

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

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

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

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

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

Python

SDK for Python (Boto3)

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


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

- For API details, see DeleteTopic in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

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

```
    TRY.
        lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).
        MESSAGE 'SNS topic deleted.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteTopic in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

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

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: arn)
        )
```

- For API details, see DeleteTopic in *AWS SDK for Swift API reference*.

Next steps

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

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

Message ordering and deduplication strategies using Amazon SNS FIFO topics

This topic provides information of the features and functionalities of Amazon SNS FIFO (First-In-First-Out) topics and how they integrate with Amazon SQS FIFO queues. You'll learn how to use these services together to ensure strict message ordering and deduplication, essential for applications that require data consistency. This content covers the specific use cases where Amazon SNS FIFO topics are beneficial, providing insight into scenarios where message order and uniqueness are critical.

You'll also learn about the technical details of message ordering, message grouping, and how these affect message delivery. The message deduplication topic explains the mechanisms that prevent duplicate messages, ensuring that each message is processed only once. Additionally, you'll learn about message filtering, security, and durability, which are important for maintaining the integrity and reliability of your messaging system. This content also includes information on message archiving and replay, offering strategies for managing message histories. Practical code examples are also provided to help you implement these features in your own applications, giving you hands-on experience with Amazon SNS FIFO topics and their integration with Amazon SQS FIFO queues.

High throughput FIFO topics in Amazon SNS

High throughput FIFO topics in Amazon SNS efficiently manage high message throughput while maintaining strict message order, ensuring reliability and scalability for applications processing numerous messages. This solution is ideal for scenarios demanding both high throughput and ordered message delivery. To improve message throughput using high throughput FIFO topics, increasing the number of message groups is recommended. For more information on high throughput message quotas, see Amazon SNS service quotas in the *Amazon Web Services General Reference*.

Use cases for high throughput for Amazon SNS FIFO topics

The following use cases highlight the diverse applications of high throughput FIFO topics, showcasing their effectiveness across industries and scenarios:

- **Real-time data processing:** Applications dealing with real-time data streams, such as event processing or telemetry data ingestion, can benefit from high throughput FIFO topics to handle the continuous influx of messages while preserving their order for accurate analysis.

- **E-commerce order processing:** In e-commerce platforms where maintaining the order of customer transactions is critical, high throughput FIFO topics ensure that orders are delivered sequentially and without delays, even during peak shopping seasons.
- **Financial services:** Financial institutions handling high-frequency trading or transactional data rely on high throughput FIFO topics to process market data and transactions with minimal latency while adhering to strict regulatory requirements for message ordering.
- **Media streaming:** Streaming platforms and media distribution services utilize high throughput FIFO topics to manage the delivery of media files and streaming content, ensuring smooth playback experiences for users while maintaining the correct order of content delivery

Partitions and data distribution for high throughput for Amazon SNS FIFO topics

With high throughput topics, Amazon SNS distributes FIFO topic data across partitions. A partition is an allocation of capacity for a topic that is automatically replicated across multiple Availability Zones within an AWS Region. You don't manage partitions. Instead, Amazon SNS automatically manages partitions on your behalf, based on the ingress rate.

For FIFO topics, Amazon SNS modifies the number of partitions in a topic in the following situations:

- If the current publish rate approaches or exceeds what the existing partitions can support, additional partitions are allocated until the topic reaches the regional quota. For information on quotas, see Amazon SNS service quotas in the *Amazon Web Services General Reference*.
- If the current partitions have low utilization, the number of partitions may be reduced.

Partition management occurs automatically in the background and is transparent to your applications. Your topic and messages are available at all times.

Note

Temporary Publish API throttling may occur if you suddenly and significantly increase traffic to your topic while sending multiple times the usual volume. This throttling can last up to the duration of the deduplication window while the topic scales up to accommodate the increased traffic.

Distributing data by message group IDs

When publishing a message to a FIFO topic, Amazon SNS uses the value of each message’s message group ID as input to an internal hash function. The output value from the hash function determines which partition processes the message, one or more message group IDs may be handled by a given partition.

> **Note**
>
> Amazon SNS is optimized for uniform distribution of items across a FIFO topic's partitions, regardless of the number of partitions. AWS recommends that you use message group IDs that can have a large number of distinct values.

Enable high throughput on your Amazon SNS FIFO topic

By default Amazon SNS FIFO topics are configured for topic-level deduplication, this is controlled by the topic attribute `FifoThroughputScope` set to `Topic` and have more restricted throughput quotas, see Amazon SNS service quotas in the *Amazon Web Services General Reference*.

To enable high throughput for your Amazon SNS FIFO topic, update the `FifoThroughputScope` attribute to `MessageGroup`. This change can be done through the console or using the AWS CLI and SDK, and can also be set during topic creation, which Amazon SNS recommends for the best customer experience and to reduce the chances of your topic being throttled.

> **Important**
>
> Once you've enabled a topic's `FifoThroughputScope` to `MessageGroup`, it cannot be reverted back to the `Topic` throughput.

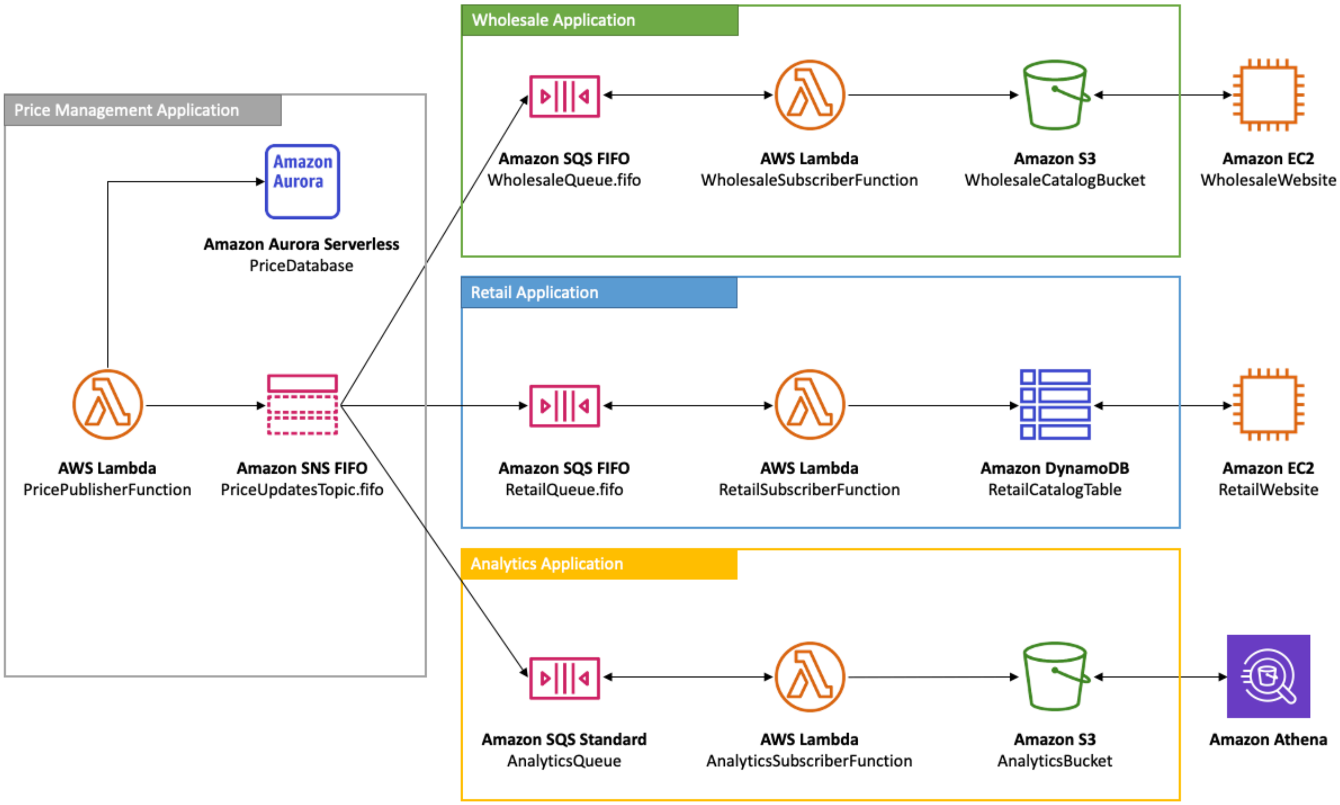
Enable high throughput mode for any subscribed Amazon SQS FIFO queue

When publishing to your Amazon SNS FIFO topic with high throughput enabled, and one or more Amazon SQS FIFO queues are subscribed, it is recommended that you enable high throughput on your Amazon SQS FIFO queues to enable your Amazon SNS FIFO high throughput topic to deliver smoothly. For more see High throughput for FIFO queues in the *Amazon Simple Queue Service Developer Guide*.

Amazon SNS FIFO topic example use case

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

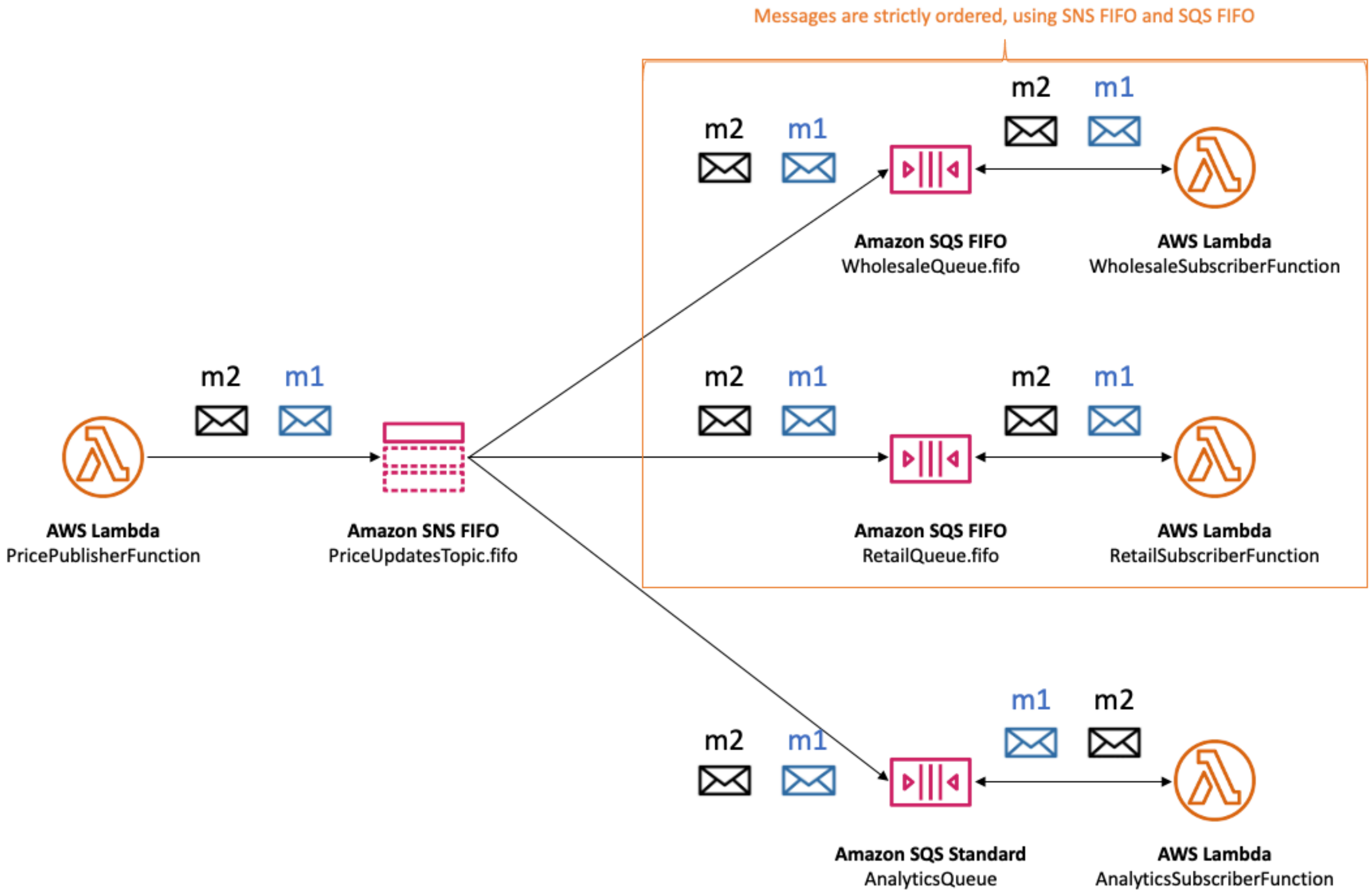
- Inventory managers use a price management application to set the price for each item in stock. At this company, product prices can change based on currency exchange fluctuation, market demand, and shifts in sales strategy. The price management application uses an AWS Lambda function that publishes price updates to an Amazon SNS FIFO topic whenever prices change.
- A wholesale application provides the backend for a website where auto body shops and car manufacturers can buy the company's auto parts in bulk. To get price change notifications, the wholesale application subscribes its Amazon SQS FIFO queue to the price management application's Amazon SNS FIFO topic.
- A retail application provides the backend for another website where car owners and car tuning enthusiasts can purchase individual auto parts for their vehicles. To get price change notifications, the retail application also subscribes its Amazon SQS FIFO queue to the price management application's Amazon SNS FIFO topic.
- An analytics application that aggregates price updates and stores them into an Amazon S3 bucket, enabling Amazon Athena to query the bucket for business intelligence (BI) purposes. To get price change notifications, the analytics application subscribes its Amazon SQS standard queue to the price management application's Amazon SNS FIFO topic. Unlike the other applications, the analytics one doesn't require the price updates to be strictly ordered.



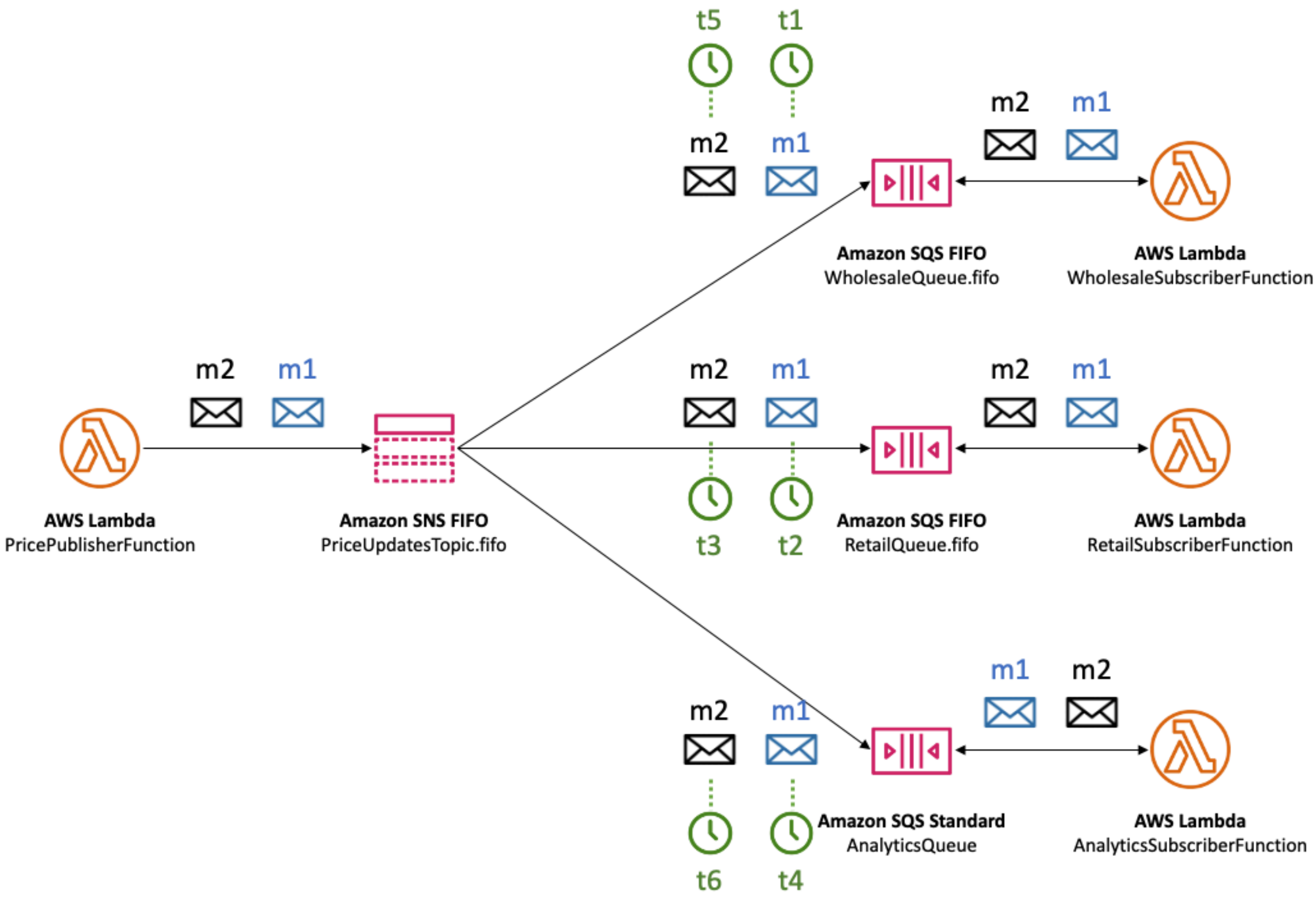
For the wholesale and retail applications to receive price updates in the correct order, the price management application must use a strictly ordered message distribution system. Using Amazon SNS FIFO topics and Amazon SQS FIFO queues enables the processing of messages in order and with no duplication.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

Amazon SNS message ordering details for FIFO topics

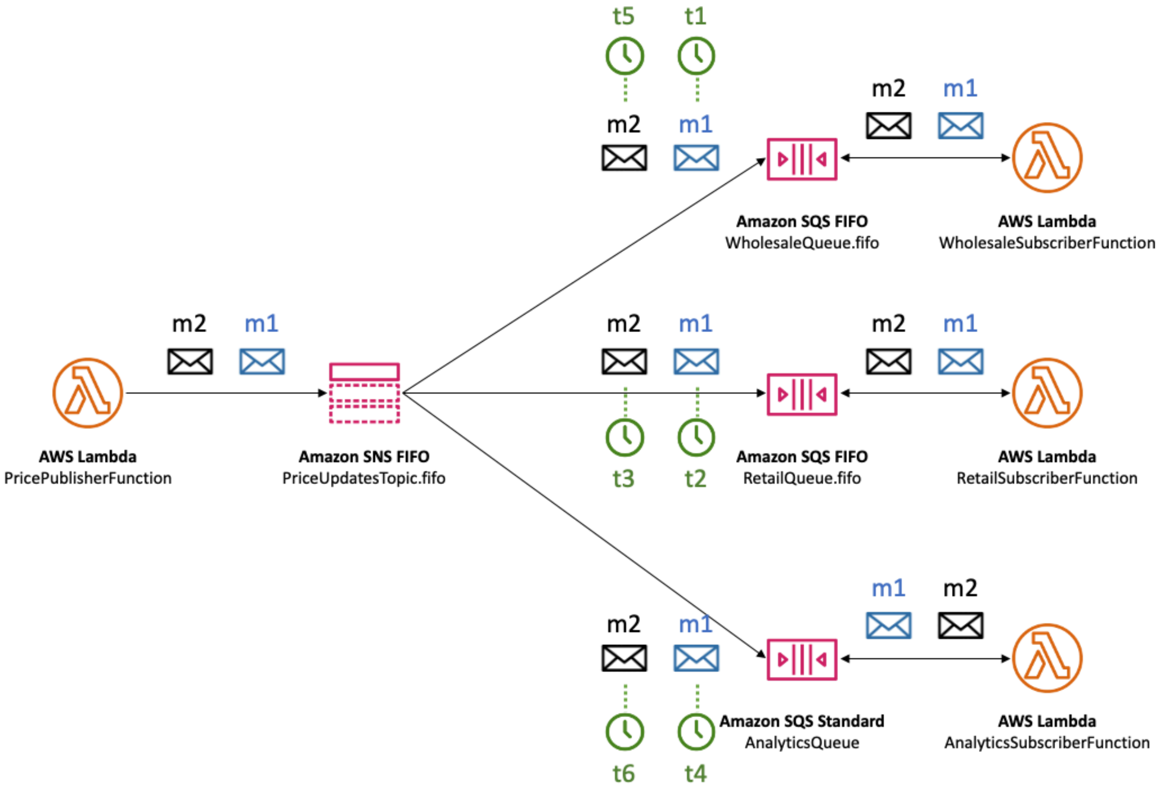
An Amazon SNS FIFO topic always delivers messages to subscribed Amazon SQS queues in the exact order in which the messages are published to the topic, and only once. With an Amazon SQS FIFO queue subscribed, the consumer of the queue receives the messages in the exact order in which the messages are delivered to the queue, and no duplicates. With an Amazon SQS standard queue subscribed, however, the consumer of the queue may receive messages out of order, and more than once. This enables further decoupling of subscribers from publishers, giving subscribers more flexibility in terms of message consumption and cost optimization, as shown in the following diagram, based on the Amazon SNS FIFO topic example use case.



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

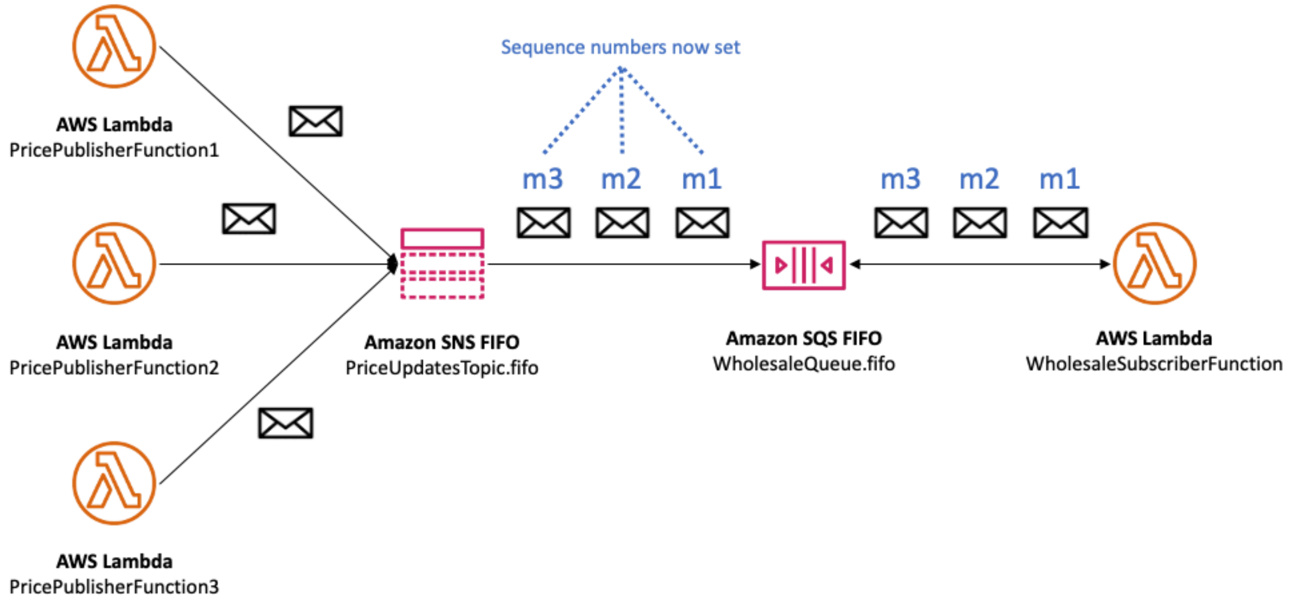


If an Amazon SQS queue subscriber becomes unreachable, it can get out of sync. For example, say the wholesale application queue owner mistakenly changes the Amazon SQS queue policy in a way that prevents the Amazon SNS service principal from delivering messages to the queue. In this case, price update deliveries to the wholesale queue fail, while the ones to the retail and analytics queues succeed, causing the subscribers to be out of sync. When the wholesale application queue owner corrects its queue policy, Amazon SNS resumes delivering messages to the subscribed queue. Any messages published to the topic that target the incorrectly configured queue are dropped, unless the corresponding subscription has a dead-letter queue configured.



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

The sequence number is a large, non-consecutive number that Amazon SNS assigns to each message. The length of the sequence number is 128-bits, and continues to increase for each Message Group. The sequence number is passed to the subscribed Amazon SQS queues as part of the message body. However, if you enable raw message delivery, the message that's delivered to the Amazon SQS queue doesn't include the sequence number or any other Amazon SNS message metadata.



Amazon SNS FIFO topics define ordering in the context of a message group. For more information, see Amazon SNS message grouping for FIFO topics.

Amazon SNS message grouping for FIFO topics

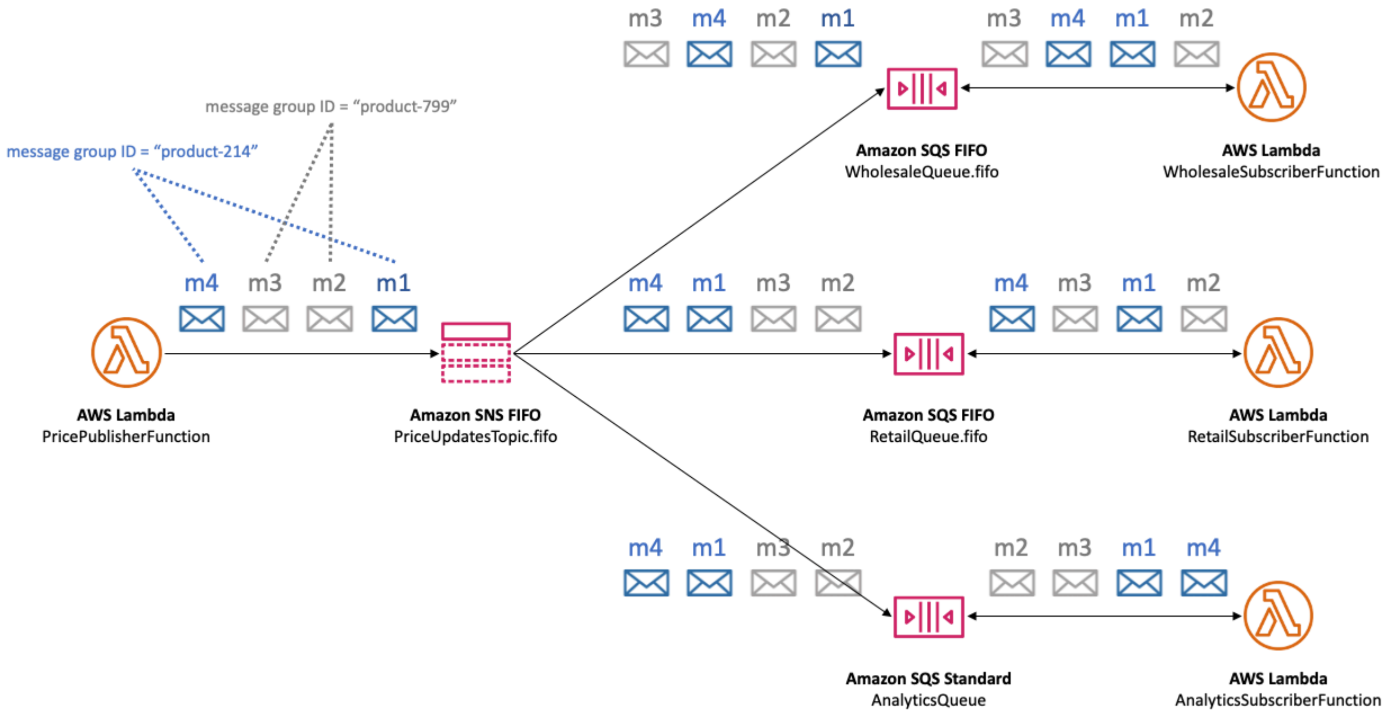
Messages that belong to the same group are processed one by one, in a strict order relative to the group.

When you publish messages to an Amazon SNS FIFO topic, you set the message group ID. The group ID is a mandatory token that specifies that a message belongs to a specific message group. The SNS FIFO topic passes the group ID to the subscribed Amazon SQS FIFO queues. There is no limit to the number of group IDs in SNS FIFO topics or SQS FIFO queues. Message group ID is not passed to Amazon SQS standard queues.

There's no affinity between a message group and a subscription. Therefore, messages that are published to any message group are delivered to all subscribed queues, subject to any filter policies attached to subscriptions. For more information, see Amazon SNS message delivery for FIFO topics and Amazon SNS message filtering for FIFO topics.

In the auto parts price management example use case, there's a dedicated message group for each product sold in the platform. The same Amazon SNS FIFO topic is used for processing all price updates. The sequence of price updates is preserved within the context of a single auto parts product, but not across multiple products. The following diagram shows how this works. Notice that, for the product whose message group ID is **product-214**, message **m1** is processed before

m4. This sequence is preserved throughout the workflows that use Amazon SNS FIFO to Amazon SQS FIFO. Likewise, for the product whose message group ID is **product-799**, message **m2** is processed before **m3**. However, when using Amazon SQS standard queues, the message order is no longer guaranteed, and message groups do not exist. The **product-214** and **product-799** message groups are independent of each other, so there is no relationship between how their messages are sequenced.



Distributing data by message group IDs for improved performance

To optimize delivery throughput, Amazon SNS FIFO topics deliver messages from different message groups in parallel, while message order is strictly maintained within each message group. Each individual message group can deliver a maximum of 300 messages per second. Therefore, to achieve high throughput for a single topic, use a large number of distinct message group IDs. By utilizing a diverse set of message groups, Amazon SNS FIFO topics automatically distributes messages across a larger number of parallel partitions.

Note

Amazon SNS FIFO topics are optimized for uniform distribution of messages across message group IDs, regardless of the number of groups. AWS recommends that you use a large number of distinct message group IDs for optimized performance.

When publishing to your Amazon SNS FIFO topic with high throughput and one or more Amazon SQS FIFO queues are subscribed, it is recommended that you enable high throughput on your queues. For more see High throughput for FIFO queues in the *Amazon Simple Queue Service Developer Guide*.

Amazon SNS message delivery for FIFO topics

Amazon SNS FIFO (first in, first out) topics support delivery to both Amazon SQS standard and FIFO queues to provide customers with flexibility and control when integrating distributed applications that require data consistency in near real-time.

For workloads that need to preserve strict message ordering or de-duplication, the combination of Amazon SNS FIFO topics with Amazon SQS FIFO queues subscribed as the delivery endpoint provides enhance messaging between applications when the order of operations and events is critical, or where duplicates can’t be tolerated.

For workloads that tolerate best-effort ordering and at-least-once delivery, subscribing Amazon SQS standard queues to Amazon SNS FIFO topics provides the ability to lower costs, in addition to sharing queues across workloads that don't utilize FIFO.

Note

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

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

SNS FIFO topics support the same message filtering capabilities as standard topics. For more information, see Amazon SNS message filtering for FIFO topics and the Simplify Your Pub/Sub Messaging with Amazon SNS Message Filtering post on the *AWS Compute Blog*.

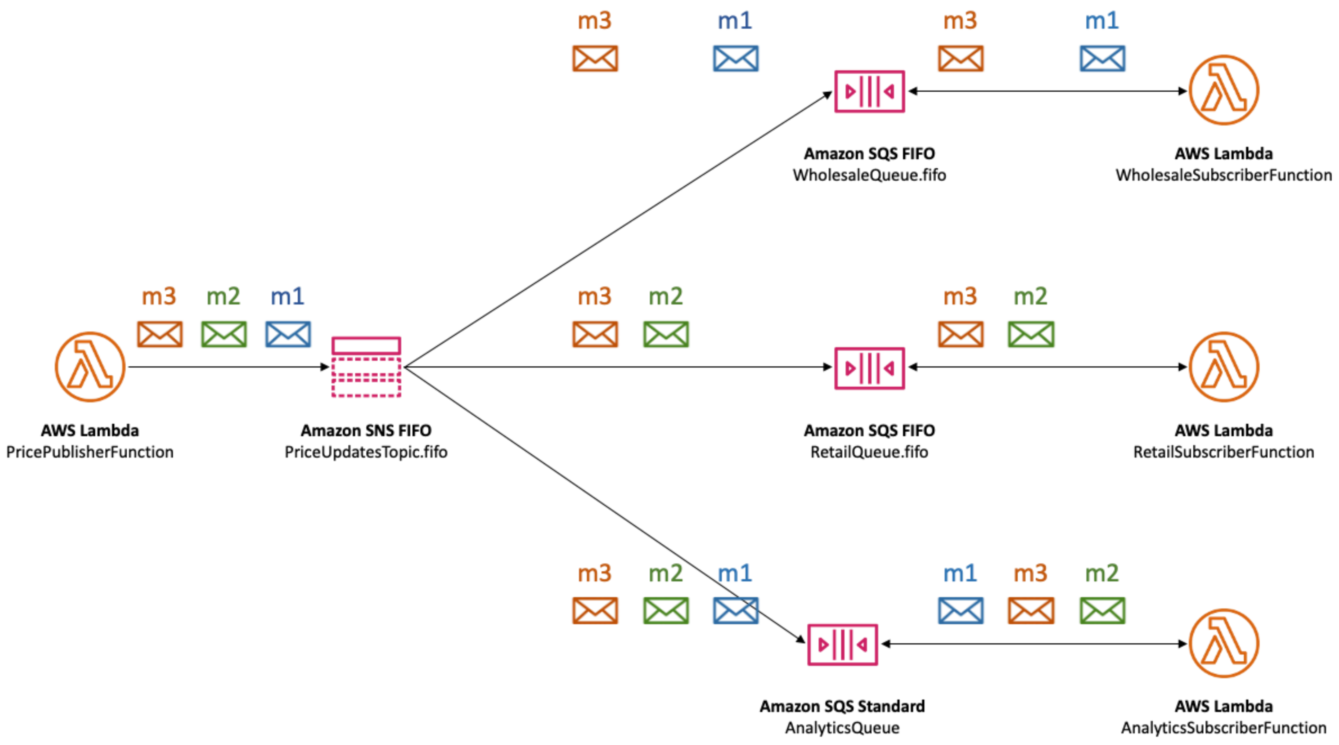
Amazon SNS message filtering for FIFO topics

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

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

In the auto parts price management example use case, assume that the following Amazon SNS filter policies are set and filter policy scope is `MessageBody`:

- For the wholesale queue, the filter policy `{"business":["wholesale"]}` matches every message which contains a key named `business` and with `wholesale` in the set of values. In the following diagram, one of the keys in message **m1** is `business` with the value of `wholesale`. One of the keys in message **m3** is `business` with a value of `["wholesale,retail"]`. Thus, both **m1** and **m3** match the filter policy's criteria, and both messages are delivered to the wholesale queue.
- For the retail queue, the filter policy `{"business":["retail"]}` matches every message which contains a key named `business` and with `retail` in the set of values. In the diagram, one of the keys in message **m2** is `business` with the value of `retail`. One of the keys in message **m3** is `business` with the value of `["wholesale,retail"]`. Thus, both **m2** and **m3** match the filter policy's criteria, and both messages are delivered to the retail queue.
- For the analytics queue, we want Amazon Athena to receive all records, so no filter policy is applied.



SNS FIFO topics support a variety of matching operators, including attribute string values, attribute numeric values, and attribute keys. For more information, see Amazon SNS message filtering.

SNS FIFO topics don't deliver duplicate messages to subscribed endpoints. For more information, see Amazon SNS message deduplication for FIFO topics.

Amazon SNS message deduplication for FIFO topics

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

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

> **Note**
>
> Message deduplication applies to an entire Amazon SNS FIFO topic when the topic attribute `FifoThroughputScope` is set to `Topic`. When the topic attribute `FifoThroughputScope` is set to `MessageGroup`, message deduplication applies to each individual message group.

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

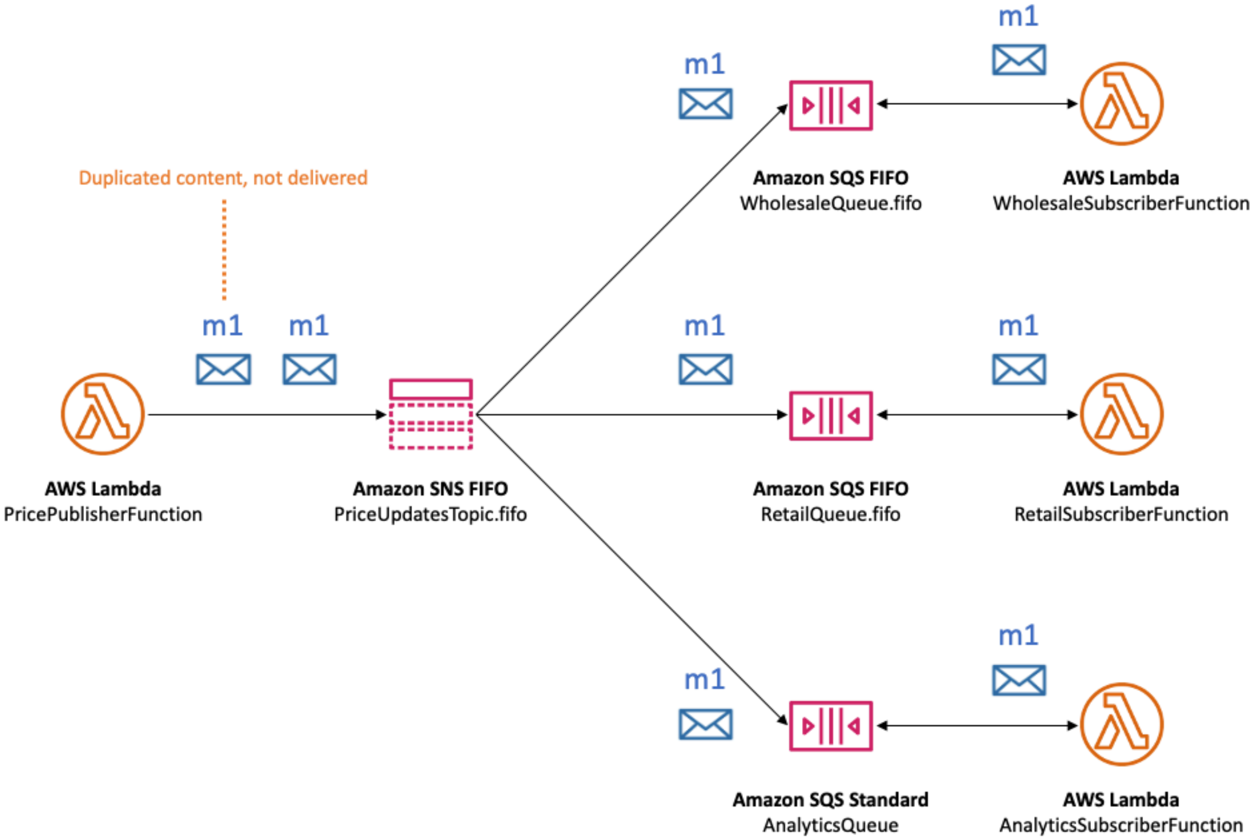
If a message with a particular deduplication ID is successfully published to an Amazon SNS FIFO topic, any message published with the same deduplication ID, within the five minute deduplication interval, is accepted but not delivered. The Amazon SNS FIFO topic continues to track the message deduplication ID, at the deduplication scope configured by the topic attribute `FifoThroughputScope`, even after the message is delivered to subscribed endpoints.

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

> **Note**
>
> Message attributes are not included in the hash calculation.

When content-based deduplication is enabled for an Amazon SNS FIFO topic, and a message is published with a deduplication ID, the published deduplication ID overrides the generated content-based deduplication ID.

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



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

Amazon SNS message security for FIFO topics

You can enable encryption for Amazon SNS FIFO topics and Amazon SQS FIFO queues using AWS Key Management Service (AWS KMS) customer master keys (CMKs).

- You can create new encrypted FIFO topics and queues or enable encryption for existing ones.
- Only the message body is encrypted. Message attributes, resource metadata, and resource metrics remain unencrypted.

Note

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

SNS FIFO topics decrypt the messages immediately before delivering them to subscribed endpoints. SQS FIFO queues decrypt the message just before returning them to the consumer application. For more information, see Amazon SNS data encryption and the Encrypting messages published to Amazon SNS with AWS KMS post on the *AWS Compute Blog*.

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

SNS FIFO topics also support dead-letter queues and message storage across Availability Zones. For more information, see Amazon SNS message durability for FIFO topics.

Amazon SNS message durability for FIFO topics

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

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

- Client-side errors occur when the Amazon SNS FIFO topic has stale subscription metadata. Two common causes of client-side errors are when the Amazon SQS queue owner does one of the following:
  - Deletes the queue.

  - Changes the queue policy in a way that prevents the Amazon SNS service principal from delivering messages to it.

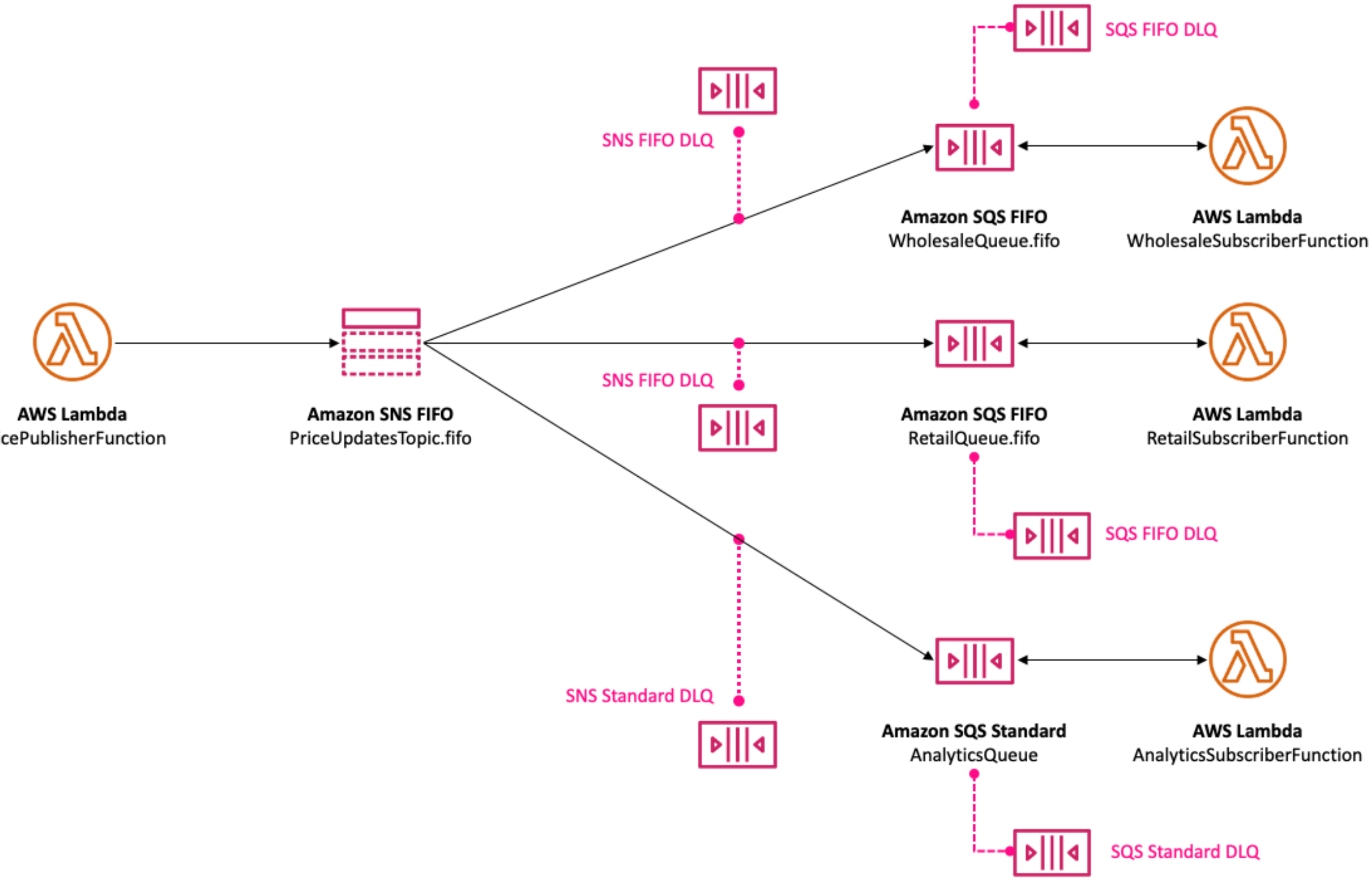
  Amazon SNS doesn't retry delivering messages that failed due to client-side errors.
- Server-side errors can occur in these situations:
  - The Amazon SQS service is unavailable.
  - Amazon SQS fails to process a valid request from the Amazon SNS service.

  When server-side errors occur, Amazon SNS FIFO topics retry the failed deliveries up to 100,015 times over 23 days. For more information, see Amazon SNS message delivery retries.

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

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

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



The dead-letter queue associated with an Amazon SNS subscription must be an Amazon SQS queue of the same type as the subscribing queue. For example, the Amazon SNS FIFO subscription for an Amazon SQS FIFO queue must have an Amazon SQS FIFO queue as the dead-letter queue. Similarly, the Amazon SNS FIFO subscription for an Amazon SQS standard queue must have an Amazon SQS standard queue as its dead-letter queue. For more information, see Amazon SNS dead-letter queues and the Designing durable serverless apps with DLQs for Amazon SNS, Amazon SQS, AWS Lambda post on the *AWS Compute Blog*.

For extended durability to assist in recovery from downstream failures, topic owners can also use FIFO topics to archive messages up to 365 days. Topic subscribers can then replay those messages to a subscribed endpoint to recover messages lost due to a failure in a downstream application, or to replicate a state of an existing application. For more, see Amazon SNS message archiving and replay for FIFO topics.

Amazon SNS message archiving and replay for FIFO topics

What is message archiving and replay?

Amazon SNS provides a no-code message archiving and replay feature, specifically designed for FIFO (First-In-First-Out) topics. This feature allows topic owners to store messages directly within the topic archive for up to 365 days and replay them to subscribers when needed. Message archiving and replay are essential for recovering lost messages and synchronizing applications across regions or systems by replicating states.

This functionality can be accessed through the AWS API, SDK, AWS CloudFormation, and AWS Management Console.

Key use cases

- **Message recovery** – Recover messages lost due to downstream application failures by replaying them to the subscriber's endpoint.
- **State replication –** Replicate the state of an existing system in a new environment by replaying messages starting from a specific timestamp.
- **Error correction –** Resend missed messages during outages to ensure all events are processed correctly.

Components of message archiving and replay

Manage message archiving and replay for Amazon SNS FIFO topics, including setting retention periods, monitoring archived messages using CloudWatch, initiating replays through subscription attributes, and understanding the permissions required to modify and initiate replays.

Message archiving

- The topic owner enables the archiving feature and sets the message retention period, which can be up to 365 days. For more, see Amazon SNS message archiving for FIFO topic owners
- CloudWatch metrics help monitor the archived messages.

Message replay

- A subscriber initiates a replay, selecting the time window for the messages to be reprocessed to the subscribed endpoint. For more see, Amazon SNS message replay for FIFO topic subscribers.

- You manage the replay through subscription attributes using the `ReplayPolicy` feature.

Relevant permissions

- **`SetSubscriptionAttributes`** – Required to configure or modify replay settings using the `ReplayPolicy` attribute on a subscription.
- **`Subscribe`** – Necessary to attach a new subscription and initiate replays.
- **`GetTopicAttributes`** – Allows viewing the topic's properties, but replay initiation primarily revolves around subscription management.

Amazon SNS message archiving for FIFO topic owners

Message archiving provides the ability to archive a single copy of all messages published to your topic. You can store published messages within your topic by enabling the message archive policy on the topic, which enables message archiving for all subscriptions linked to that topic. Messages can be archived for a minimum of one day to a maximum of 365 days.

Additional charges apply when setting an archive policy. For pricing information, see Amazon SNS pricing.

Create a message archive policy using the AWS Management Console

Use this option to create a new message archive policy using the AWS Management Console.

1. Sign in to the Amazon SNS console.
2. Choose a topic or create a new one. To learn more about creating topics, see Creating an Amazon SNS topic.

   > **Note**
   >
   > Amazon SNS message archiving and replay is only available for application-to-application (A2A) FIFO topics.

3. On the **Edit topic** page, expand the **Archive policy** section.
4. Enable the **Archive policy** feature, and enter the **number of days** for which you want to archive messages in the topic.
5. Choose **Save changes**.

To view, edit, and deactivate a message archiving topic policy

- On the **Topic details** page, the **Retention policy** displays the status of the archive policy, including the number of days for which it is set. Select the **Archive policy** tab to view the following message archive details:
  - **Status –** The archive and replay status appears as **active** when an archive policy is applied. The archive and replay status appears as **inactive** when the archive policy is set to an empty JSON object.
  - **Message retention period –** The specified number of days for message retention.
  - **Archive start date –** The date from which subscribers can replay messages.
  - **JSON preview –** The JSON preview of the archive policy.
- (Optional) To **edit** an archive policy, go to the topic summary page and choose **Edit**.
- (Optional) To **deactivate** an archive policy, go to the topic summary page and choose **Edit**. Deactivate the **Archive Policy** and choose **Save changes**.
- (Optional) To **delete** a topic with an archive policy, you must first deactivate the archive policy as previously described.

> **Important**
>
> To avoid accidental message deletions, you can not delete a topic with an active message archive policy. The topic's message archive policy must be deactivated before the topic can be deleted. When you deactivate a message archive policy, Amazon SNS deletes all of the archived messages. When deleting a topic, subscriptions are removed, and any messages in transit may not be delivered.

Create a message archive policy using the API

To create a message archive policy using the API, you need to add the attribute `ArchivePolicy` to your topic. You can set an `ArchivePolicy` using the API actions `CreateTopic` and `SetTopicAttributes`. `ArchivePolicy` has a single value, `MessageRetentionPeriod`, which represents the number of days Amazon SNS retains messages. To activate message archiving for your topic, set the `MessageRetentionPeriod` to an integer value greater than zero. For example, to retain messages in your archive for 30 days, set the `ArchivePolicy` to:

```
{
    "ArchivePolicy": {
        "MessageRetentionPeriod": "30"
    }
}
```

To disable message archiving for your topic, and clear the archive, unset the `ArchivePolicy`, as follows:

```
{}
```

Create a message archive policy using the SDK

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

The following code example shows how to set the `ArchivePolicy` for an Amazon SNS topic to retain all messages published to the topic for 30 days.

```
// Specify the ARN of the Amazon SNS topic to set the ArchivePolicy for.
String topicArn =
    "arn:aws:sns:us-east-2:123456789012:MyArchiveTopic.fifo";

// Set the MessageRetentionPeriod to 30 days for the ArchivePolicy.
String archivePolicy =
    "{\"MessageRetentionPeriod\":\"30\"}";

// Set the ArchivePolicy for the Amazon SNS topic
SetTopicAttributesRequest request = new SetTopicAttributesRequest()
    .withTopicArn(topicArn)
    .withAttributeName("ArchivePolicy")
    .withAttributeValue(archivePolicy);
sns.setTopicAttributes(request);
```

Create a message archive policy using AWS CloudFormation

To create an archive policy using AWS CloudFormation see [`AWS::SNS::Topic`](#) in the *AWS CloudFormation User Guide*.

Grant access to an encrypted archive

Before a subscriber can begin replaying messages from an encrypted topic, you must complete the following steps. Because past messages are replayed, Amazon SNS needs to be provisioned `Decrypt` access to the KMS key that was used to encrypt the messages in the archive.

1. When you encrypt messages with a KMS key and store them within the topic, you must grant Amazon SNS the ability to decrypt these messages via Key Policy. For more, see Grant decrypt permissions to Amazon SNS.
2. Enable AWS KMS for Amazon SNS. For more, see Configuring AWS KMS permissions.

> ⚠ **Important**
>
> When you add the new sections to your KMS key policy, do not change any existing sections in the policy. If encryption is enabled on a topic, and the KMS key is disabled or deleted, or the KMS key policy is not correctly configured for Amazon SNS, Amazon SNS cannot replay messages to your subscribers.

Grant decrypt permissions to Amazon SNS

In order for Amazon SNS to access encrypted messages from within your topic's archive and replay them to subscribed endpoints, you must enable the Amazon SNS service principle to decrypt these messages.

The following is an example policy that is required to allow the Amazon SNS service principal to decrypt stored messages during a replay of historical messages from within your topic.

```
{
    "Sid": "Allow SNS to decrypt archived messages",
    "Effect": "Allow",
    "Principal": {
        "Service": "sns.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

Monitor message archive metrics using Amazon CloudWatch

You can monitor archived messages using Amazon CloudWatch using the following metrics. To be notified of anomalies in your workloads and help avoid impact, you can configure Amazon CloudWatch alarms on these metrics. For more details, see Logging and monitoring in Amazon SNS.

Metric	Description
ApproximateNumberOfMessagesArchived	Provides the topic owner with the aggregate number of messages archived in the topic archive, at 60-minute resolution.
ApproximateNumberOfBytesArchived	Provides the topic owner with the aggregate number of bytes archived, across all messages in the topic archive, at 60-minute resolution.
NumberOfMessagesArchiveProcessing	Provides the topic owner with the number of messages saved to the topic archive during the interval in 1-minute resolution.
NumberOfBytesArchiveProcessing	Provides the topic owner with the aggregate number of bytes saved to the topic archive during the interval in 1-minute resolution.

The `GetTopicAttributes` API has a `BeginningArchiveTime` property, which represents the oldest timestamp at which a subscriber can start a replay. The following represents a sample response for this API action:

```
{
 "ArchivePolicy": {
    "MessageRetentionPeriod": "<integer>"
  },
  "BeginningArchiveTime": "<timestamp>",
  ...
}
```

Amazon SNS message replay for FIFO topic subscribers

Amazon SNS replay allows topic subscribers to retrieve and redeliver archived messages from the topic data store to a subscribed endpoint.

- Messages can be replayed immediately after the subscription is created.
- A replayed message retains the same content, `MessageId`, and `Timestamp` as the original.
- The message includes a `Replayed` attribute to indicate that it is a replayed message.
- To replay only specific messages, apply a filter policy to your subscription.

For more on filtering messages, see Filter replayed messages.

Create a message replay policy using the AWS Management Console

Use this option to create a new replay policy using the AWS Management Console.

1. Sign in to the Amazon SNS console.
2. Choose a topic subscription or create a new one. To learn more about creating subscriptions, see Creating a subscription to an Amazon SNS topic.
3. To initiate the message replay, go to the **Replay** drop-down and choose **Start replay**.
4. From the **Replay timeframe** modal, make the following selections:

   a. **Choose replay start date and time** – Choose the **date** (YYYY/MM/DD format) and **time** (24-hour hh:mm:ss format) from which you want to start replaying archived messages. The start time should be later than the beginning of the approximated archive time.

   b. **(Optional) Choose replay end date and time** – Choose the **date** (YYYY/MM/DD format) and **time** (24-hour hh:mm:ss format) when you want to stop replaying archived messages.

   c. Choose **Start replay**.
5. (Optional) To **stop** a message replay, go to the **Subscription details** page and choose **Stop replay** from the **Replay** drop-down.
6. (Optional) To **monitor** message replay metrics from within this workflow using CloudWatch, see Monitor message replay metrics using Amazon CloudWatch.

To view and edit a message replay policy

You can perform the following actions from the **Subscription details** page:

- To **view** the message replay status, the **Replay status** field displays the following values:
  - **Completed –** The replay has successfully redelivered all messages, and is now delivering newly published messages.
  - **In progress –** The replay is currently replaying the selected messages.
  - **Failed –** The replay was unable to complete.
  - **Pending –** The default state while the replay initiates.
- (Optional) To **modify** a message replay policy, go to the **Subscription details** page and choose **Start replay** from the **Replay** drop-down. Starting a replay will replace the existing replay.

Add a replay policy to the subscription using the API

To replay archived messages use the attribute `ReplayPolicy`. `ReplayPolicy` can be used with the `Subscribe` and `SetSubscriptionAttributes` API actions. This policy has the following values:

- **`StartingPoint`** (Required) – Signals where to start replaying messages from.
- **`EndingPoint`** (Optional) – Signals when to stop replaying messages. If `EndingPoint` is omitted, then the replay will continue until caught up to the current time.
- **`PointType`** (Required) – Sets the type of starting and ending points. Currently, the supported value for `PointType` is `Timestamp`.

For example, to recover from a downstream failure and resend all messages for a two hour time period on October 1, 2023, use the `SetSubscriptionAttributes` API action to set a `ReplayPolicy` as follows:

```
{
  "PointType":"Timestamp",
  "StartingPoint":"2023-10-01T10:00:00.000Z",
  "EndingPoint":"2023-10-01T12:00:00.000Z"
}
```

To replay all messages sent to the topic as of October 1, 2023, and continue receiving all newly published messages to your topic, use the `SetSubscriptionAttributes` API action to set a `ReplayPolicy` on your subscription as follows:

```
{
  "PointType":"Timestamp",
  "StartingPoint":"2023-10-01T00:00:00.000Z"
}
```

To verify that a message has been replayed, the boolean attribute `Replayed` is added to each replayed message.

Add a replay policy to the subscription using the SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see Shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

The following code example shows how to set the `ReplayPolicy` on a subscription to redeliver messages from the Amazon SNS FIFO topic's archive for a 2-hour time window on October 1st 2023.

```
// Specify the ARN of the Amazon SNS subscription to initiate the ReplayPolicy on.
String subscriptionArn =
    "arn:aws:sns:us-
east-2:123456789012:MyArchiveTopic.fifo:1d2a3e9d-7f2f-447c-88ae-03f1c68294da";

// Set the ReplayPolicy to replay messages from the topic's archive
// for a 2 hour time period on October 1st 2023 between 10am and 12pm UTC.
String replayPolicy =
    "{\"PointType\":\"Timestamp\",\"StartingPoint\":\"2023-10-01T10:00:00.000Z\",
\"EndingPoint\":\"2023-10-01T12:00:00.000Z\"}";

// Set the ArchivePolicy for the Amazon SNS topic
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("ReplayPolicy")
    .withAttributeValue(replayPolicy);
sns.setSubscriptionAttributes(request);
```

Understanding the EndingPoint

When you apply a `ReplayPolicy` to an Amazon SNS subscription, the `EndingPoint` value is optional. If no `EndingPoint` is provided, the replay will start from the specified `StartingPoint` and continue until it catches up to the current time, including processing any newly published messages. Once caught up, the subscription will function as a regular subscription, receiving new messages as they are published.

If an `EndingPoint` is specified, the service will replay messages from the `StartingPoint` up to the `EndingPoint` and then stop. **This action effectively pauses the subscription.** While the subscription is paused, newly published messages will not be delivered to the subscribed endpoint.

To resume message delivery, apply a new `ReplayPolicy` without providing an `EndingPoint`, and set the `StartingPoint` to the desired point in time from which to continue receiving messages. For example, to resume a subscription from where a prior replay finished, set the new `StartingPoint` to the previously provided `EndingPoint`.

Filter replayed messages

Amazon SNS message filtering let's you control the replayed messages that Amazon SNS replays to your subscriber endpoint. When message filtering and message archiving are both enabled, Amazon SNS first retrieves the message from the topic’s data store, then applies the message against the subscription’s `FilterPolicy`. The message is delivered to the subscribed endpoint when there is a match, otherwise message is filtered out. For more information, see [Amazon SNS subscription filter policies](#).

Monitor message replay metrics using Amazon CloudWatch

You can monitor replay messages using Amazon CloudWatch using the following metrics. To be notified of anomalies in your workloads and help avoid impact, you can configure Amazon CloudWatch alarms on these metrics. For more details, see [Logging and monitoring in Amazon SNS](#).

Metric	Description
NumberOfReplayedNotificationsDelivered	Provides the subscriber with the aggregate number of messages replayed from the topic archive, at 1-minute resolution.
NumberOfReplayedNotificationsFailed	Provides the subscriber with the aggregate number of messages replayed that failed to deliver from the topic archive, at 1-minute resolution.

Amazon SNS code examples for FIFO topics

Use the following code examples to integrate the auto parts price management example use case with an Amazon SNS FIFO topic and either an Amazon SQS FIFO queue or a standard queue.

Using an AWS SDK

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

The following code examples create these FIFO and standard queue resources:

- The Amazon SNS FIFO topic that distributes the price updates
- The Amazon SQS FIFO queues that provide these updates to the wholesale and retail applications
- The Amazon SQS standard queue for the analytics application that stores records, which can be queried for business intelligence (BI)
- The Amazon SNS FIFO subscriptions that connect the three queues to the topic

This example sets filter policies in the subscriptions. If you test the example by publishing a message to the topic, make sure that you publish the message with the `business` attribute. Specify either `retail` or `wholesale` for the attribute value. Otherwise, the message is filtered out and not delivered to the subscribed queues. For more information, see Amazon SNS message filtering for FIFO topics.

Java

SDK for Java 2.x

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

This example

- creates an Amazon SNS FIFO topic, two Amazon SQS FIFO queues, and one Standard queue.
- subscribes the queues to the topic and publishes a message to the topic.

The test verifies the receipt of the message to each queue. The complete example also shows the addition of access policies and deletes the resources at the end.

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
 <analyticsQueueName>\n\n" +
            "Where:\n" +
            "   fifoTopicName - The name of the FIFO topic that you want to
 create. \n\n" +
            "   wholesaleQueueARN - The name of a SQS FIFO queue that will be
 created for the wholesale consumer. \n\n"
            +
            "   retailQueueARN - The name of a SQS FIFO queue that will created
 for the retail consumer. \n\n" +
            "   analyticsQueueARN - The name of a SQS standard queue that will be
 created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue:
 ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
        String topicARN = createFIFOTopic(fifoTopicName);

        // Subscribe each queue to the topic.
        subscribeQueues(queues, topicARN);

        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

        // Publish a sample price update message with payload.
        publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
 "Consumables");

        // Clean up resources.
        deleteSubscriptions(queues);
        deleteQueues(queues);
        deleteTopic(topicARN);
    }

    public static String createFIFOTopic(String topicName) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = Map.of(
                "FifoTopic", "true",
                "ContentBasedDeduplication", "false",
                "FifoThroughputScope", "MessageGroup");

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

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

            return topicArn;

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

    public static void subscribeQueues(List<QueueData> queues, String topicARN) {
        queues.forEach(queue -> {
            SubscribeRequest subscribeRequest = SubscribeRequest.builder()
                .topicArn(topicARN)
                .endpoint(queue.queueARN)
                .protocol("sqs")
                .build();

            // Subscribe to the endpoint by using the SNS service client.
            // Only Amazon SQS queues can receive notifications from an Amazon
 SNS FIFO
            // topic.
            SubscribeResponse subscribeResponse =
 snsClient.subscribe(subscribeRequest);
            System.out.println("The queue [" + queue.queueARN + "] subscribed to
 the topic [" + topicARN + "]");
            queue.subscriptionARN = subscribeResponse.subscriptionArn();
        });
    }

    public static void publishPriceUpdate(String topicArn, String payload, String
 groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

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

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

```
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
            System.out.println(response.sequenceNumber());
            System.out.println("Message was published to " + topicArn);

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

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

Python

SDK for Python (Boto3)

Note

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

Create an Amazon SNS FIFO topic, subscribe Amazon SQS FIFO and standard queues to the topic, and publish a message to the topic.

```
def usage_demo():
    """Shows how to subscribe queues to a FIFO topic."""
    print("-" * 88)
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")
    print("-" * 88)

    sns = boto3.resource("sns")
    sqs = boto3.resource("sqs")
    fifo_topic_wrapper = FifoTopicWrapper(sns)
    sns_wrapper = SnsWrapper(sns)

    prefix = "sqs-subscribe-demo-"
    queues = set()
    subscriptions = set()

    wholesale_queue = sqs.create_queue(
        QueueName=prefix + "wholesale.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(wholesale_queue)
    print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

    retail_queue = sqs.create_queue(
        QueueName=prefix + "retail.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(retail_queue)
    print(f"Created FIFO queue with URL: {retail_queue.url}.")

    analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
 Attributes={})
    queues.add(analytics_queue)
    print(f"Created standard queue with URL: {analytics_queue.url}.")
    topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
    print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])

    print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        sub = fifo_topic_wrapper.subscribe_queue_to_topic(
            topic, q.attributes["QueueArn"]
        )
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")

    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)



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

    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.
        Topic names must be made up of only uppercase and lowercase ASCII
 letters,
        numbers, underscores, and hyphens, and must be between 1 and 256
 characters long.
        For a FIFO topic, the name must end with the .fifo suffix.

        :param topic_name: The name for the topic.
        :return: The new topic.
        """
        try:
            topic = self.sns_resource.create_topic(
                Name=topic_name,
                Attributes={
                    "FifoTopic": str(True),
                    "ContentBasedDeduplication": str(False),
                    "FifoThroughputScope": "MessageGroup",
                },
            )
            logger.info("Created FIFO topic with name=%s.", topic_name)
            return topic
        except ClientError as error:
            logger.exception("Couldn't create topic with name=%s!", topic_name)
            raise error


    @staticmethod
    def add_access_policy(queue, topic_arn):
        """
        Add the necessary access policy to a queue, so
        it can receive messages from a topic.

        :param queue: The queue resource.
        :param topic_arn: The ARN of the topic.
        :return: None.
        """
        try:
            queue.set_attributes(
                Attributes={
                    "Policy": json.dumps(
                        {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Sid": "test-sid",
                                    "Effect": "Allow",
                                    "Principal": {"AWS": "*"},
                                    "Action": "SQS:SendMessage",
                                    "Resource": queue.attributes["QueueArn"],
                                    "Condition": {
                                        "ArnLike": {"aws:SourceArn": topic_arn}
                                    },
                                }
                            ],
                        }
                    )
                }
            )
            logger.info("Added trust policy to the queue.")
        except ClientError as error:
            logger.exception("Couldn't add trust policy to the queue!")
            raise error


    @staticmethod
    def subscribe_queue_to_topic(topic, queue_arn):
        """
        Subscribe a queue to a topic.

        :param topic: The topic resource.
        :param queue_arn: The ARN of the queue.
        :return: The subscription resource.
        """
        try:
            subscription = topic.subscribe(
                Protocol="sqs",
                Endpoint=queue_arn,
            )
            logger.info("The queue is subscribed to the topic.")
            return subscription
        except ClientError as error:
            logger.exception("Couldn't subscribe queue to topic!")
            raise error


    @staticmethod
    def publish_price_update(topic, payload, group_id):
        """
        Compose and publish a message that updates the wholesale price.

        :param topic: The topic to publish to.
        :param payload: The message to publish.
        :param group_id: The group ID for the message.
        :return: The ID of the message.
        """
        try:
            att_dict = {"business": {"DataType": "String", "StringValue":
"wholesale"}}
            dedup_id = uuid.uuid4()
            response = topic.publish(
                Subject="Price Update",
                Message=payload,
                MessageAttributes=att_dict,
                MessageGroupId=group_id,
                MessageDeduplicationId=str(dedup_id),
            )
            message_id = response["MessageId"]
            logger.info("Published message to topic %s.", topic.arn)
        except ClientError as error:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise error
        return message_id


    @staticmethod
    def delete_queue(queue):
        """
        Removes an SQS queue. When run against an AWS account, it can take up to
        60 seconds before the queue is actually deleted.

        :param queue: The queue to delete.
        :return: None
        """
        try:
            queue.delete()
            logger.info("Deleted queue with URL=%s.", queue.url)
        except ClientError as error:
            logger.exception("Couldn't delete queue with URL=%s!", queue.url)
            raise error
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

SAP ABAP

SDK for SAP ABAP

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

Create a FIFO topic, subscribe an Amazon SQS FIFO queue to the topic, and publish a message to an Amazon SNS topic.

```
    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_w( iv_value =
 'true' ).
    INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

    TRY.
```

```
        DATA(lo_create_result) = lo_sns->createtopic(
               iv_name = iv_topic_name
               it_attributes = lt_tpc_attributes ).
        DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
        ov_topic_arn = lv_topic_arn.                                    "
 ov_topic_arn is returned for testing purposes. "
        MESSAGE 'FIFO topic created' TYPE 'I'.
      CATCH /aws1/cx_snstopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
 SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
 to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
               iv_topicarn = lv_topic_arn
               iv_protocol = 'sqs'
               iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.                      "
 ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
 'String'

 iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.
        DATA(lo_result) = lo_sns->publish(
             iv_topicarn = lv_topic_arn
             iv_message = 'The price of your mobile plan has been increased from
$19 to $23'
             iv_subject = 'Changes to mobile plan'
             iv_messagegroupid = 'Update-2'
             iv_messagededuplicationid = 'Update-2.1'
             it_messageattributes = lt_msg_attributes ).
        ov_message_id = lo_result->get_messageid( ).                    "
ov_message_id is returned for testing purposes. "
        MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see the following topics in *AWS SDK for SAP ABAP API reference*.
  - CreateTopic
  - Publish
  - Subscribe

Receiving messages from FIFO subscriptions

You can now receive price updates in the three subscribed applications. As shown in the the section called "FIFO topic use case", the point of entry for each consumer application is the Amazon SQS queue, which its corresponding AWS Lambda function can poll automatically. When an Amazon SQS queue is an event source for a Lambda function, Lambda scales its fleet of pollers as needed to efficiently consume messages.

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

Using AWS CloudFormation

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

- The Amazon SNS FIFO topic that distributes the price updates
- The Amazon SQS FIFO queues that provide these updates to the wholesale and retail applications
- The Amazon SQS standard queue for the analytics application that stores records, which can be queried for business intelligence (BI)
- The Amazon SNS FIFO subscriptions that connect the three queues to the topic
- A filter policy that specifies that subscriber applications receive only the price updates that they need

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

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "PriceUpdatesTopic": {
      "Type": "AWS::SNS::Topic",
      "Properties": {
        "TopicName": "PriceUpdatesTopic.fifo",
        "FifoTopic": true,
        "ContentBasedDeduplication": false,
        "ArchivePolicy": {
        "MessageRetentionPeriod": "30"
        }
      }
    },
    "WholesaleQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "QueueName": "WholesaleQueue.fifo",
        "FifoQueue": true,
        "ContentBasedDeduplication": false
      }
    },
    "RetailQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "QueueName": "RetailQueue.fifo",
        "FifoQueue": true,
        "ContentBasedDeduplication": false
      }
    },
    "AnalyticsQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "QueueName": "AnalyticsQueue"
      }
    },
    "WholesaleSubscription": {
      "Type": "AWS::SNS::Subscription",
      "Properties": {
        "TopicArn": {
          "Ref": "PriceUpdatesTopic"
        },
        "Endpoint": {
          "Fn::GetAtt": [
            "WholesaleQueue",
            "Arn"
          ]
        },
        "Protocol": "sqs",
        "RawMessageDelivery": "false",
        "FilterPolicyScope": "MessageBody",
        "FilterPolicy": {
          "business": [
            "wholesale"
          ]
        }
      }
    },
    "RetailSubscription": {
      "Type": "AWS::SNS::Subscription",
      "Properties": {
        "TopicArn": {
          "Ref": "PriceUpdatesTopic"
        },
        "Endpoint": {
          "Fn::GetAtt": [
            "RetailQueue",
            "Arn"
          ]
        },
        "Protocol": "sqs",
        "RawMessageDelivery": "false",
        "FilterPolicyScope": "MessageBody",
        "FilterPolicy": {
          "business": [
            "retail"
          ]
        }
      }
    },
    "AnalyticsSubscription": {
      "Type": "AWS::SNS::Subscription",
      "Properties": {
        "TopicArn": {
          "Ref": "PriceUpdatesTopic"
        },
        "Endpoint": {
          "Fn::GetAtt": [
            "AnalyticsQueue",
            "Arn"
          ]
        },
        "Protocol": "sqs",
        "RawMessageDelivery": "false"
      }
    },
    "SalesQueuesPolicy": {
      "Type": "AWS::SQS::QueuePolicy",
      "Properties": {
        "PolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "sns.amazonaws.com"
              },
              "Action": [
                "sqs:SendMessage"
              ],
              "Resource": "*",
              "Condition": {
                "ArnEquals": {
                  "aws:SourceArn": {
                    "Ref": "PriceUpdatesTopic"
                  }
                }
              }
            }
          ]
        },
        "Queues": [
          {
            "Ref": "WholesaleQueue"
          },
          {
            "Ref": "RetailQueue"
          },
          {
            "Ref": "AnalyticsQueue"
          }
        ]
      }
    }
  }
}
```

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

Amazon SNS message filtering

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

A filter policy is a JSON object containing properties that define which messages the subscriber receives. Amazon SNS supports policies that act on the message attributes or on the message body, according to the filter policy scope that you set for the subscription. Filter policies for the message body assume that the message payload is a well-formed JSON object.

If a subscription doesn't have a filter policy, the subscriber receives every message published to its topic. When you publish a message to a topic with a filter policy in place, Amazon SNS compares the message attributes or the message body to the properties in the filter policy for each of the topic's subscriptions. If all of the message attributes or message body properties satisfy the conditions specified in the filter policy, Amazon SNS sends the message to the subscriber. Otherwise, Amazon SNS doesn't send the message to that subscriber.

For more information, see Filter Messages Published to Topics.

Amazon SNS subscription filter policy scope

The `FilterPolicyScope` subscription attribute allows you define the filtering scope by setting one of the following values:

- `MessageAttributes` – Applies the filter policy to message attributes (default setting).
- `MessageBody` – Applies the filter policy to the message body.

> **Note**
>
> If no filter policy scope is defined for an existing filter policy, the scope defaults to `MessageAttributes`.

Amazon SNS subscription filter policies

A subscription filter policy allows you to specify property names and assign a list of values to each property name. For more information, see Amazon SNS message filtering.

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

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

A subscription accepts a message under the following conditions:

- When the filter policy scope is set to `MessageAttributes`, each property name in the filter policy matches a message attribute name. For each matching property name in the filter policy, at least one property value matches the message attribute value.
- When the filter policy scope is set to `MessageBody`, each property name in the filter policy matches a message body property name. For each matching property name in the filter policy, at least one property value matches the message body property value.

Amazon SNS currently supports the following filter operators:

- AND logic
- OR logic
- OR operator
- Key matching
- Numeric value exact matching
- Numeric value anything-but matching
- Numeric value range matching
- String value exact matching
- String value anything-but matching

- String matching using a prefix with the anything-but operator
- String value equals-ignore case
- String value IP address matching
- String value prefix matching
- String value suffix matching

Amazon SNS example filter policies

The following example shows a message payload delivered by an Amazon SNS topic that processes customer transactions.

The first example includes the `MessageAttributes` field with attributes that describe the transaction:

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

Because this message includes the `MessageAttributes` field, any topic subscription that sets a `FilterPolicy` can selectively accept or reject the message, as long as `FilterPolicyScope` is set to `MessageAttributes` in the subscription. For information about applying attributes to a message, see Amazon SNS message attributes.

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

The following example shows the same attributes included within the `Message` field, also referred to as the *message payload* or *message body*. Any topic subscription that includes a `FilterPolicy` can selectively accept or reject the message, as long as `FilterPolicyScope` is set to `MessageBody` in the subscription.

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

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

A policy that accepts the example message

The properties in the following subscription filter policy match the attributes assigned to the example message. Note that the same filter policy works for a `FilterPolicyScope` whether it's set to `MessageAttributes` or `MessageBody`. Each subscriber chooses their filtering scope according to the composition of the messages that they receive from the topic.

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

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

A policy that rejects the example message

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

If any mismatches occur, the policy rejects the message.

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

Filter policy constraints in Amazon SNS

When you’re setting up filter policies in Amazon SNS, there are a few important rules to keep in mind. These rules help ensure the effective application of filter policies while maintaining system performance and compatibility.

Common policy constraints

When configuring filter policies in Amazon SNS, follow these important rules to ensure they work effectively while maintaining system performance and compatibility:

- **String matching –** For string matching in the filter policy, the comparison is case-sensitive.
- **Numeric matching –** For numeric matching, the value can range from -10^9 to 10^9 (-1 billion to 1 billion), with five digits of accuracy after the decimal point.
- **Filter policy complexity –** The total combination of values in a filter policy must not exceed **150**. To calculate the total combination, multiply the number of values in each array in the filter policy.
- **Limit number of keys –** A filter policy can have a maximum of **five** keys.

Additional considerations

- The JSON of the filter policy can contain the following:
  - Strings enclosed in quotation marks
  - Numbers
  - The keywords `true`, `false`, and `null`, without quotation marks

- When using the Amazon SNS API, you must pass the JSON of the filter policy as a valid **UTF-8** string.

- The maximum size of a filter policy is **256 KB**.
- By default, you can have up to **200** filter policies per topic, and **10,000** filter policies per AWS account.

  This policy limit won't stop Amazon SQS queue subscriptions from being created with the `Subscribe` API. However, it will fail when you attach the filter policy in the `Subscribe` API call (or the `SetSubscriptionAttributes` API call).

  To increase this quota, you can use AWS Service Quotas.

Policy constraints for attribute-based filtering

Attribute-based filtering is the default option. `FilterPolicyScope` is set to `MessageAttributes` in the subscription.

- Amazon SNS doesn't accept a nested filter policy for attribute-based filtering.
- Amazon SNS compares policy properties only to message attributes that have the following data types:
  - `String`
  - `String.Array`

> **Important**
>
> When using attribute-based filtering in Amazon SNS, you must double-escape certain special characters, specifically:
>
> - Double quotes (")
> - Backslashes ()
>
> Failure to double-escape these characters will result in the filter policy not matching the attributes of a published message, and the notification won't be delivered.

Additional considerations

- Passing objects in arrays isn't recommended because it may yield unexpected results due to the nesting, which isn't supported by attribute-based filtering. Use payload-based filtering for nested policies.
- `Number` is supported for numeric attribute values.
- Amazon SNS ignores message attributes with the Binary data type.

Example policy for complexity:

In the following policy example, the first key has **three** match operators, the second has **one** match operator, and the third has **two** match operators.

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

The total combination is calculated as the product of the number of match operators for each key in the filter policy:

```
3(match operators of key_a)
x 1(match operators of key_b)
x 2(match operators of key_c)
= 6
```

Policy constraints for payload-based filtering

To switch from attribute-based (default) to payload-based filtering, you must set the `FilterPolicyScope` to `MessageBody` in the subscription.

- Amazon SNS accepts a nested filter policy for payload-based filtering.
- For a nested policy, only **leaf keys** are counted towards the **five** key limit.

Example policy for key limit:

In the following policy example:

- There are two leaf keys: `key_c` and `key_e`.
- `key_c` has **four** match operators with a nested level of **three**, and `key_e` has **three** match operators with a nested level of **two**.

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

The total combination is calculated as the product of the number of match operators and the nested level for each key in the filter policy:

```
4(match operators of key_c)
x 3(nested level of key_c)
x 3(match operators of key_e)
x 2(nested level of key_e)
= 72
```

Wildcard pattern usage guidelines

Amazon SQS implements protections for when you register a filter policy containing wildcards to ensure that filter policies too complex are not created, as this would impact your application performance.

Pattern structure

Fields contain one or more patterns. The following example shows a field using two patterns:

```
{
    "greeting": [
      {"anything-but": {"prefix": "Hello"}},
      {"wildcard": "H*"}
    ] // 2 patterns
  }
```

Complexity rules

- Total wildcard complexity across all fields must not exceed 100 points
- Maximum 3 wildcards per pattern

Complexity calculation

- Field complexity = `(Sum of pattern points)` × `(Number of patterns)`
- Pattern points:

  Single wildcard: 1 point

  Multiple wildcards: 3 points each

  Anything-but: 1 point

The following is an example of the complexity calculation:

```
{
  "filename": [
    {"wildcard": "*.txt"},     // 1 point
    {"wildcard": "log*"}      // 1 point
  ]                           // Total: (1 + 1) × 2 = 4 points
}
```

AND/OR logic

Use AND/OR logic in filter policies to match message attributes or message body properties in Amazon SNS. This enables more precise and flexible message filtering.

AND logic

You can apply AND logic using multiple property names.

Consider the following policy:

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

It matches any message attribute or message body property with the value of `customer_interests` set to `rugby` *and* the value of `price_usd` set to a number larger than 100.

Note

You can't apply AND logic to values of the same attribute.

OR logic

You can apply OR logic by assigning multiple values to a property name.

Consider the following policy:

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

It matches any message attribute or message body property with the value of `customer_interests` set to `rugby`, `football`, *or* `baseball`.

OR operator

You can use the `"$or"` operator to explicitly define a filter policy to express the OR relationship between multiple attributes in the policy.

Amazon SNS only recognizes an `"$or"` relationship when the policy has met all of the following conditions. When all of these conditions are not met, `"$or"` is treated as a regular attribute name, the same as any other string in the policy.

- There is an `"$or"` field attribute in the rule followed with an array, for example `“$or” : []`.
- There are at least 2 objects in the `"$or"` array: `"$or": [{}, {}]`.
- None of the objects in the `"$or"` array have field names that are reserved keywords.

Otherwise `"$or"` is treated as a normal attribute name, the same as other strings in the policy.

The following policy isn't parsed as an OR relationship because numeric and prefix are reserved keywords.

```
{
   "$or": [ {"numeric" : 123}, {"prefix": "abc"} ]
}
```

OR operator examples

Standard OR:

```
{
  "source": [ "aws.cloudwatch" ],
  "$or": [
    { "metricName": [ "CPUUtilization" ] },
    { "namespace": [ "AWS/EC2" ] }
  ]
}
```

The filter logic for this policy is:

```
"source" && ("metricName" || "namespace")
```

It matches either of the following sets of message attributes:

```
"source": {"Type": "String", "Value": "aws.cloudwatch"},
"metricName": {"Type": "String", "Value": "CPUUtilization"}
```

or

```
"source": {"Type": "String", "Value": "aws.cloudwatch"},
"namespace": {"Type": "String", "Value": "AWS/EC2"}
```

It also matches either of the following message bodies:

```
{
    "source": "aws.cloudwatch",
    "metricName": "CPUUtilization"
}
```

or

```
{
    "source": "aws.cloudwatch",
    "namespace": "AWS/EC2"
}
```

Policy constraints that include OR relationships

Consider the following policy:

```
{
    "source": [ "aws.cloudwatch" ],
    "$or": [
      { "metricName": [ "CPUUtilization", "ReadLatency" ] },
      {
        "metricType": [ "MetricType" ] ,
        "$or" : [
          { "metricId": [ 1234, 4321 ] },
          { "spaceId": [ 1000, 2000, 3000 ] }
        ]
      }
    ]
  }
```

The logic for this policy can also be simplified as:

```
("source" AND "metricName")
OR
("source" AND "metricType" AND "metricId")
OR
("source" AND "metricType" AND "spaceId")
```

The complexity calculation for policies with OR relationships can be simplified as the sum of the combination complexities for each OR statement.

The total combination is calculated as follows:

```
(source * metricName) + (source * metricType * metricId) + (source * metricType *
 spaceId)
= (1 * 2) + (1 * 1 * 2) + (1 * 1 * 3)
= 7
```

`source` has one value, `metricName` has two values, `metricType` has one value, `metricId` has two values and `spaceId` has three values.

Consider the following nested filter policy:

```
{
    "$or": [
      { "metricName": [ "CPUUtilization", "ReadLatency" ] },
      { "namespace": [ "AWS/EC2", "AWS/ES" ] }
    ],
    "detail" : {
      "scope" : [ "Service" ],
      "$or": [
        { "source": [ "aws.cloudwatch" ] },
        { "type": [ "CloudWatch Alarm State Change"] }
      ]
    }
  }
```

The logic for this policy can be simplified as:

```
("metricName" AND ("detail"."scope" AND "detail"."source")
OR
("metricName" AND ("detail"."scope" AND "detail"."type")
OR
("namespace" AND ("detail"."scope" AND "detail"."source")
OR
("namespace" AND ("detail"."scope" AND "detail"."type")
```

The calculation for total combinations is the same for non-nested policies except we need to consider the a key's nesting level.

The total combination is calculated as follows:

```
(2 * 2 * 2) + (2 * 2 * 2) + (2 * 2 * 2) + (2 * 2 * 2) = 32
```

`metricName` has two values, `namespace` has two values, `scope` is a two level nested key with one value, `source` is a two level nested key with one value, and `type` is a two level nested key with one value.

Key matching

Use the `exists` operator in a filter policy to match incoming messages based on whether a specific property is present or absent.

- `exists` works only on leaf nodes (final attributes in the structure).
- It does not apply to intermediate nodes within a nested JSON structure.

- Use `"exists": true` to match incoming messages that include the specified property. The key must have a non-null and non-empty value.

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

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

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

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

It also matches either of the following message body:

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

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

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

Nor does it match the following message body:

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

- Use `"exists": false` to match incoming messages that *don't* include the specified property.

> **Note**
>
> `"exists": false` only matches if at least one attribute is present. An empty set of attributes results in the filter not matching.

For example, the following policy property uses the `exists` operator with a value of `false`:

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

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

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

It also doesn’t match the following message body:

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

However, it matches any list of message attributes *without* the `store` attribute key, such as the following:

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

It also matches the following message body:

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

Numeric value matching

Filter messages by matching numeric values to message attribute values or to message body property values. Numeric values aren't enclosed in double quotation marks in the JSON policy. You can use the following numeric operations for filtering.

> **Note**
>
> Prefixes are supported for *string* matching only.

Exact matching

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

Consider the following policy property:

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

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
   "price_usd": 301.5
}
```

```
{
   "price_usd": 3.015e2
}
```

Anything-but matching

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

Consider the following policy property:

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

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
   "price": 101
}
```

```
{
   "price": 100.1
}
```

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

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

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

```
{
   "price": [100, 50]
}
```

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

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

Nor does it match the following message body:

```
{
   "price": 100
}
```

Value range matching

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

Consider the following policy property:

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

It matches any message attribute or message body property with negative numeric values.

Consider another message attribute:

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

It matches any message attribute or message body property with positive numbers up to and including 150.

String value matching

Filter messages by matching string values to message attribute values or message body property values. String values are enclosed in double quotation marks in the JSON policy. You can use the following string operations to match message attributes or message body properties:

Exact matching

Exact matching occurs when a policy property value matches one or more message attribute values. For `String.Array` type attributes, each element in the array is treated as a separate string for matching purposes.

Consider the following policy property:

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

It matches the following message attributes:

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

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

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

It also matches the following message bodies:

```
{
   "customer_interests": "rugby"
}
```

```
{
   "customer_interests": "tennis"
}
```

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

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

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

Nor does it match the following message body:

```
{
   "customer_interests": "baseball"
}
```

Anything-but matching

When a policy property value includes the keyword `anything-but`, it matches any message attribute or message body values that *don't* include any of the policy property values. `anything-but` can be combined with `"exists": false`. For `String.Array` type attributes, it matches if none of the array elements are listed in the policy property.

Consider the following policy property:

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

It matches any of the following message attributes:

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

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

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

It also matches either of the following message bodies:

```
{
   "customer_interests": "baseball"
}
```

```
{
   "customer_interests": "football"
}
```

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

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

And it also matches the following message body (because it contains a value that isn't `rugby` or `tennis`):

```
{
   "customer_interests": ["rugby", "baseball"]
}
```

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

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

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

Nor does it match the following message body:

```
{
   "customer_interests": ["rugby"]
}
```

Using a prefix with `anything-but`

For string matching, you can also use a prefix with the `anything-but` operator. For example, the following policy property denies the `order-` prefix:

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

It matches either of the following attributes:

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

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

It also matches either of the following message bodies:

```
{
   "event": "data-entry"
}
```

```
{
   "event": "order_number"
}
```

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

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

Nor does it match the following message body:

```
{
   "event": "order-cancelled"
}
```

anything-but wildcard

The following policy property denies the `*ball` wildcard:

```
"customer_interests" : [{ "anything-but": { "wildcard": "*ball" }}]
```

It matches the following attributes:

```
{"customer_interests": ["hockey", "rugby", "soccer] }
```

However, it does not match the following message attribute:

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

anything-but suffix

The following policy property denies the `-ball`

suffix:

```
"customer_interests": [ { "anything-but": { "suffix": "ball" } } ]
```

It matches the following attributes:

```
{"customer_interests": ["hockey", "rugby", "soccer] }
```

However, it does not match the following message attribute:

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

Equals-ignore-case matching

When a policy property includes the keyword `equals-ignore-case`, it will perform a case-insensitive match with any message attribute or body property value.

Consider the following policy property:

```
"customer_interests": [{"equals-ignore-case": "tennis"}]
```

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
    "customer_interests": "TENNIS"
}
```

```
{
    "customer_interests": "teNnis"
{
```

IP address matching

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

Consider the following policy property:

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

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
   "source_ip": "10.0.0.0"
}
```

```
{
   "source_ip": "10.0.0.255"
}
```

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

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

Nor does it match the following message body:

```
{
   "source_ip": "10.1.1.0"
}
```

Prefix matching

When a policy property includes the keyword `prefix`, it matches any message attribute or body property values that begin with the specified characters.

Consider the following policy property:

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

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
   "customer_interests": "baseball"
}
```

```
{
   "customer_interests": "basketball"
}
```

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

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

Nor does it match the following message body:

```
{
    "customer_interests": "rugby"
}
```

Suffix matching

When a policy property includes the keyword `suffix`, it matches any message attribute or body property values that end with the specified characters.

Consider the following policy property:

```
"customer_interests": [{"suffix": "ball"}]
```

It matches either of the following message attributes:

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

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

It also matches either of the following message bodies:

```
{
    "customer_interests": "baseball"
}
```

```
{
    "customer_interests": "basketball"
}
```

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

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

Nor does it match the following message body:

```
{
    "customer_interests": "rugby"
}
```

Wildcard

You can use the wildcard character (*) to match string values in event patterns.

The following policy uses the wildcard (*) character:

```
"customer_interests": [ { "wildcard": "*ball" } ]
```

It matches the following attributes:

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

Applying a subscription filter policy in Amazon SNS

Message filtering in Amazon SNS allows you to selectively deliver messages to subscribers based on filter policies. These policies define conditions that messages must meet to be delivered to a subscription. While raw message delivery is an option that can affect message processing, it is not required for subscription filters to work.

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

Enabling Raw Message Delivery

Raw message delivery ensures that message payloads are delivered as-is to subscribers without any additional encoding or transformation. This can be useful when subscribers require the original message format for processing. However, raw message delivery is not directly related to the functionality of subscription filters.

Applying Subscription Filters

To apply message filters to a subscription, you define a filter policy using JSON syntax. This policy specifies the conditions that a message must meet to be delivered to the subscription. Filters can be based on message attributes, such as message attributes, message structure, or even message content.

Relationship between Raw Message Delivery and Subscription Filters

While enabling raw message delivery can affect how messages are delivered and processed by subscribers, it is not a prerequisite for using subscription filters. However, in scenarios where subscribers require the original message format without any modifications, enabling raw message delivery might be beneficial alongside subscription filters.

Considerations for Effective Filtering

When implementing message filtering, consider the specific requirements of your application and subscribers. Define filter policies that accurately match the criteria for message delivery to ensure efficient and targeted message distribution.

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

AWS Management Console

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Subscriptions**.
3. Select a subscription and then choose **Edit**.
4. On the **Edit** page, expand the **Subscription filter policy** section.
5. Choose between **attribute-based filtering** or **payload-based filtering**.
6. In the **JSON editor** field, provide the **JSON body** of your filter policy.
7. Choose **Save changes**.

   Amazon SNS applies your filter policy to the subscription.

AWS CLI

To apply a filter policy with the AWS Command Line Interface (AWS CLI), use the `set-subscription-attributes` command, as shown in the following example. For the `--attribute-name` option, specify `FilterPolicy`. For `--attribute-value`, specify your **JSON policy**.

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

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

If you want to switch from attribute-based (default) to payload-based message filtering, you can use the set-subscription-attributes command as well. For the `--attribute-name` option, specify `FilterPolicyScope`. For `--attribute-value`, specify `MessageBody`.

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns: ... --attribute-
name FilterPolicyScope --attribute-value MessageBody
```

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

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

AWS SDKs

The following code examples show how to use `SetSubscriptionAttributes`.

⚠ **Important**

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

CLI

AWS CLI

To set subscription attributes

The following `set-subscription-attributes` example sets the `RawMessageDelivery` attribute to an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name RawMessageDelivery \
    --attribute-value true
```

This command produces no output.

The following `set-subscription-attributes` example sets a `FilterPolicy` attribute to an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name FilterPolicy \
    --attribute-value "{ \"anyMandatoryKey\": [\"any\", \"of\", \"these\"] }"
```

This command produces no output.

The following `set-subscription-attributes` example removes the `FilterPolicy` attribute from an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name FilterPolicy \
    --attribute-value "{}"
```

This command produces no output.

- For API details, see SetSubscriptionAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.ArrayList;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class UseMessageFilterPolicy {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <subscriptionArn>

                Where:
                   subscriptionArn - The ARN of a subscription.

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

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        usePolicy(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void usePolicy(SnsClient snsClient, String subscriptionArn) {
        try {
            SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();
            // Add a filter policy attribute with a single value
            fp.addAttribute("store", "example_corp");
            fp.addAttribute("event", "order_placed");

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

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

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

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

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

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

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

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

Python

SDK for Python (Boto3)

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


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

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

- For API details, see SetSubscriptionAttributes in *AWS SDK for Python (Boto3) API Reference*.

Amazon SNS API

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

If you want to switch from attribute-based (default) to payload-based message filtering, you can use the `SetSubscriptionAttributes` action as well. Set the `AttributeName` parameter to `FilterPolicyScope`, and set the `AttributeValue` parameter to `MessageBody`.

AWS CloudFormation

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

1. Sign in to the AWS CloudFormation console.
2. Choose **Create Stack**.
3. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.

4. On the **Specify Details** page, do the following:

   a. For **Stack Name**, type `MyFilterPolicyStack`.

   b. For **myHttpEndpoint**, type the HTTP endpoint to be subscribed to your topic.

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

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

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

Removing a subscription filter policy in Amazon SNS

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

Using the AWS Management Console

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

   Amazon SNS applies your filter policy to the subscription.

Using the AWS CLI

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

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

Using the Amazon SNS API

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

Message data protection in Amazon SNS

What is message data protection?

Message data protection safeguards the data that's published to your Amazon SNS topics by using data protection policies to audit, mask, redact, or block the sensitive information that moves between applications or AWS services.

Message data protection scans data in motion for personally identifiable information (PII) and protected health information (PHI) using *data identifiers*. You can choose to use predefined (or Amazon SNS managed) data identifiers (for example, names, addresses, credit card numbers, and prescription drug codes), or you can create your own custom data identifiers, specific to your business use case. Using the scanned information, message data protection provides detailed audit logs, and allows you to take action to protect that data.

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

- **Audit** – Audit up to 99% of the data that's published to an Amazon SNS topic. You can then choose to send the findings to Amazon CloudWatch, Amazon S3, or https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html.
- **De-identify** – Mask or redact sensitive data without interrupting message publishing or delivering.
- **Deny** – Block the transmission of data between applications and AWS resources if sensitive data is present within the payload.

> **Note**
>
> Amazon SNS supports message data protection for Amazon SNS standard topics only.

Why should I use message data protection?

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

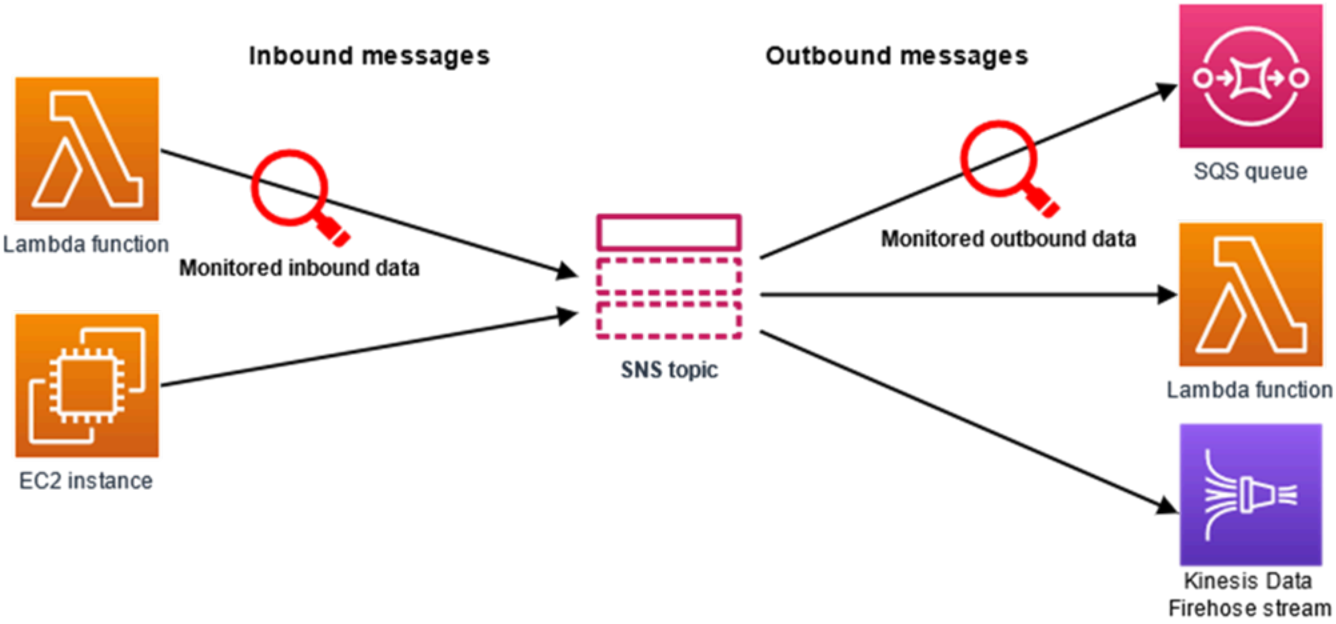
For example, you can use message data protection to create an *audit policy* to determine whether any of your systems are inadvertently sending or receiving sensitive data. If your audit results show that systems are sending credit card information to systems that don’t require it, you can use a *block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

> **Note**
>
> Amazon SNS supports message data protection for Amazon SNS standard topics only.

Understanding Amazon SNS data protection policies

What are data protection policies?

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

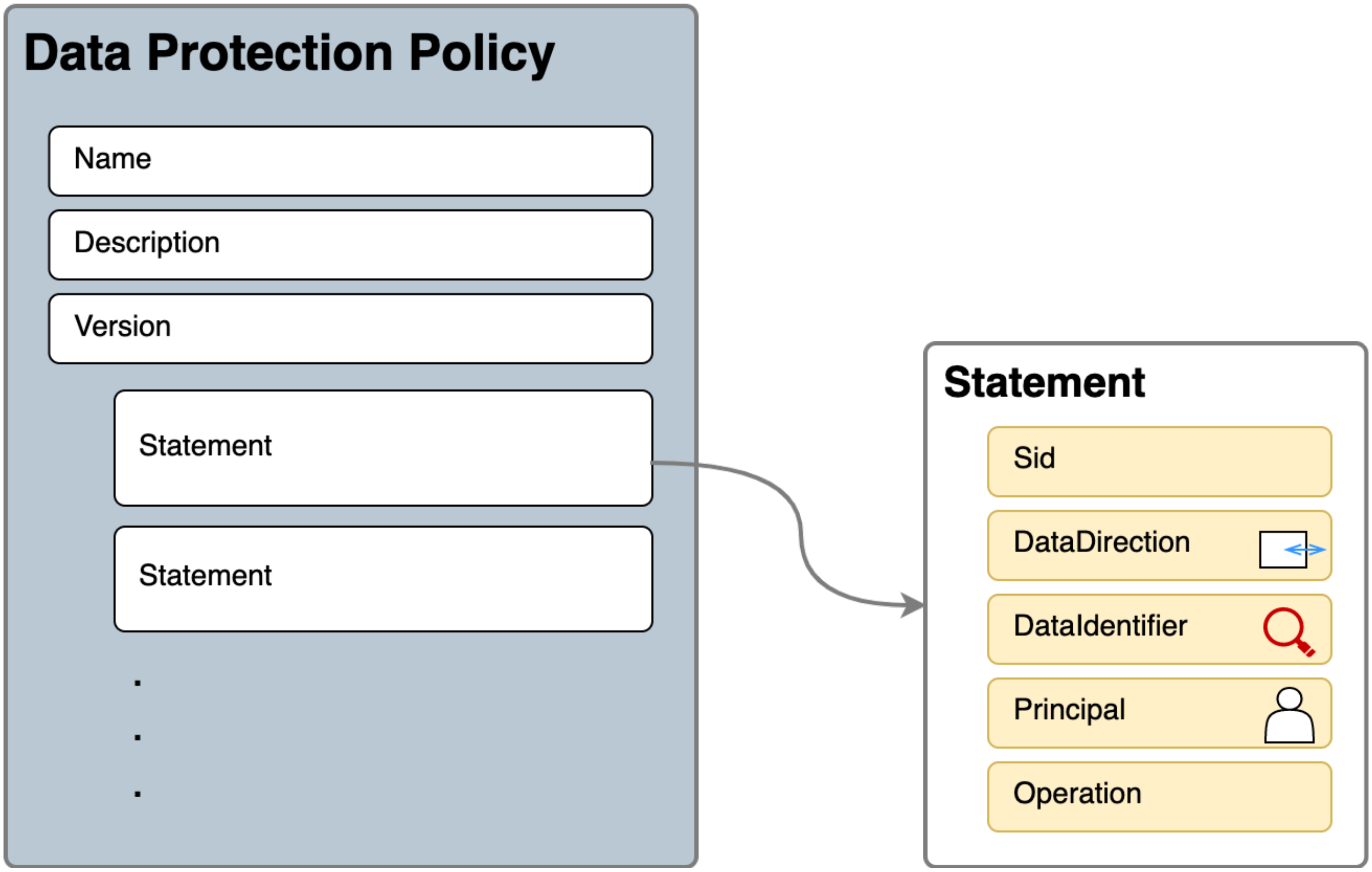


How is the data protection policy structured?

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

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

Each statement includes information about a single permission.



Only one data protection policy can be defined per Amazon SNS topic. The data protection policy can have one or more deny or de-identify statements, but only one audit statement.

JSON properties for the data protection policy

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

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

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

JSON properties for a policy statement

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

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

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

JSON properties for a policy statement operation

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

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

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

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

1. **Publish API Principal** (Inbound) – The authenticated IAM principal calling the Amazon SNS `Publish` API.
2. **Subscription Principal** (Outbound) – The authenticated IAM principal that called the `Subscribe` API during subscription creation.

The `SubscriptionPrincipal` is a publicly available Amazon SNS subscription property that can be retrieved from the `GetSubscriptionAttributes` API.

```
{
  "Attributes": {
    "SubscriptionPrincipal": "arn:aws:iam::123456789012:user/NoNameAccess",
    "Owner": "123412341234",
    "RawMessageDelivery": "true",
    "TopicArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
    "Endpoint": "arn:aws:sqs:us-east-1:123456789012:NoNameAccess",
    "Protocol": "sqs",
    "PendingConfirmation": "false",
    "ConfirmationWasAuthenticated": "true",
    "SubscriptionArn": "arn:aws:sns:us-east-1:123412341234:PII-data-
topic:5d8634ef-67ef-49eb-a824-4042b28d6f55"
  }
}
```

Data protection policy operations in Amazon SNS

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

Audit operation

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

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

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

- **Amazon CloudWatch Logs** (Optional) – The `LogGroup` must be in the topic region and the name must start with **/aws/vendedlogs/**.
- (Optional) – The `DeliveryStream` must be in the topic region and have **Direct PUT** as the source of delivery stream. For additional details, see Source, Destination, and Name in the *Amazon Data Firehose Developer Guide*.
- **Amazon S3** (Optional) – An Amazon S3 bucket name. Extra actions are required for using Amazon S3 bucket with SSE-KMS encryption enabled.

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

Required permissions when specifying log destinations

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

Audit destination	IAM permission
Default	logs:CreateLogDelivery logs:GetLogDelivery logs:UpdateLogDelivery logs:DeleteLogDelivery logs:ListLogDeliveries
CloudWatchLogs	logs:PutResourcePolicy logs:DescribeResourcePolicies logs:DescribeLogGroups
Firehose	iam:CreateServiceLinkedRole firehose:TagDeliveryStream
S3	s3:PutBucketPolicy s3:GetBucketPolicy Extra actions are required for using Amazon S3 bucket with SSE-KMS encryption enabled.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-1:123456789012:SampleLogGroupName:*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole",
                "firehose:TagDeliveryStream"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutBucketPolicy",
                "s3:GetBucketPolicy"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

Required key policy for use with SSE-KMS

If you use an Amazon S3 bucket as a log destination, you can protect the data in your bucket by enabling either Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3), or Server-Side Encryption with AWS KMS keys (SSE-KMS). For more information, see Protecting data using server-side encryption in the *Amazon S3 User Guide*.

If you choose SSE-S3, no additional configuration is required. Amazon S3 handles the encryption key.

If you choose SSE-KMS, you must use a customer managed key. You must update the key policy for your customer managed key so that the log delivery account can write to your S3 bucket. For more information about the required key policy for use with SSE-KMS, see Amazon S3 bucket server-side encryption in the *Amazon CloudWatch Logs User Guide*.

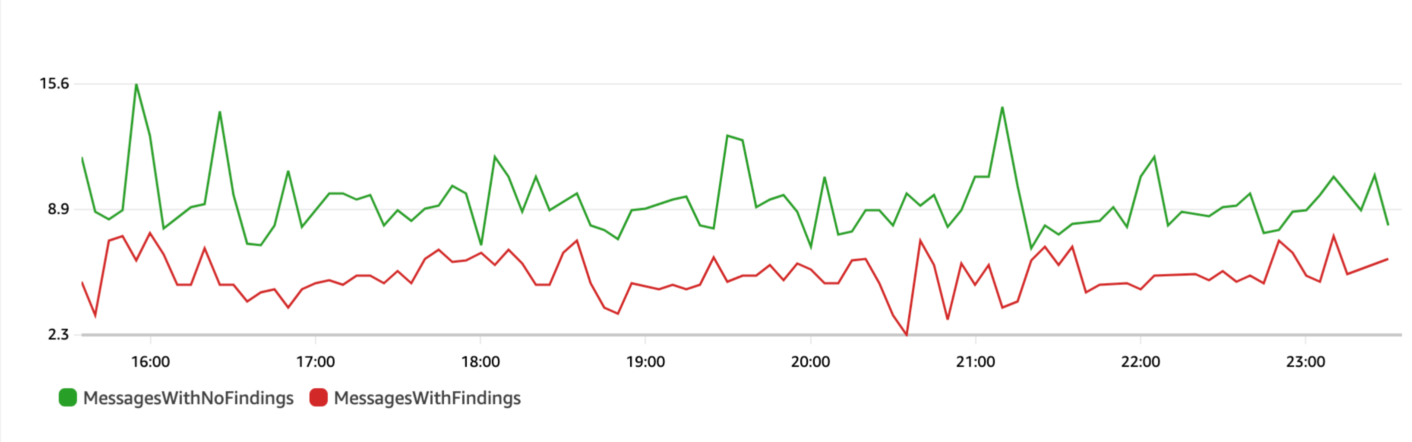
Audit destination log example

In the following example, `callerPrincipal` is used to identify the source of the sensitive content, and `messageID` is used as a reference to check against the `Publish` API response.

```
{
  "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
  "auditTimestamp": "2022-05-12T2:10:44Z",
  "callerPrincipal": "arn:aws:iam::123412341234:role/Publisher",
  "resourceArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
  "dataIdentifiers": [
    {
      "name": "Name",
      "count": 1,
      "detections": [
        {
          "start": 1,
          "end": 2
        }
      ]
    },
    {
      "name": "PhoneNumber",
      "count": 2,
      "detections": [
        {
          "start": 3,
          "end": 4
        },
        {
          "start": 5,
          "end": 6
        }
      ]
    }
  ]
}
```

Audit operation metrics

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



De-identify operation

The **De-identify** operation masks or redacts sensitive data from published or delivered messages. This operation is available for both inbound and outbound messages, and requires one of the following types of configurations:

- **MaskConfig** – Mask using a supported character from the following table. For example, ssn: `123-45-6789` becomes ssn: ###########.

```
{
"Operation": {
    "Deidentify": {
        "MaskConfig": {
            "MaskWithCharacter": "#"
          }
    }
}
```

Supported mask character	Name
*	Asterisk
A-Z, a-z, and 0-9	Alphanumeric
	Space
!	Exclamation mark
$	Dollar sign
%	Percent sign
&	Ampersand
()	Parenthesis
+	Plus sign
,	Comma
-	Hyphen
.	Period
/\	Slash, back slash
#	Number sign

Supported mask character	Name
:	Colon
;	Semicolon
=, <>	Equals. less or greater than
@	At sign
[]	Brackets
^	Caret symbol
_	Underscore
`	Backtick
|	Vertical bar
~	Tilde symbol

- **RedactConfig** – Redact by removing the data entirely. For example, ssn: 123-45-6789 becomes ssn: .

```
{
"Operation": {
    "Deidentify": {
        "RedactConfig": {}
    }
}
```

On an inbound message, the sensitive data is de-identified after the audit operation, and the SNS:Publish API caller receives the following invalid parameter error when the entire message is sensitive.

```
Error code: AuthorizationError ...
```

Deny operation

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

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

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

```
Error code: AuthorizationError ...
```

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

```
{
    "notification": {
        "messageMD5Sum": "29638742ffb68b32cf56f42a79bcf16b",
        "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
        "topicArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
        "timestamp": "2022-05-12T2:12:44Z"
    },
    "delivery": {
        "deliveryId": "98236591c-56aa-51ee-a5ed-0c7d43493170",
        "destination": "arn:aws:sqs:us-east-1:123456789012:NoNameAccess",
        "providerResponse": "The topic's data protection policy prohibits this message
 from being delivered to <subscription-arn>",
        "dwellTimeMs":20,
        "attempts":1,
        "statusCode": 403
    },
    "status": "FAILURE"
}
```

Amazon SNS data protection policy examples

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

Example policy for auditing

Audit policies allow you to audit up to 99% of inbound messages and send findings to Amazon CloudWatch, https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html, and Amazon S3.

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

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

Data protection policy:

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

Audit results format example:

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

Example policy with inbound de-identify mask statement

The following example prevents a user from publishing a message to a topic with `CreditCardNumber` by masking the sensitive data from the message content.

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

Inbound de-identify mask results example:

```
// original message
My credit card number is 4539894458086459

// delivered message
My credit card number is ################
```

Example policy with inbound de-identify redact statement

The following example prevents a user from publishing a message to a topic with `CreditCardNumber` by redacting the sensitive data from the message content.

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

Inbound de-identify redact results example:

```
// original message
My credit card number is 4539894458086459

// delivered message
My credit card number is
```

Example policy with outbound de-identify mask statement

The following example prevents a user from receiving a message with `CreditCardNumber` by masking the sensitive data from the message content.

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

Outbound de-identify mask results example:

```
// original message
My credit card number is 4539894458086459

// delivered message
My credit card number is ----------------
```

Example policy with outbound de-identify redact statement

The following example prevents a user from receiving a message with `CreditCardNumber` by redacting the sensitive data from the message content.

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

Outbound de-identify redact results example:

```
// original message
My credit card number is 4539894458086459

// delivered message
My credit card number is
```

Example policy with inbound deny statement

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

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

Example policy with outbound deny statement

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

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

Outbound deny results example, logged in Amazon CloudWatch:

```
{
  "notification": {
    "messageMD5Sum": "2e8f58ff2eeed723b56b15493fbfb5a5",
    "messageId": "8747a956-ebf1-59da-b291-f2c2e4b87c9c",
    "topicArn": "arn:aws:sns:us-east-2:664555388960:test1",
    "timestamp": "2022-09-08 15:40:57.144"
  },
  "delivery": {
    "deliveryId": "6a422437-78cc-5171-ad64-7fa3778507aa",
    "destination": "arn:aws:sqs:us-east-2:664555388960:test",
    "providerResponse": "The topic's data protection policy prohibits this message from
 being delivered to <subscription arn>",
    "dwellTimeMs": 22,
    "attempts": 1,
    "statusCode": 403
  },
  "status": "FAILURE"
}
```

Creating data protection policies in Amazon SNS

Data protection policies help you safeguard the data that's published to your Amazon SNS topics by auditing, de-identifying (masking or redacting), and denying (blocking) sensitive information that moves between applications or AWS services. You can use AWS API, AWS CLI, AWS CloudFormation, or AWS Management Console to create data protection policies in Amazon SNS. Only one policy can be defined per Amazon SNS topic. Each data protection policy can have one or more de-identify and deny statements, but only one audit statement.

Topics

- Creating data protection policies in Amazon SNS using the CLI
- Creating data protection policies in Amazon SNS using CloudFormation
- Creating data protection policies in Amazon SNS using the console
- Creating Amazon SNS data protection policies to secure message data using the SDK

Creating data protection policies in Amazon SNS using the API

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

Creating a data protection policy using API

Create an Amazon SNS data protection policy using the AWS API.

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

Use the `DataProtectionPolicy` property of a standard Amazon SNS topic:

- `CreateTopic`

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

Call one of the following operations:

- GetDataProtectionPolicy
- PutDataProtectionPolicy

Creating data protection policies in Amazon SNS using the CLI

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

Creating data protection policies using the AWS CLI

Create an Amazon SNS data protection policy using the AWS Command Line Interface.

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

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

- create-topic

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

Call one of the following operations:

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

Creating data protection policies in Amazon SNS using CloudFormation

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

Creating data protection policies (CloudFormation)

Create an Amazon SNS data protection policy using AWS CloudFormation.

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

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

- AWS::SNS::Topic

Creating data protection policies in Amazon SNS using the console

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

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

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

1. Sign in to the Amazon SNS console.
2. Choose a topic or create a new one. For more details on creating topics, see Creating an Amazon SNS topic.
3. On the **Create topic** page, in the **Details** section, choose **Standard**.

   a. Enter a **Name** for the topic.

b. (Optional) Enter a **Display name** for the topic.

4. Expand **Data protection policy**.

5. Choose a **Configuration mode**:

   - **Basic –** Define a data protection policy using a simple menu.

   - **Advanced –** Define a custom data protection policy using JSON.

6. (Optional) To create your own **custom data identifier**, expand the **Custom data identifier configuration section** do the following:

   a. Enter a unique **name** for the custom data identifier. Custom data identifier names support alphanumeric, underscore (_), and hyphen (-) characters. Up to 128 character are supported. This name cannot share the same name as a managed data identifier. For a full list of custom data identifier limitations, see Custom data identifier constraints.

   b. Enter a regular expression (RegEx) for the custom data identifier. RegEx supports alphanumeric characters, RegEx reserved characters, and symbols. RegEx has a maximum length of 200 characters. If the RegEx is too complicated, Amazon SNS will fail the API call. For a full list of RegEx limitations, see Custom data identifier constraints.

   c. (Optional) Choose **Add custom data identifier** to add additional data identifiers as needed. A maximum of 10 custom data identifiers are supported for each data protection policy.

7. Choose the statement(s) that you'd like to add to your data protection policy. You can add **audit**, **de-identify** (mask or redact), and **deny** (block) statement types to the same data protection policy.

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

   > **Note**
   >
   > Only one audit statement is allowed per data protection policy or topic.

      i. Select **data identifiers** to define the sensitive data that you want to audit.

      ii. For **Audit sample rate**, enter the percentage of messages to audit for sensitive information, up to a maximum of 99%.

  iii. For **Audit destination**, select which AWS services to send the audit finding results, and enter a destination name for each AWS service that you use. You can select from the following Amazon Web Services:

    - **Amazon CloudWatch** – CloudWatch Logs is the AWS standard logging solution. Using CloudWatch Logs, you can perform log analytics using Logs Insights (see samples here) and create metrics and alarms. CloudWatch Logs is where many services publish logs, which makes it easier to aggregate all logs using one solution. For information about Amazon CloudWatch, see the Amazon CloudWatch User Guide.
    - – Firehose satisfies the demands for real-time streaming to Splunk, OpenSearch, and Amazon Redshift for further log analytics. For information about , see the User Guide.
    - **Amazon Simple Storage Service** – Amazon S3 is an economical log destination for archival purposes. You may be required to retain logs for a period of years. In this case, you can put logs into Amazon S3 to save costs. For information about Amazon Simple Storage Service, see the Amazon Simple Storage Service User Guide.

b. **Add a de-identify statement** – Configure the sensitive data you want to de-identify in the message, whether you want to mask or redact that data, and the accounts to stop delivery of that data.

  i. For **Data identifiers**, select the sensitive data that you want to de-identify.

  ii. For **Define this de-identify statement for**, select the AWS accounts or IAM principals to which this de-identify statement applies. You can apply it to **all AWS accounts**, or to **specific AWS accounts** or **IAM entities** (account roots, roles, or users) that use account IDs or IAM entity ARNs. Separate multiple IDs or ARNs using a comma (,).

  The following IAM principals are supported:

    - **IAM account principals** – For example,`arn:aws:iam::AWS-account-ID:root`.
    - **IAM role principals** – For example, `arn:aws:iam::AWS-account-ID:role/role-name`.
    - **IAM user principals** – For example, `arn:aws:iam::AWS-account-ID:user/user-name`.

  iii. For **De-identify Option**, select how you want to de-identify the sensitive data. The following options are supported:

    - **Redact** – Completely removes data. For example, email: `classified@amazon.com` becomes email: .
    - **Mask** – Replaces the data with single characters. For example, email: `classified@amazon.com` becomes email: `*********************`.

   iv. (Optional) Continue to add de-identify statements as needed.

c. **Add deny statement** – Configure which sensitive data to prevent from moving through your topic, and which principals to prevent from delivering that data.

   i. For **data direction** , choose the direction of the messages for the deny statement:

    - **Inbound messages** – Apply this deny statement to messages that are sent to the topic.
    - **Outbound messages** – Apply this deny statement to messages that the topic delivers to subscription endpoints.

   ii. Choose the **data identifiers** to define the sensitive data that you want to deny.

   iii. Choose the **IAM principals** that apply to this deny statement. You can apply it to **all AWS accounts**, to **specific AWS accounts**, or **IAM entities** (for example, account roots, roles, or users) that use account IDs or IAM entity ARNs. Separate multiple IDs or ARNs using a comma (,). The following IAM principals are supported:

    - **IAM account principals** – For example, `arn:aws:iam::AWS-account-ID:root`.
    - **IAM role principals** – For example, `arn:aws:iam::AWS-account-ID:role/role-name`.
    - **IAM user principals** – For example, `arn:aws:iam::AWS-account-ID:user/user-name`.

   iv. (Optional) Continue to add deny statements as needed.

Creating Amazon SNS data protection policies to secure message data using the SDK

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

Creating data protection policies using the AWS SDK

Create an Amazon SNS data protection policy using the AWS SDK.

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

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

Java

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

public static String createSNSTopicWithDataProtectionPolicy(SnsClient snsClient,
 String topicName, String dataProtectionPolicy) {

    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .dataProtectionPolicy(dataProtectionPolicy)
                .build();

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

JavaScript

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

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

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

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

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

Java

```

public static void putDataProtectionPolicy(SnsClient snsClient, String topicName,
 String dataProtectionPolicy) {

    try {
        PutDataProtectionPolicyRequest request =
 PutDataProtectionPolicyRequest.builder()
                .resourceArn(topicName)
                .dataProtectionPolicy(dataProtectionPolicy)
                .build();

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


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

        GetDataProtectionPolicyResponse result =
 snsClient.getDataProtectionPolicy(request);

        System.out.println("\n\nStatus is " + result.sdkHttpResponse().statusCode()
        + "\n\nDataProtectionPolicy: \n\n" + result.dataProtectionPolicy());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

JavaScript

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

// Set the parameters
const putParams = { ResourceArn: "TOPIC_ARN", DataProtectionPolicy:
 "DATA_PROTECTION_POLICY" };

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

// Set the parameters
const getParams = { ResourceArn: "TOPIC_ARN" };
const runGet = async () => {
  try {
    const data = await snsClient.send(new
 GetDataProtectionPolicyCommand(getParams));
    console.log("Success.",  data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
runGet();
```

Deleting data protection policies in Amazon SNS

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

For general information about Amazon SNS data protection policies, see Understanding Amazon SNS data protection policies.

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

Deleting data protection policies using the console

To delete a managed data protection policy using the console

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

Deleting a data protection policy using an empty JSON string

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

Deleting a data protection policy using the AWS CLI

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

```
//aws sns put-data-protection-policy --resource-arn topic-arn --data-
protection-policy ""
```

Amazon SNS data identifiers

Amazon SNS uses a combination of criteria and techniques, including machine learning and pattern matching, to detect sensitive data. These criteria and techniques, collectively referred to as *data identifiers*, can detect a large and growing list of sensitive data types for many countries and regions. Amazon SNS managed data identifiers offer preconfigured data types for protecting financial data, personal health information (PHI), and personally identifiable information (PII). You can also use custom data identifiers to create your own data identifiers tailored to your specific use case.

Using managed data identifiers in Amazon SNS

What are managed data identifiers?

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

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

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

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

Keyword requirements

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

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

Amazon SNS managed data identifiers for sensitive data types

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

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

Identifier	Category	Countries/Languages
BankAccountNumber	Financial	DE, ES, FR, GB, IT
CepCode	Personal	BR
Cnpj	Personal	BR
CpfCode	Personal	BR
DriversLicense	Personal	AT, AU, BE, BG, CA, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IT, LT, LU, LV, MT, NL, PL, PT, RO, SE, SI, SK, US
DrugEnforcementAgencyNumber	Health	US

Identifier	Category	Countries/Languages
ElectoralRollNumber	Personal	GB
HealthInsuranceCardNumber	Health	EU
HealthInsuranceClaimNumber	Health	US
HealthInsuranceNumber	Health	FR
HealthcareProcedureCode	Health	US
IndividualTaxIdentification Number	Personal	US
InseeCode	Personal	FR
MedicareBeneficiaryNumber	Health	US
NationalDrugCode	Health	US
NationalIdentificationNumber	Personal	DE, ES, IT
NationalInsuranceNumber	Personal	GB
NationalProviderId	Health	US
NhsNumber	Health	GB
NieNumber	Personal	ES
NifNumber	Personal	ES
PassportNumber	Personal	CA, DE, ES, FR, GB, IT, US
PermanentResidenceNumber	Personal	CA
PersonalHealthNumber	Health	CA
PhoneNumber	Personal	BR, DE, ES, FR, GB, IT, US
PostalCode	Personal	CA

Identifier	Category	Countries/Languages
RgNumber	Personal	BR
SocialInsuranceNumber	Personal	CA
Ssn	Personal	ES, US
TaxId	Personal	DE, ES, FR, GB
ZipCode	Personal	US

Supported Identifiers that are language/region independent

Identifier	Category
Address	Personal
AwsSecretKey	Credentials
CreditCardExpiration	Financial
CreditCardNumber	Financial
CreditCardSecurityCode	Financial
EmailAddress	Personal
IpAddress	Personal
LatLong	Personal
Name	Personal
OpenSshPrivateKey	Credentials
PgpPrivateKey	Credentials
PkcsPrivateKey	Credentials

Identifier	Category
PuttyPrivateKey	Credentials
VehicleIdentificationNumber	Personal

Amazon SNS sensitive data types: Credentials

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

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

Data identifier ARNs for credential data types

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

Credential data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/AwsSecretKey
arn:aws:dataprotection::aws:data-identifier/OpenSshPrivateKey
arn:aws:dataprotection::aws:data-identifier/PgpPrivateKey
arn:aws:dataprotection::aws:data-identifier/PkcsPrivateKey
arn:aws:dataprotection::aws:data-identifier/PuttyPrivateKey

Amazon SNS sensitive data types: Devices

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

Detection type	Managed data identifier ID	Keyword required	Countries and regions
IP Address	IpAddress	No	Any

Data identifier ARNs for device data types

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

Device data identifier ARN
arn:aws:dataprotection::aws:data-identifier/IpAddress

Amazon SNS sensitive data types: Financial

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

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Bank account number	BankAccountNumber BankAccountNumber-US	Yes, see Keywords for bank account numbers.	This includes: International Bank Account Numbers (IBANs) that consist of up to 34 alphanumeric characters, including elements such as country code.	France, Germany, Italy, Spain, UK
Credit card expiration date	CreditCardExpiration	exp d, exp m, exp y, expiration, expiry	–	Any
Credit card magnetic strip data	CreditCardMagneticStripe	Yes, including: card data, iso7813, mag, magstripe, stripe, swipe.	This includes tracks 1 and 2.	Any
Credit card number	CreditCardNumber	account number, american express, amex, bank card, card, card num, card number, cc #, ccn, check card, credit, credit card#, dankort, debit, debit card, diners club, discover,	Detection requires the data to be a 13–19 digit sequence that adheres to the Luhn check formula, and uses a standard card number prefix for any of the following types of credit	Any

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
		electron, elo verification code, japanese card bureau, jcb, mastercard, mc, pan, payment account number, payment card number, pcn, union pay, visa	cards: American Express, Dankort, Diner’s Club, Discover, Electron, Japanese Card Bureau (JCB), Mastercard, UnionPay, and Visa (superscript link below 1).	
Credit card verification code	CreditCardSecurityCode	card id, card identification code, card identification number, card security code, card validation code, card validation number, card verification data, card verification value, cvc, cvc2, cvv, cvv2, elo verification code	–	Any

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

   122000000000003, 2222405343248877, 2222990905257051, 2223007648726984, 2223577120017656, 30569309025904, 34343434343434, 3528000700000000, 3530111333300000, 3566002020360505, 36148900647913, 36700102000000, 371449635398431, 378282246310005, 378734493671000, 38520000023237, 4012888888881881, 4111111111111111, 4222222222222, 4444333322221111, 4462030000000000, 4484070000000000, 4911830000000, 4917300800000000, 4917610000000000, 4917610000000000003, 5019717010103742, 5105105105105100, 5111010030175156, 5185540810000019, 5200828282828210, 5204230080000017, 5204740009900014, 5420923878724339, 5454545454545454, 5455330760000018, 5506900490000436, 5506900490000444, 5506900510000234, 5506920809243667, 5506922400634930, 5506927427317625, 5553042241984105, 5555553753048194, 5555555555554444, 5610591081018250, 6011000990139424, 6011000400000000, 6011111111111117, 630490017740292441, 630495060000000000, 6331101999990016, 6759649826438453, 6799990100000000019, and 76009244561.

Keywords for bank account numbers

Use the following keywords to detect International Bank Account Numbers (IBANs) that consist of up to 34 alphanumeric characters, including elements such as country code.

Country or region	Keywords			
France	account code, account number, accountno#, accountnumber#, bban, code bancaire, compte bancaire, customer account id, customer account number, customer bank account id, iban,			

Country or region	Keywords
	numéro de compte
Germany	account code, account number, accountno#, accountnumber#, bankleitzahl, bban, customer account id, customer account number, customer bank account id, geheimzahl, iban, kartennummer, kontonummer, kreditkartennummer, sepa

Country or region	Keywords			
Italy	account code, account number, accountno#, accountnumber#, bban, codice bancario, conto bancario, customer account id, customer account number, customer bank account id, iban, numero di conto			

Country or region	Keywords			
Spain	account code, account number, accountno#, accountnumber#, bban, código cuenta, código cuenta bancaria, cuenta cliente id, customer account ID, customer account number, customer bank account id, iban, número cuenta bancaria cliente, número cuenta cliente			
UK	account code, account number, accountno#, accountnumber#, bban, customer account id, customer account number, customer bank account id, iban, sepa			

Country or region	Keywords			
US	bank account, bank acct, checking account, checking acct, deposit account, deposit acct, savings account, savings acct, chequing account, chequing acct			

Data identifier ARNs for financial data types

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

Financial data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-DE
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-ES
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-FR
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-GB
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-IT
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-US
arn:aws:dataprotection::aws:data-identifier/CreditCardExpiration
arn:aws:dataprotection::aws:data-identifier/CreditCardNumber

Financial data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/CreditCardSecurityCode

Amazon SNS sensitive data types: Protected health information (PHI)

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

Detection type	Managed data identifier ID	Keyword required	Countries and regions
Drug Enforcement Agency (DEA) Registration Number	DrugEnforcementAgencyNumber	dea number, dea registration	US
Health Insurance Card Number (EHIC)	HealthInsuranceCardNumber	assicurazione sanitaria numero, carta assicurazione numero, carte d’assurance maladie, carte européenne d'assurance maladie, ceam, ehic, ehic#, finlandehicnumber#, gesundheitskarte, hälsokort, health card, health card number, health insurance card, health insurance number, insurance card number, krankenversicherungskarte, krankenversicherungsnummer, medical account number,	EU

Detection type	Managed data identifier ID	Keyword required	Countries and regions
		numero conto medico, numéro d’assurance maladie, numéro de carte d’assurance, numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanho itokortin, sairausva kuutuskortti, sairausvakuutusnum ero, sjukförsäkring nummer, sjukförsä kringskort, suomi ehic-numero, tarjeta de salud, terveysko rtti, tessera sanitaria assicurazione numero, versicher ungsnummer	
Health Insurance Claim Number (HICN)	HealthInsuranceCla imNumber	health insurance claim number, hic no, hic no., hic number, hic#, hicn, hicn#., hicno#	US
Health insurance or medical identification number	HealthInsuranceNum ber	carte d'assuré social, carte vitale, insurance card	FR

Detection type	Managed data identifier ID	Keyword required	Countries and regions
Healthcare Common Procedure Coding System (HCPCS) code	HealthcareProcedureCode	current procedural terminology, hcpcs, healthcare common procedure coding system	US
Medicare Beneficiary Number (MBN)	MedicareBeneficiaryNumber	mbi, medicare beneficiary	US
National Drug Code (NDC)	NationalDrugCode	national drug code, ndc	US
National Provider Identifier (NPI)	NationalProviderId	hipaa, n.p.i, national provider, npi	US
National Health Service (NHS) Number	NhsNumber	national health service, NHS	GB
Personal Health Number (PHN)	PersonalHealthNumber	canada healthcare number, msp number, personal healthcare number, phn, soins de santé	CA

Keywords for health insurance and medical identification numbers

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

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

Country or region	Keywords
Canada	Canada healthcare number, msp number, personal healthcare number, phn, soins de santé
EU	assicurazione sanitaria numero, carta assicurazione numero, carte d’assurance maladie, carte européenne d'assurance maladie, ceam, ehic, ehic#, finlandehicnumber#, gesundheitskarte, hälsokort, health card, health card number, health insurance card, health insurance number, insurance card number, krankenversicherungskarte, krankenversicherungsnummer, medical account number, numero conto medico, numéro d’assurance maladie, numéro de carte d’assurance, numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanhoitokortin, sairausvakuutuskortti, sairausvakuutusnumero, sjukförsäkring nummer, sjukförsäkringskort, suomi ehic-numero, tarjeta de salud, terveyskortti, tessera sanitaria assicurazione numero, versicherungsnummer
Finland	ehic, ehic#, finland health insurance card, finlandehicnumber#, finska sjukförsäkringskort, hälsokort, health card, health card number, health insurance card, health insurance number, sairaanhoitokortin, sairaanhoitokortin, sairausvakuutuskortti, sairausvakuutusnumero, sjukförsäkring nummer, sjukförsäkringskort, suomen sairausvakuutuskortti, suomi ehic-numero, terveyskortti

Country or region	Keywords
France	carte d'assuré social, carte vitale, insurance card
UK	national health service, NHS
US	mbi, medicare beneficiary

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

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

PHI data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/DrugEnforcementAgencyNumber-US
arn:aws:dataprotection::aws:data-identifier/HealthcareProcedureCode-US
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceCardNumber-EU
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceClaimNumber-US
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceNumber-FR
arn:aws:dataprotection::aws:data-identifier/MedicareBeneficiaryNumber-US
arn:aws:dataprotection::aws:data-identifier/NationalDrugCode-US
arn:aws:dataprotection::aws:data-identifier/NationalInsuranceNumber-GB
arn:aws:dataprotection::aws:data-identifier/NationalProviderId-US
arn:aws:dataprotection::aws:data-identifier/NhsNumber-GB
arn:aws:dataprotection::aws:data-identifier/PersonalHealthNumber-CA

Amazon SNS sensitive data types: Personally identifiable information (PII)

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

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Birth date	DateOfBirth	dob, date of birth, birthdate , birth date, birthday, b-day, bday	Support includes most date formats, such as all digits and combinations of digits and names of months. Date components can be separated by spaces, slashes (/), or hyphens (-).	Any
Código de Endereçamento Postal (CEP)	CepCode	cep, código de endereçamento postal, codigo de endereçam ento postal	–	Brazil
Cadastro Nacional da Pessoa Jurídica (CNPJ)	Cnpj	cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj	–	Brazil
Cadastro de Pessoas Físicas (CPF)	CpfCode	Cadastro de pessoas fisicas, cadastro de	–	Brazil

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
		pessoas físicas, cadastro de pessoa física, cadastro de pessoa fisica, cpf		
Driver's license identification number	DriversLicense	Yes, see Keywords for driver's license identification numbers.	–	Australia, Austria, Belgium, Bulgaria, Canada, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Latvia, Lithuania , Luxembourg, Malta, Netherlands, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, UK, US

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Electoral roll number	ElectoralRollNumber	electoral#, electoral #, electoralnumber, electoral number, electoralroll#, electoral roll#, electoral roll #, electoral roll no., electoral roll number, electoralrollno	–	UK
Individual taxpayer identification	IndividualTaxIdentificationNumber	Yes, see Keywords for taxpayer identification and reference numbers.	–	US
National Institute for Statistics and Economic Studies (INSEE)	InseeCode	Yes, see Keywords for national identification numbers.	–	France

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
National identification number	NationalIdentificationNumber	Yes, see Keywords for national identification numbers.	This includes Documento Nacional de Identidad (DNI) identifiers (Spain), Codice fiscale codes (Italy), and National Identity Card numbers (German).	Germany, Italy, Spain
National Insurance Number (NINO)	NationalInsuranceNumber	insurance no., insurance number, insurance #, national insurance number, nationalinsurance#, nationalinsurancenumber, nin, nino	–	UK
Número de identidad de extranjero (NIE)	NieNumber	Yes, see Keywords for taxpayer identification and reference numbers.	–	Spain

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Número de Identificación Fiscal (NIF)	NifNumber	Yes, see Keywords for taxpayer identification and reference numbers.	–	Spain
Passport number	PassportNumber	Yes, see Keywords for passport numbers.	–	Canada, France, Germany, Italy, Spain, UK, US

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Permanent residence number	Permanent Residence Number	carte résident permanent , numéro carte résident permanent, numéro résident permanent , permanent resident card, permanent resident card number, permanent resident no, permanent resident no., permanent resident number, pr no, pr no., pr non, pr number, résident permanent no., résident permanent non	–	Canada

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Phone number	PhoneNumber	Brazil: keywords also include: cel, celular, fone, móvel, número residencial, numero residencial, telefone Others: cell, contact, fax, fax number, mobile, phone, phone number, tel, telephone , telephone number	This includes toll-free numbers in the US and fax numbers. If a keyword is in proximity of the data, the number doesn't have to include a country code. If a keyword isn't in proximity of the data, the number has to include a country code.	Brazil, Canada, France, Germany, Italy, Spain, UK, US
Postal Code	PostalCode	No	–	Canada
Registro Geral (RG)	RgNumber	Yes, see Keywords for national identification numbers.	–	Brazil
Social Insurance Number (SIN)	SocialInsuranceNumber	canadian id, numéro d'assurance sociale, social insurance number, sin	–	Canada

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Social Security number (SSN)	Ssn	Spain – número de la seguridad social, social security no., social security no. número de la seguridad social, social security number, socialsec urityno#, ssn, ssn# US – social security, ss#, ssn	–	Spain, US
Taxpayer identification or reference number	TaxId	Yes, see Keywords for taxpayer identification and reference numbers.	This includes TIN (France); Steueride ntifikati onsnummer (Germany); CIF (Spain); and TRN, UTR (UK).	France, Germany, Spain, UK
US postal code	ZipCode	zip code, zip+4	–	US

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Mailing address	Address	No	Although a keyword isn't required, detection requires the address to include the name of a city or place and a ZIP or Postal Code.	Australia, Canada, France, Germany, Italy, Spain, UK, US
Electronic mail address	EmailAddress	email, email address, e mail, e mail address	–	Any

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Global Positioning System (GPS) coordinates	LatLong	coordinate, coordinates, lat long, latitude longitude, location, position	Amazon SNS can detect GPS coordinates if the latitude and longitude coordinates are stored as a pair and they're in Decimal Degrees (DD) format, for example, 41.948614,-87.655311. Support doesn't include coordinates in Degrees Decimal Minutes (DDM) format, for example 41°56.9168'N 87°39.3187'W, or Degrees, Minutes, Seconds (DMS) format, for example 41°56'55.0104"N 87°39'19.1196"W.	Any

Detection type	Managed data identifier ID	Keyword required	Additional information	Countries and regions
Full name	Name	No	Amazon SNS can detect full names only. Support is limited to Latin character sets.	Any
Vehicle identification number (VIN)	VehicleIdentificationNumber	Fahrgestellnummer, niv, numarul de identificare, numarul seriei de sasiu, serie sasiu, numer VIN, Número de Identificação do Veículo, Número de Identificación de Automóviles, numéro d'identification du véhicule, vehicle identification number, vin, VIN numeris	Amazon SNS can detect VINs that consist of a 17-character sequence and adhere to the ISO 3779 and 3780 standards. These standards were designed for worldwide use.	Any

Keywords for driver’s license identification numbers

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

Country or region	Keywords
Australia	dl# dl:, dl :, dlno# driver licence, driver license, driver permit, drivers lic., drivers licence, driver's licence, drivers license, driver's license, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
Austria	führerschein, fuhrerschein, führerschein republik österreich, fuhrerschein republik osterreich
Belgium	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, führerschein, fuhrerschein- nr, führerschein- nr, fuhrersch einnummer, führerscheinnummer, numéro permis conduire, permis de conduire, rijbewijs, rijbewijsnummer
Bulgaria	превозно средство, свидетелство за управление на моторно, свидетелство за управление на мпс, сумпс, шофьорска книжка
Canada	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit, permis de conduire
Croatia	vozačka dozvola
Cyprus	άδεια οδήγησης

Country or region	Keywords
Czech Republic	číslo licence, císlo licence řidiče, číslo řidičskéh o průkazu, ovladače lic., povolení k jízdě, povolení řidiče, řidiči povolení, řidičský prúkaz, řidičský průkaz
Denmark	kørekort, kørekortnummer
Estonia	juhi litsentsi number, juhiloa number, juhiluba, juhiluba number
Finland	ajokortin numero, ajokortti, förare lic., körkort, körkort nummer, kuljettaja lic., permis de conduire
France	permis de conduire
Germany	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, führerschein, fuhrerschein- nr, führerschein- nr, fuhrersch einnummer, führerscheinnummer
Greece	δεια οδήγησης, adeia odigisis
Hungary	illesztőprogramok lic, jogosítvány, jogsi, licencszám, vezető engedély, vezetői engedély
Ireland	ceadúnas tiomána
Italy	patente di guida, patente di guida numero, patente guida, patente guida numero
Latvia	autovadītāja apliecība, licences numurs, vadītāja apliecība, vadītāja apliecības numurs, vadītāja atļauja, vadītāja licences numurs, vadītāji lic.
Lithuania	vairuotojo pažymėjimas

Country or region	Keywords
Luxembourg	fahrerlaubnis, führerschäin
Malta	liċenzja tas-sewqan
Netherlands	permis de conduire, rijbewijs, rijbewijsnummer
Poland	numer licencyjny, prawo jazdy, zezwolenie na prowadzenie
Portugal	carta de condução, carteira de habilitação, carteira de motorist, carteira habilitação, carteira motorist, licença condução, licença de condução, número de licença, número licença, permissão condução, permissão de condução
Romania	numărul permisului de conducere, permis de conducere
Slovakia	číslo licencie, číslo vodičského preukazu, ovládače lic., povolenia vodičov, povolenie jazdu, povolenie na jazdu, povolenie vodiča, vodičský preukaz
Slovenia	vozniško dovoljenje
Spain	carnet conducer, el carnet de conducer, licencia conducer, licencia de manejo, número carnet conducer, número de carnet de conducer, número de permiso conducer, número de permiso de conducer, número licencia conducer, número permiso conducer, permiso conducción, permiso conducer, permiso de conducción
Sweden	ajokortin numero, dlno# ajokortti, drivere lic., förare lic., körkort, körkort nummer, körkortsn ummer, kuljettajat lic.

Country or region	Keywords
UK	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
US	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit

Keywords for national identification numbers

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

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

Country or region	Keywords
Brazil	registro geral, rg
France	assurance sociale, carte nationale d’identit é, cni, code sécurité sociale, French social security number, fssn#, insee, insurance

Country or region	Keywords
	number, national id number, nationalid#, numéro d'assurance, sécurité sociale, sécurité sociale non., sécurité sociale numéro, social, social security, social security number, socialsecuritynumber, ss#, ssn, ssn#
Germany	ausweisnummer, id number, identification number, identity number, insurance number, personal id, personalausweis
Italy	codice fiscal, dati anagrafici, ehic, health card, health insurance card, p. iva, partita i.v.a., personal data, tax code, tessera sanitaria
Spain	dni, dni#, dninúmero#, documento nacional de identidad, identidad único, identidad único#, insurance number, national identific ation number, national identity, nationalid#, nationalidno#, número nacional identidad , personal identification number, personal identity no, unique identity number, uniqueid#

Keywords for passport numbers

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

Country or region	Keywords
Canada	passeport, passeport#, passport, passport#, passportno, passportno#
France	numéro de passeport, passeport, passeport #, passeport #, passeportn °, passeport n °, passeportNon, passeport non

Country or region	Keywords
Germany	ausstellungsdatum, ausstellungsort, geburtsdatum, passport, passports, reisepass, reisepass–nr, reisepassnummer
Italy	italian passport number, numéro passeport , numéro passeport italien, passaporto, passaporto italiana, passaporto numero, passport number, repubblica italiana passaporto
Spain	españa pasaporte, libreta pasaporte, número pasaporte, pasaporte, passport, passport book, passport no, passport number, spain passport
UK	passeport #, passeport n °, passeportNon, passeport non, passeportn °, passport #, passport no, passport number, passport#, passportid
US	passport, travel document

Keywords for taxpayer identification and reference numbers

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

Country or region	Keywords
Brazil	cadastro de pessoa física, cadastro de pessoa fisica, cadastro de pessoas físicas, cadastro de pessoas fisicas, cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj, cpf

Country or region	Keywords
France	numéro d'identification fiscale, tax id, tax identification number, tax number, tin, tin#
Germany	identifikationsnummer, steuer id, steueride ntifikationsnummer, steuernummer, tax id, tax identification number, tax number
Spain	cif, cif número, cifnúmero#, nie, nif, número de contribuyente, número de identidad de extranjero, número de identificación fiscal, número de impuesto corporativo, personal tax number, tax id, tax identification number, tax number, tin, tin#
UK	paye, tax id, tax id no., tax id number, tax identification, tax identification#, tax no., tax number, tax reference, tax#, taxid#, temporary reference number, tin, trn, unique tax reference, unique taxpayer reference, utr
US	individual taxpayer identification number, itin, i.t.i.n.

Data identifier ARNs for personally identifiable information (PII)

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

PII data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/Address
arn:aws:dataprotection::aws:data-identifier/CepCode-BR
arn:aws:dataprotection::aws:data-identifier/Cnpj-BR

PII data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/CpfCode-BR
arn:aws:dataprotection::aws:data-identifier/DateOfBirth
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AT
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AU
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BE
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BG
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CA
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CY
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CZ
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DE
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DK
arn:aws:dataprotection::aws:data-identifier/DriversLicense-EE
arn:aws:dataprotection::aws:data-identifier/DriversLicense-ES
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FI
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FR
arn:aws:dataprotection::aws:data-identifier/DriversLicense-GB
arn:aws:dataprotection::aws:data-identifier/DriversLicense-GR
arn:aws:dataprotection::aws:data-identifier/DriversLicense-HR
arn:aws:dataprotection::aws:data-identifier/DriversLicense-HU
arn:aws:dataprotection::aws:data-identifier/DriversLicense-IE

PII data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/DriversLicense-IT
arn:aws:dataprotection::aws:data-identifier/DriversLicense-LT
arn:aws:dataprotection::aws:data-identifier/DriversLicense-LU
arn:aws:dataprotection::aws:data-identifier/DriversLicense-LV
arn:aws:dataprotection::aws:data-identifier/DriversLicense-MT
arn:aws:dataprotection::aws:data-identifier/DriversLicense-NL
arn:aws:dataprotection::aws:data-identifier/DriversLicense-PL
arn:aws:dataprotection::aws:data-identifier/DriversLicense-PT
arn:aws:dataprotection::aws:data-identifier/DriversLicense-RO
arn:aws:dataprotection::aws:data-identifier/DriversLicense-SE
arn:aws:dataprotection::aws:data-identifier/DriversLicense-SI
arn:aws:dataprotection::aws:data-identifier/DriversLicense-SK
arn:aws:dataprotection::aws:data-identifier/DriversLicense-US
arn:aws:dataprotection::aws:data-identifier/ElectoralRollNumber-GB
arn:aws:dataprotection::aws:data-identifier/EmailAddress
arn:aws:dataprotection::aws:data-identifier/IndividualTaxIdentificationNumber-US
arn:aws:dataprotection::aws:data-identifier/InseeCode-FR
arn:aws:dataprotection::aws:data-identifier/LatLong
arn:aws:dataprotection::aws:data-identifier/Name
arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-DE

PII data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-ES
arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-IT
arn:aws:dataprotection::aws:data-identifier/NieNumber-ES
arn:aws:dataprotection::aws:data-identifier/NifNumber-ES
arn:aws:dataprotection::aws:data-identifier/PassportNumber-CA
arn:aws:dataprotection::aws:data-identifier/PassportNumber-DE
arn:aws:dataprotection::aws:data-identifier/PassportNumber-ES
arn:aws:dataprotection::aws:data-identifier/PassportNumber-FR
arn:aws:dataprotection::aws:data-identifier/PassportNumber-GB
arn:aws:dataprotection::aws:data-identifier/PassportNumber-IT
arn:aws:dataprotection::aws:data-identifier/PassportNumber-US
arn:aws:dataprotection::aws:data-identifier/PermanentResidenceNumber-CA
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-BR
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-DE
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-ES
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-FR
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-GB
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-IT
arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US
arn:aws:dataprotection::aws:data-identifier/PostalCode-CA

PII data identifier ARNs
arn:aws:dataprotection::aws:data-identifier/RgNumber-BR
arn:aws:dataprotection::aws:data-identifier/SocialInsuranceNumber-CA
arn:aws:dataprotection::aws:data-identifier/Ssn-ES
arn:aws:dataprotection::aws:data-identifier/Ssn-US
arn:aws:dataprotection::aws:data-identifier/TaxId-DE
arn:aws:dataprotection::aws:data-identifier/TaxId-ES
arn:aws:dataprotection::aws:data-identifier/TaxId-FR
arn:aws:dataprotection::aws:data-identifier/TaxId-GB
arn:aws:dataprotection::aws:data-identifier/VehicleIdentificationNumber
arn:aws:dataprotection::aws:data-identifier/ZipCode-US

Using custom data identifiers in Amazon SNS

Custom data identifiers (CDIs) let you define your own custom regular expressions that can be used in your data protection policy. Using custom data identifiers, you can target business-specific personally identifiable information (PII) use cases that managed data identifiers can't provide. For example, you can use a custom data identifier to look for company-specific employee IDs. Custom data identifiers can be used in conjunction with managed data identifiers.

What are custom data identifiers?

Custom data identifiers (CDIs) let you define your own custom regular expressions that can be used in your data protection policy. Using custom data identifiers, you can target business-specific personally identifiable information (PII) use cases that managed data identifiers can't provide. For example, you can use a custom data identifier to look for company-specific employee IDs. Custom data identifiers can be used in conjunction with managed data identifiers.

Using custom data identifiers in your data protection policy

The following data protection policy instructs the Amazon SNS topic to detect payloads that carry company-specific employee IDs, then mask these IDs using the hash symbol (#).

1. Create a `Configuration` block within your data protection policy.
2. Enter a `Name` for your custom data identifier. For example, **`EmployeeId`**.
3. Enter a `Regex` for your custom data identifier. For example, **`EID-\d{9}-US`**.
4. Refer to the following custom data identifier in a policy statement.

```
 {
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Configuration": {
    "CustomDataIdentifier": [
      {"Name": "EmployeeId", "Regex": "EID-\d{9}-US"}
    ]
  },
  "Statement": [
    {
      "DataDirection": "Inbound",
      "Principal": ["*"],
      "DataIdentifier": [
         "EmployeeId"
      ],
      "Operation": {
        "Deidentify": {
          "MaskConfig": {
            "MaskWithCharacter": "#"
          }
        }
      }
    }
  ]
}
```

5. (Optional) Continue to add additional **custom data identifiers** to the `Configuration` block as needed. Data protection policies currently support a maximum of 10 custom data identifiers.

Custom data identifier constraints

Amazon SNS custom data identifiers have the following limitations:

- A maximum of 10 custom data identifiers are supported for each data protection policy.
- Custom data identifier names have a maximum length of 128 characters. The following characters are supported:
  - Alphanumeric: (a-zA-Z0-9)
  - Symbols: ('_' | '-')
- RegEx has a maximum length of 200 characters. The following characters are supported:
  - Alphanumeric: (a-zA-Z0-9)
  - Symbols: ('_' | '#' | '=' | '@' |'/' | ';' | ',' | '-' | ' ')
  - RegEx reserved characters: ('^' | '$' | '?' | '[' | ']' | '{' | '}' | '|' | '\\' | '*' | '+' | '.')
- Custom data identifiers cannot share the same name as a managed data identifier.
- Custom data identifiers must be specified in every data protection policy for each Amazon SNS topic.

Amazon SNS message delivery

This topic describes how Amazon SNS handles message delivery across various scenarios. You'll learn about raw message delivery, where Amazon SNS delivers messages in their original, unmodified format to the endpoint. You'll also discover how to send messages from an Amazon SNS topic to an Amazon SQS queue in a different AWS account, providing insights into cross-account messaging.

This topic provides information on the delivery of Amazon SNS messages to an Amazon SQS queue or a Lambda function in different AWS Regions, how cross-region delivery works, and the considerations involved.

Additionally, you'll learn how to monitor and interpret message delivery status, which provides critical information on whether messages were successfully delivered or encountered issues. In cases where message delivery fails, you'll understand the message delivery retry process, including how Amazon SNS automatically attempts to redeliver messages to ensure they reach their intended destinations. This topic also discusses the use of dead-letter queues to capture messages that could not be delivered after multiple attempts, enabling you to analyze and troubleshoot these failures effectively.

Amazon SNS raw message delivery

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

- When you enable raw message delivery for or Amazon SQS endpoints, any Amazon SNS metadata is stripped from the published message and the message is sent as is.
- When you enable raw message delivery for HTTP/S endpoints, the HTTP header `x-amz-sns-rawdelivery` with its value set to `true` is added to the message, indicating that the message has been published without JSON formatting.
- When you enable raw message delivery for HTTP/S endpoints, the message body, client IP, and the required headers are delivered. When you specify message attributes, it won't be sent.
- When you enable raw message delivery for Firehose endpoints, the message body is delivered. When you specify message attributes, it won't be sent.

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

Enabling raw message delivery using the AWS Management Console

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

Message format examples

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

- Raw message delivery is **disabled**

```
{
  "Type": "Notification",
  "MessageId": "dc1e94d9-56c5-5e96-808d-cc7f68faa162",
  "TopicArn": "arn:aws:sns:us-east-2:111122223333:ExampleTopic1",
  "Subject": "TestSubject",
  "Message": "This is a test message.",
  "Timestamp": "2021-02-16T21:41:19.978Z",
  "SignatureVersion": "1",
  "Signature":
 "FMG5tlZhJNHLHUXvZgtZzlk24FzVa7oX0T4P03neeXw8ZEXZx6z35j2FOTuNYShn2h0bKNC/
zLTnMyIxEzmi2X1shOBWsJHkrW2xkR58ABZF+4uWHEE73yDVR4SyYAikP9jstZzDRm
+bcVs8+T0yaLiEGLrIIIL4esi1llhIkgErCuy5btPcWXBdio2fpCRD5x9oR6gmE/
rd5O7lX1c1uvnv4r1Lkk4pqP2/iUfxFZva1xLSRvgyfm6D9hNklVyPfy
+7TalMD0lzmJuOrExtnSIbZew3foxgx8GT+lbZkLd0ZdtdRJlIyPRP44eyq78sU0Eo/
LsDr0Iak4ZDpg8dXg==",
  "SigningCertURL": "https://sns.us-east-2.amazonaws.com/
SimpleNotificationService-010a507c1833636cd94bdb98bd93083a.pem",
  "UnsubscribeURL": "https://sns.us-east-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
east-2:111122223333:ExampleTopic1:e1039402-24e7-40a3-a0d4-797da162b297"
}
```

- Raw message delivery is **enabled**

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

Message attributes and raw message delivery for Amazon SQS subscriptions

Amazon SNS supports the delivery of message attributes, which allow you to provide structured metadata items, such as timestamps, geospatial data, signatures, and identifiers, about the message. For Amazon SQS subscriptions with **Raw Message Delivery** enabled, a maximum of 10 message attributes can be sent. To send more than 10 message attributes, you must disable Raw Message Delivery. However, Amazon SNS discards messages with more than 10 message attributes directed towards Amazon SQS subscriptions with Raw Message Delivery enabled, treating them as client-side errors.

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

This document describes how to publish a notification to an Amazon SNS topic with one or more subscriptions to Amazon SQS queues in another account. You set up the topic and queues the same way you would if they were in the same account (see Fanout Amazon SNS notifications to Amazon SQS queues for asynchronous processing). The major difference is how you handle subscription confirmation, and that depends on how you subscribe the queue to the topic.

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

Note

If the Amazon SQS queue has a high volume of messages, we recommend that the queue owner creates the subscription.

Queue owner creates subscription

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

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

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

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

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

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

6. Choose **Save changes.**

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

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

Before you begin, make sure you have the ARNs for your topic and queue, and that you have given permission to the topic to send messages to the queue.

1. Sign in to the Amazon SQS console.
2. On the navigation panel, choose **Queues**.
3. From the list of queues, choose the **queue** to subscribe to the Amazon SNS topic.
4. Choose **Subscribe to Amazon SNS topic**.
5. From the **Specify an Amazon SNS topic available for this queue menu**, choose the **Amazon SNS topic** for your queue.
6. Choose **Enter Amazon SNS topic ARN** and then enter the topic's **Amazon Resource Name (ARN)**.
7. Choose **Save**.

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

A user who does not own the queue creates a subscription

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

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

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

> **Note**
>
> Before you subscribe an endpoint to the topic, make sure that the queue can receive messages from the topic by setting the `sqs:SendMessage` permission for the queue. For more information, see Step 2: Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.

Step 1: To add an Amazon SQS queue subscription to a topic in another AWS account using the AWS Management Console

Before you begin, make sure you have the ARNs for your topic and queue, and that you have given permission to the topic to send messages to the queue.

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:

    a. For **Topic ARN**, enter the ARN of the topic.

    b. For **Protocol**, choose **Amazon SQS**.

    c. For **Endpoint**, enter the ARN of the queue.

    d. Choose **Create subscription**.

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

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

```
{
   "Sid": "Stmt1234",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "sqs:SendMessage",
   "Resource": "arn:aws:sqs:us-west-2:111111111111:QueueName",
   "Condition": {
      "ArnEquals": {
         "aws:SourceArn": "arn:aws:sns:us-west-2:555555555555:TopicName"
      }
   }
}
```

Step 2: To confirm a subscription using the AWS Management Console

1. Sign in to the Amazon SQS console.
2. Select the queue that has a pending subscription to the topic.
3. Choose **Send and receive messages**, and then choose **Poll for messages**.

   A message with the subscription confirmation is received in the queue.
4. In the **Body** column, do the following:

   a. Choose **More Details**.

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

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

   c. Make a note of the subscription confirmation link. The URL must be passed from the queue owner to the subscription owner. The subscription owner must enter the URL into the Amazon SNS console.

5. Log in as the **subscription owner** to the Amazon SNS console The subscription owner performs the confirmation.
6. Choose the relevant **topic**.
7. Choose the relevant **subscription** in the topic's subscription listings table. It is labeled as "Pending confirmation".
8. Choose **Confirm subscription**.
9. A modal appears prompting the subscription confirmation link. **Paste** the subscription confirmation link.
10. Select the **Confirm subscription** in the modal.

    An XML response is displayed, for example:

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

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

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

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

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

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

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

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

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

The Amazon SNS subscription command must be executed in the region where Amazon SNS is hosted, in the corresponding region. For example, if Amazon SNS is in account "A" in the us-east-1 region, and the Lambda function is in account "B" in the us-east-2 region, the subscription CLI command must be executed in account "A" in the us-east-1 region.

Opt-in Regions

Amazon SNS supports the following opt-in Regions:

Region name	Region
Africa (Cape Town) Region	af-south-1
Asia Pacific (Hong Kong) Region	ap-east-1

Region name	Region
Asia Pacific (Hyderabad) Region	ap-south-2
Asia Pacific (Jakarta) Region	ap-southeast-3
Asia Pacific (Melbourne) Region	ap-southeast-4
Europe (Milan) Region	eu-south-1
Europe (Spain) Region	eu-south-2
Europe (Zurich) Region	eu-central-2
Israel (Tel Aviv) Region	il-central-1
Middle East (Bahrain) Region	me-south-1
Middle East (UAE) Region	me-central-1

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

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

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

Cross-region delivery support to Amazon SQS

Cross-region delivery type	Supported/Not supported
Default-enabled Region to opt-in Region	Supported using `sns.`*`queue-region`*`.amazonaws.com` in the service principal for the queue
Opt-in Region to default-enabled Region	Supported using `sns.`*`topic-region`*`.amazonaws.com` in the service principal for the queue
Opt-in Region to opt-in Region	Supported using `sns.`*`queue-region`*`.amazonaws.com` in the service principal for the queue

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

JSON

```
{
    "Version": "2008-10-17",
    "Id": "__default_policy_ID",
    "Statement": [
        {
            "Sid": "allow_sns_arn:aws:sns:af-
south-1:111111111111:source_topic_name",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.af-south-1.amazonaws.com"
            },
            "Action": "SQS:SendMessage",
            "Resource": "arn:aws:sqs:us-
west-1:111111111111:destination_queue_name",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:af-
south-1:111111111111:source_topic_name"
                }
            }
        }
    ]
}
```

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

Cross-region delivery support to AWS Lambda

Cross-region delivery type	Supported/Not supported	
Default-enabled Region to opt-in Region	Supported using `sns.`*`lambda-function-region`*`.amazonaws.com` in the service principal for the Lambda function	
Opt-in Region to default-enabled Region	Supported using `sns.`*`topic-region`*`.amazonaws.com` in the service principal for the Lambda function	
Opt-in Region to opt-in Region	Supported using `sns.`*`lambda-function-region`*`.amazonaws.com` in the service principal for the Lambda function	

Amazon SNS message delivery status

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

- Amazon Data Firehose

- Amazon Simple Queue Service
- AWS Lambda
- HTTPS
- Platform application endpoint

Delivery status logs are sent to Amazon CloudWatch Logs, providing insights into message delivery operations. These logs help you:

- Determine whether a message was successfully delivered to an endpoint.
- Identify the response from the endpoint to Amazon SNS.
- Measure message dwell time (time between publish timestamp and handoff to the endpoint).

You can configure delivery status logging using the AWS Management Console, AWS SDKs, Query API, or AWS CloudFormation.

Prerequisites for delivery status logging

This topic outlines the necessary IAM permissions for enabling Amazon SNS to write delivery logs to CloudWatch and explains the default log group naming convention. This ensures you have the correct setup and access to monitor and analyze message delivery logs in CloudWatch logs.

Required IAM permissions

The IAM role attached for delivery status logging must include the following permissions to enable Amazon SNS to write to CloudWatch Logs. You can use an existing role with these permissions or create a new role during setup.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Log group naming convention

By default, Amazon SNS creates CloudWatch log groups for delivery status logs using the following naming convention. Log streams within this group correspond to the endpoint protocols (for example, Lambda, Amazon SQS). Ensure you have permissions to view these logs in the CloudWatch Logs console.

```
sns/<region>/<account-id>/<topic-name>
```

Configuring delivery status logging using the AWS Management Console

This topic explains how to enable message delivery status logging for Amazon SNS topics, including configuring logging settings, assigning IAM roles, and verifying that CloudWatch Logs capture delivery logs for monitoring and troubleshooting.

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Topics**.
3. Select the desired **topic** and then choose **Edit**.
4. Expand the **Delivery status logging** section.
5. Choose the **protocol** for which you want to enable logging (for example, HTTP, Lambda, Amazon SQS).
6. Enter the **Success sample rate**, which is the percentage of successful messages for which you want to receive CloudWatch Logs.
7. In the **IAM roles** section, you must configure roles for both **success** and **failure** logging:

   - **Use an existing service role** – Select an existing IAM role that has the required permissions for Amazon SNS to write logs to CloudWatch.

  - **Create a new service role** – Choose **Create new roles** to define the IAM roles for successful and failed deliveries in the IAM console. For permission details, see Prerequisites for delivery status logging.

8. Choose **Save changes**.

   After enabling logging, you can view and parse the CloudWatch Logs containing the message delivery status. For more information about using CloudWatch, see the CloudWatch documentation.

Verifying log setup

1. Sign into the CloudWatch Logs console.
2. Locate the log group named `sns/<region>/<account-id>/<topic-name>`.
3. Ensure log streams exist for the configured endpoint protocol.
4. Send a test message to your topic and confirm that log entries appear, indicating successful or failed deliveries.

Configuring delivery status logging using the AWS SDKs

The AWS SDKs provide APIs in several languages to set topic attributes for message delivery status logging. For example, use the SetTopicAttributes API to configure:

- `LambdaSuccessFeedbackRoleArn` – IAM role for successful message delivery to Lambda endpoints.
- `LambdaSuccessFeedbackSampleRate` – Sampling rate for successful messages to Lambda endpoints.
- `LambdaFailureFeedbackRoleArn` – IAM role for failed message delivery to Lambda endpoints.

Example AWS CLI command

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --attribute-name LambdaSuccessFeedbackRoleArn \
    --attribute-value arn:aws:iam::123456789012:role/MyFeedbackRole
```

Topic attributes

Use the following topic attribute name values for message delivery status:

HTTP

- `HTTPSuccessFeedbackRoleArn` – Successful message delivery status for an Amazon SNS topic that is subscribed to an HTTP endpoint.
- `HTTPSuccessFeedbackSampleRate` – Percentage of successful messages to sample for an Amazon SNS topic that is subscribed to an HTTP endpoint.
- `HTTPFailureFeedbackRoleArn` – Failed message delivery status for an Amazon SNS topic that is subscribed to an HTTP endpoint.

Amazon Data Firehose

- `FirehoseSuccessFeedbackRoleArn` – Successful message delivery status for an Amazon SNS topic that is subscribed to an Amazon Data Firehose endpoint.
- `FirehoseSuccessFeedbackSampleRate` – Percentage of successful messages to sample for an Amazon SNS topic that is subscribed to an Amazon Data Firehose endpoint.
- `FirehoseFailureFeedbackRoleArn` – Failed message delivery status for an Amazon SNS topic that is subscribed to an Amazon Data Firehose endpoint.

AWS Lambda

- `LambdaSuccessFeedbackRoleArn` – Successful message delivery status for an Amazon SNS topic that is subscribed to an Lambda endpoint.
- `LambdaSuccessFeedbackSampleRate` – Percentage of successful messages to sample for an Amazon SNS topic that is subscribed to an Lambda endpoint.
- `LambdaFailureFeedbackRoleArn` – Failed message delivery status for an Amazon SNS topic that is subscribed to an Lambda endpoint.

Platform application endpoints

- `ApplicationSuccessFeedbackRoleArn` – Successful message delivery status for an Amazon SNS topic that is subscribed to an AWS application endpoint.
- `ApplicationSuccessFeedbackSampleRate` – Percentage of successful messages to sample for an Amazon SNS topic that is subscribed to an AWS application endpoint.
- `ApplicationFailureFeedbackRoleArn` – Failed message delivery status for an Amazon SNS topic that is subscribed to an AWS application endpoint.

> **Note**
>
> Additionally, you can configure application attributes to log delivery status directly to push notification services. For more information, see Using Amazon SNS Application Attributes for Message Delivery Status.

Amazon SQS

- `SQSSuccessFeedbackRoleArn` – Successful message delivery status for an Amazon SNS topic that is subscribed to an Amazon SQS endpoint.
- `SQSSuccessFeedbackSampleRate` – Percentage of successful messages to sample for an Amazon SNS topic that is subscribed to an Amazon SQS endpoint.
- `SQSFailureFeedbackRoleArn` – Failed message delivery status for an Amazon SNS topic that is subscribed to an Amazon SQS endpoint.

Logs for platform application endpoints are written to the same CloudWatch Logs group as other endpoints.

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

AWS SDK examples to configure topic attributes

The following code examples show how to use `SetTopicAttributes`.

CLI

AWS CLI

To set an attribute for a topic

The following `set-topic-attributes` example sets the `DisplayName` attribute for the specified topic.

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --attribute-name DisplayName \
    --attribute-value MyTopicDisplayName
```

This command produces no output.

- For API details, see SetTopicAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetTopicAttributes {

    public static void main(String[] args) {
        final String usage = """

                Usage:    <attribute> <topicArn> <value>

                Where:
                   attribute - The attribute action to use. Valid parameters are:
 Policy | DisplayName | DeliveryPolicy .
                   topicArn - The ARN of the topic.\s
                   value - The value for the attribute.
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String attribute = args[0];
        String topicArn = args[1];
        String value = args[2];

        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        setTopAttr(snsClient, attribute, topicArn, value);
        snsClient.close();
    }

    public static void setTopAttr(SnsClient snsClient, String attribute, String
 topicArn, String value) {
        try {
            SetTopicAttributesRequest request =
 SetTopicAttributesRequest.builder()
                    .attributeName(attribute)
                    .attributeValue(value)
                    .topicArn(topicArn)
                    .build();

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

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

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

Kotlin

SDK for Kotlin

Note

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

```
suspend fun setTopAttr(
    attribute: String?,
    topicArnVal: String?,
    value: String?,
) {
    val request =
        SetTopicAttributesRequest {
            attributeName = attribute
            attributeValue = value
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.setTopicAttributes(request)
        println("Topic ${request.topicArn} was updated.")
    }
}
```

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

PHP

SDK for PHP

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

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

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


/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
```

```
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

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

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

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

Ruby

SDK for Ruby

Note

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

```
# Service class to enable an SNS resource with a specified policy
class SnsResourceEnabler
  # Initializes the SnsResourceEnabler with an SNS resource client
  #
  # @param sns_resource [Aws::SNS::Resource] The SNS resource client
  def initialize(sns_resource)
    @sns_resource = sns_resource
    @logger = Logger.new($stdout)
  end

  # Sets a policy on a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param resource_arn [String] The ARN of the resource to include in the policy
  # @param policy_name [String] The name of the policy attribute to set
  def enable_resource(topic_arn, resource_arn, policy_name)
    policy = generate_policy(topic_arn, resource_arn)
    topic = @sns_resource.topic(topic_arn)

    topic.set_attributes({
                           attribute_name: policy_name,
                           attribute_value: policy
                         })
    @logger.info("Policy #{policy_name} set successfully for topic
 #{topic_arn}.")
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Failed to set policy: #{e.message}")
  end

  private

  # Generates a policy string with dynamic resource ARNs
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param resource_arn [String] The ARN of the resource
  # @return [String] The policy as a JSON string
  def generate_policy(topic_arn, resource_arn)
    {
      Version: '2008-10-17',
      Id: '__default_policy_ID',
      Statement: [{
        Sid: '__default_statement_ID',
        Effect: 'Allow',
        Principal: { "AWS": '*' },
        Action: ['SNS:Publish'],
        Resource: topic_arn,
        Condition: {
          ArnEquals: {
            "AWS:SourceArn": resource_arn
          }
        }
      }]
    }.to_json
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'MY_TOPIC_ARN' # Should be replaced with a real topic ARN
  resource_arn = 'MY_RESOURCE_ARN' # Should be replaced with a real resource ARN
  policy_name = 'POLICY_NAME' # Typically, this is "Policy"

  sns_resource = Aws::SNS::Resource.new
  enabler = SnsResourceEnabler.new(sns_resource)

  enabler.enable_resource(topic_arn, resource_arn, policy_name)
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see SetTopicAttributes in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

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

```
    TRY.
        lo_sns->settopicattributes(
            iv_topicarn = iv_topic_arn
            iv_attributename  = iv_attribute_name
            iv_attributevalue = iv_attribute_value ).
        MESSAGE 'Set/updated SNS topic attributes.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see SetTopicAttributes in *AWS SDK for SAP ABAP API reference*.

Configuring delivery status logging using AWS CloudFormation

To configure `DeliveryStatusLogging` using AWS CloudFormation, use a JSON or YAML template to create an AWS CloudFormation stack. For more information, see the `DeliveryStatusLogging` property of the `AWS::SNS::Topic` resource in the AWS CloudFormation User Guide. Below are examples of AWS CloudFormation templates in JSON and YAML to create a new topic or update an existing topic with all `DeliveryStatusLogging` attributes for the Amazon SQS protocol.

Ensure the IAM roles referenced in `SuccessFeedbackRoleArn` and `FailureFeedbackRoleArn` have the required CloudWatch Logs permissions.

JSON

```
"Resources": {
    "MySNSTopic" : {
        "Type" : "AWS::SNS::Topic",
        "Properties" : {
            "TopicName" : "TestTopic",
            "DisplayName" : "TEST",
            "SignatureVersion" : "2",
            "DeliveryStatusLogging" : [{
                "Protocol": "sqs",
                "SuccessFeedbackSampleRate": "45",
                "SuccessFeedbackRoleArn": "arn:aws:iam::123456789012:role/
SNSSuccessFeedback_test1",
                "FailureFeedbackRoleArn": "arn:aws:iam::123456789012:role/
SNSFailureFeedback_test2"
            }]
        }
    }
}
```

YAML

```
Resources:
  MySNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      TopicName:TestTopic
      DisplayName:TEST
      SignatureVersion:2
      DeliveryStatusLogging:
       - Protocol: sqs
         SuccessFeedbackSampleRate: 45
         SuccessFeedbackRoleArn: arn:aws:iam::123456789012:role/
SNSSuccessFeedback_test1
         FailureFeedbackRoleArn: arn:aws:iam::123456789012:role/
SNSFailureFeedback_test2
```

Amazon SNS message delivery retries

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

Delivery protocols and policies

Note

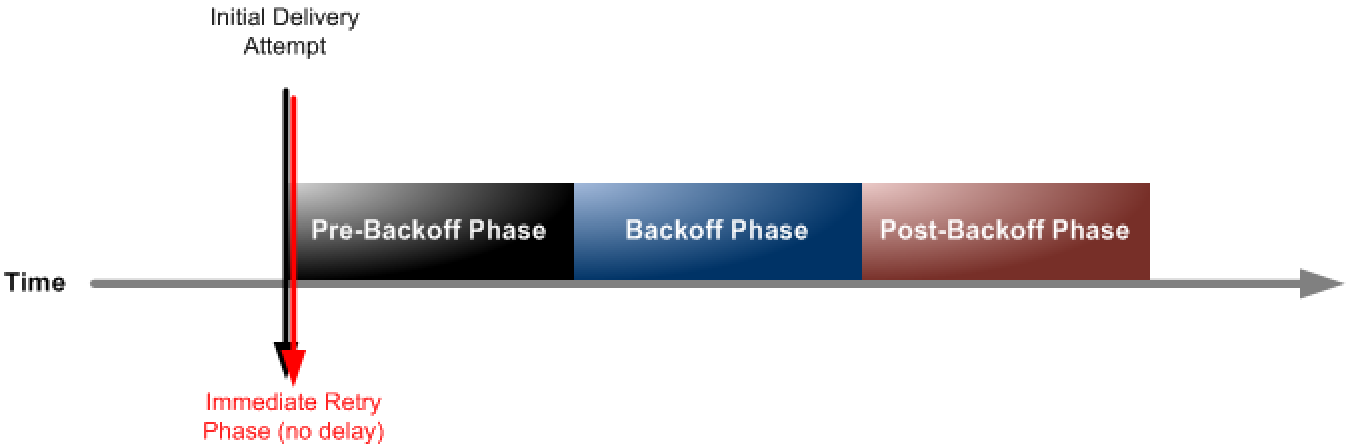
- With the exception of HTTP/S, you can't change Amazon SNS-defined delivery policies. Only HTTP/S supports custom policies. See Creating an HTTP/S delivery policy.
- Amazon SNS applies jittering to delivery retries. For more information, see the Exponential Backoff and Jitter post on the *AWS Architecture Blog*.
- **The total policy retry time for an HTTP/S endpoint cannot be greater than 3,600 seconds. This is a hard limit and cannot be increased.**

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

[1] For throttling errors with the Firehose protocol, Amazon SNS uses the same delivery policy as for customer managed endpoints.

Delivery policy stages

The following diagram shows the phases of a delivery policy.



Each delivery policy is comprised of four phases.

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

Creating an HTTP/S delivery policy

You can define how Amazon SNS retries message delivery to HTTP/S endpoints using a delivery policy with four phases: *no-delay*, *pre-backoff*, *backoff*, and *post-backoff*. This policy allows you to override the default retry settings and customize them to match the capacity of your HTTP server.

You can define your HTTP/S delivery policy as a JSON object at either the **topic** or **subscription** level:

- **Topic-level policy** – Applies to all HTTP/S subscriptions linked to the topic. Use the `CreateTopic` or `SetTopicAttributes` API action to set this policy.

- **Subscription-level policy** – Applies only to a specific subscription. Use the `Subscribe` or `SetSubscriptionAttributes` API action to set this policy.

Alternatively, you can also use the AWS::SNS::Subscription resource in your AWS CloudFormation templates.

You should customize your delivery policy based on your HTTP/S server's capacity:

- **Single server for all subscriptions** – If all HTTP/S subscriptions in a topic use the same server, set the delivery policy as a topic attribute to ensure consistency across all subscriptions.
- **Different servers for subscriptions** – If subscriptions target different servers, create a unique delivery policy for each subscription, tailored to the capacity of the specific server.

You can also set the `Content-Type` header in the request policy to specify the media type of the notification. By default, Amazon SNS sends all the notifications to HTTP/S endpoints with content type set to `text/plain; charset=UTF-8`. However, you can override this default using the `headerContentType` field in the request policy.

The following JSON object defines a delivery policy with retries structured in four phases:

1. **No-delay phase** – Retry 3 times immediately.
2. **Pre-backoff phase** – Retry 2 times with a 1 second interval.
3. **Backoff phase** – Retry 10 times with exponential delays ranging from 1 to 60 seconds.
4. **Post-backoff phase** – Retry 35 times with a fixed 60-second interval.

Amazon SNS makes a total of **50 attempts** to deliver a message before discarding it. To retain messages that can't be delivered after all retries, configure your subscription to move undeliverable messages to a dead-letter queue (DLQ). For more information, see Amazon SNS dead-letter queues.

Amazon SNS considers all 5XX errors and 429 (too many requests sent) errors as retryable. These errors are subject to the delivery policy. All other errors are considered as permanent failures and retries will not be attempted.

> **Note**
>
> This delivery policy uses the `maxReceivesPerSecond` property to throttle delivery traffic to an average of 10 messages per second per subscription. While this mechanism helps prevent your HTTP/S endpoint from being overwhelmed by high traffic, it's designed to maintain an average delivery rate and doesn't enforce a strict cap. Occasional delivery traffic spikes above the specified limit may occur, especially if your publishing rate is significantly higher than the throttling limit.
> When the publishing (inbound) traffic exceeds the delivery (outbound) rate, it can result in a message backlog and higher delivery latency. To avoid such issues, ensure the `maxReceivesPerSecond` value aligns with your HTTP/S server's capacity and workload requirements.

The following delivery policy example overrides the default content type for HTTP/S notification to `application/json`.

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

The delivery policy is composed of a **retry policy**, **throttle policy** and a **request policy**. In total, there are **9 attributes** in a delivery policy.

Policy	Description	Constraint
`minDelayTarget`	The minimum delay for a retry. **Unit:** Seconds	1 to maximum delay **Default:** 20
`maxDelayTarget`	The maximum delay for a retry. **Unit:** Seconds	Minimum delay to 3,600 **Default:** 20
`numRetries`	The total number of retries, including immediate, pre-backoff, backoff, and post-backoff retries.	0 to 100 **Default:** 3
`numNoDelayRetries`	The number of retries to be done immediately, with no delay between them.	0 or greater **Default:** 0
`numMinDelayRetries`	The number of retries in the pre-backoff phase, with the specified minimum delay between them.	0 or greater **Default:** 0
`numMaxDelayRetries`	The number of retries in the post-backoff phase, with the maximum delay between them.	0 or greater **Default:** 0
`backoffFunction`	The model for backoff between retries.	One of four options: • arithmetic • exponential • geometric • linear

Policy	Description	Constraint
		Default: linear
`maxReceivesPerSecond`	The maximum average number of message deliveries per second, per subscription.	1 or greater **Default:** No throttling (no limit on delivery rate)

Policy	Description	Constraint
`headerContentType`	The content type of the notification being sent to HTTP/S endpoints.	If the request policy is not defined, the content type defaults to `text/plain; charset=UTF-8` . When the raw message delivery is disabled for a subscription (default), or when the delivery policy is defined on the topic-level, the supported header content types are `application/json` and `text/plain` . When the raw message delivery is enabled for a subscription, the following content types are supported: • text/css • text/csv • text/html • text/plain • text/xml • application/atom+xml • application/json • application/octet-stream • application/soap+xml • application/x-www-form-urlencoded • application/xhtml+xml • application/xml

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

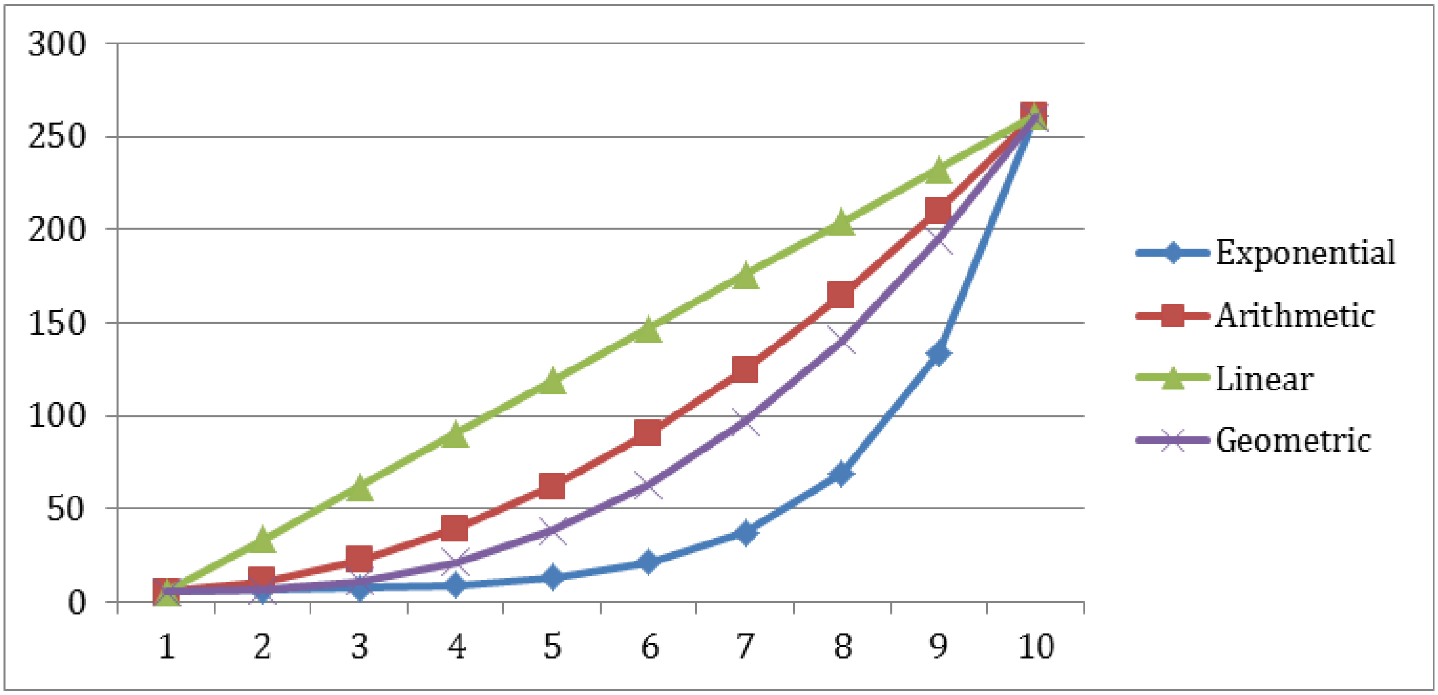
```
numRetries - numNoDelayRetries - numMinDelayRetries - numMaxDelayRetries
```

You can control the frequency of retries during the backoff phase using three parameters:

- **`minDelayTarget`** – Sets the delay for the first retry attempt in the backoff phase.
- **`maxDelayTarget`** – Sets the delay for the final retry attempt in the backoff phase.
- **`backoffFunction`** – Determines the algorithm Amazon SNS uses to calculate the delays for all retry attempts between the first and last retries. You can choose from four available retry-backoff functions.

The following diagram illustrates how different retry backoff functions affect the delays between retries during the backoff phase. The delivery policy used for this example includes the following settings: **10 total retries**, a **minimum delay of 5 seconds**, and a **maximum delay of 260 seconds**.

- The **vertical axis** shows the delay (in seconds) for each retry attempt.
- The **horizontal axis** represents the retry sequence, ranging from the first to the tenth attempt.



Amazon SNS dead-letter queues

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

Note

- The Amazon SNS subscription and Amazon SQS queue must be under the same AWS account and Region.
- For a FIFO topic, you can use an Amazon SQS queue as a dead-letter queue for the Amazon SNS subscription. FIFO topic subscriptions use FIFO queues, and standard topic subscriptions use standard queues.
- To use an encrypted Amazon SQS queue as a dead-letter queue, you must use a custom KMS with a key policy that grants the Amazon SNS service principal access to AWS KMS API actions. For more information, see Securing Amazon SNS data with server-side encryption in this guide and Protecting Amazon SQS Data Using Server-Side Encryption (SSE) and AWS KMS in the *Amazon Simple Queue Service Developer Guide*.

Why do message deliveries fail?

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

Client-side errors

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

Server-side errors

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

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

How do dead-letter queues work?

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

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

How are messages moved into a dead-letter queue?

Your messages are moved into a dead-letter queue using a *redrive policy*. A redrive policy is a JSON object that refers to the ARN of the dead-letter queue. The `deadLetterTargetArn` attribute specifies the ARN. The ARN must point to an Amazon SQS queue in the same AWS account and Region as your Amazon SNS subscription. For more information, see Configuring an Amazon SNS dead-letter queue for a subscription.

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

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

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

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

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

How can I monitor and log dead-letter queues?

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

To be notified of activity in your dead-letter queues, you can use CloudWatch metrics and alarms. Setting up an alarm for the `NumberOfMessagesSent` metric is not suitable because this metric does not capture messages sent to a DLQ as a result of failed processing attempts. Instead, use the `ApproximateNumberOfMessagesVisible` metric, which captures all messages currently available in the DLQ, including those moved due to processing failures.

Example CloudWatch alarm setup

1. Create a CloudWatch alarm for the **`ApproximateNumberOfMessagesVisible`** metric.
2. Set the alarm threshold to **1** (or another appropriate value based on your expectations and DLQ traffic).
3. Specify an Amazon SNS **topic** to be notified when the alarm goes off. This Amazon SNS topic can deliver your alarm notification to any endpoint type (such as an email address, phone number, or mobile pager app).

You can use CloudWatch Logs to investigate the exceptions that cause any Amazon SNS deliveries to fail and for messages to be sent to dead-letter queues. Amazon SNS can log both successful and failed deliveries in CloudWatch. For more information, see Amazon SNS mobile app attributes.

Configuring an Amazon SNS dead-letter queue for a subscription

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

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

> **Note**
>
> For a FIFO topic, you can use an Amazon SQS queue as a dead-letter queue for the Amazon SNS subscription. FIFO topic subscriptions use FIFO queues, and standard topic subscriptions use standard queues.

Prerequisites

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

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

   JSON

   ```
   {
   ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:MyTopic"
        }
      }
    }
  ]
}
```

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

Before your begin this tutorial, make sure you complete the prerequisites.

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

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

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

   a. Choose **Enabled**.

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

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

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

Before you run this example, make sure that you complete the prerequisites.

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

The following code example shows how to use `SetSubscriptionAttributesRedrivePolicy`.

Java

SDK for Java 1.x

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

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

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

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

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

Before your begin this tutorial, make sure you complete the prerequisites.

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

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

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

Before your begin this tutorial, make sure you complete the prerequisites.

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

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

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

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

Amazon SNS message archiving, replay, and analytics

Amazon SNS standard topics support message archiving through . You can fan out notifications to Firehose delivery streams, which allows you to send notifications to storage and analytics destinations that Firehose supports, including Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and more.

Amazon SNS FIFO topics support an in-place, no-code, message archive that lets topic owners store (or *archive*) messages published to a topic for up to 365 days. For topics with an active `ArchivePolicy`, subscribers can then create a `ReplayPolicy` to retrieve (or *replay*) the archived messages back to a subscribed endpoint. To learn more about this feature, see Amazon SNS message archiving and replay for FIFO topics.

Features	Standard Topics	FIFO Topics
Message archiving	Fanout to Firehose delivery streams	Amazon SNS message archiving for FIFO topic owners
Message replay	Replay for standard topics is not a built in feature. Many customers build their own based on their message archive.	Amazon SNS message replay for FIFO topic subscribers

Resource management and optimization in Amazon SNS

This topic provides guidance on how to leverage the full potential of Amazon SNS by ensuring optimal performance, reducing unnecessary costs, and maintaining well-organized resources.

Topics

Amazon SNS topic tagging

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

Tagging for cost allocation

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

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

Resource	Key	Value
Topic 1	Cost Center	43289
	Application	Order processing
Topic 2	Cost Center	43289
	Application	Payment processing
Topic 3	Cost Center	76585

Resource	Key	Value
	Application	Archiving

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

Tagging for access control

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

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

> **Note**
>
> Setting the IAM permission for `Publish` sets permission for both `Publish` and `PublishBatch`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": [
 "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:*:*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "production"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:*:*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "development"
      }
    }
  }]
}
```

Tagging for resource searching and filtering

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

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

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

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

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

        // Query all resources with certain tags from ResourceGroups
        SearchResourcesResult result = awsResourceGroups.searchResources(
            new SearchResourcesRequest().withResourceQuery(
                new ResourceQuery()
                .withType(QueryType.TAG_FILTERS_1_0)
                .withQuery(QUERY)
            ));
        System.out.println("SNS Topics with certain tags are " +
 result.getResourceIdentifiers());
    }
}
```

Configuring Amazon SNS topic tags

This topic explains how to configure tags for an Amazon SNS topic using the AWS Management Console, an AWS SDK, or the AWS CLI.

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

Listing, adding, and removing tags for an Amazon SNS topic using the AWS Management Console

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

   The tags added to the topic are listed.
5. Modify topic tags:

- To add a tag, choose **Add tag** and enter a **Key** and **Value** (optional).
- To remove a tag, choose **Remove tag** next to a key-value pair.

6. Choose **Save changes**.

Adding tags to a topic using an AWS SDK

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

The following code examples show how to use `TagResource`.

CLI

AWS CLI

To add a tag to a topic

The following `tag-resource` example adds a metadata tag to the specified Amazon SNS topic.

```
aws sns tag-resource \
    --resource-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --tags Key=Team,Value=Alpha
```

This command produces no output.

- For API details, see TagResource in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.Tag;
import software.amazon.awssdk.services.sns.model.TagResourceRequest;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AddTags {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn>

                Where:
                   topicArn - The ARN of the topic to which tags are added.

                """;

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

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        addTopicTags(snsClient, topicArn);
        snsClient.close();
    }

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

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

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

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

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

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

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

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

Kotlin

SDK for Kotlin

Note

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

```
suspend fun addTopicTags(topicArn: String) {
    val tag =
        Tag {
            key = "Team"
            value = "Development"
        }

    val tag2 =
        Tag {
            key = "Environment"
            value = "Gamma"
        }

    val tagList = mutableListOf<Tag>()
    tagList.add(tag)
    tagList.add(tag2)

    val request =
        TagResourceRequest {
            resourceArn = topicArn
            tags = tagList
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.tagResource(request)
        println("Tags have been added to $topicArn")
    }
}
```

- For API details, see TagResource in *AWS SDK for Kotlin API reference*.

Managing tags with Amazon SNS API actions

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

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

API actions that support ABAC

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

- `AddPermission`
- `ConfirmSubscription`
- `DeleteTopic`
- `GetDataProtectionPolicy`
- `GetSubscriptionAttributes`
- `GetTopicAttributes`
- `ListSubscriptionsByTopic`
- `ListTagsForResource`
- `Publish`
- `PublishBatch`
- `PutDataProtectionPolicy`
- `RemovePermission`
- `SetSubscriptionAttributes`
- `SetTopicAttributes`
- `Subscribe`
- `TagResource`
- `Unsubscribe`
- `UntagResource`

Amazon SNS event sources and destinations

Amazon SNS connects AWS services and external systems by routing event-driven notifications. Amazon SNS receives events from various AWS services, such as data pipeline updates, Amazon EC2 scaling actions, or security alerts, and publishes these events to Amazon SNS topics. These topics then send notifications to designated destinations.

Amazon SNS supports two main types of destinations: Application-to-Application (A2A) and Application-to-Person (A2P). In A2A messaging, Amazon SNS can send events to Lambda to trigger custom business logic, to Amazon SQS for queuing messages, and to Amazon Data Firehose for streaming data to storage and analytics services. For A2P messaging, Amazon SNS can send notifications via SMS, email, and push notifications to mobile devices, ensuring that users or teams receive timely alerts.

By acting as a central hub, Amazon SNS routes notifications to the right places, helping you automate and manage your AWS infrastructure more effectively. This setup allows for seamless integration between services and reliable communication with users and systems.

Amazon SNS event sources

Amazon SNS integrates with a wide range of AWS services across various categories, allowing these services to publish events to Amazon SNS topics. This integration provides real-time notifications of key events, such as changes in infrastructure, application performance, and cost management.

> **Note**
>
> Amazon SNS introduced FIFO topics in October, 2020. Currently, most AWS services support sending events to standard topics only.

Analytics services

The following table describes how Amazon SNS integrates with AWS analytics services such as Athena, AWS Data Pipeline, and Amazon Redshift to provide real-time notifications for key events, including control limit breaches, pipeline status updates, and data warehouse activities.

You can leverage these integrations to automate responses and maintain effective oversight of your data operations.

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

Application integration services

The following table describes how Amazon SNS integrates with application integration services such as EventBridge and AWS Step Functions, enabling real-time data routing and notifications for business-critical applications.

You can leverage these integrations to receive alerts from EventBridge events and orchestrate workflows using Step Functions, enhancing the automation and responsiveness of your applications.

AWS service	Benefit of using with Amazon SNS
Amazon EventBridge – Delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications , and AWS services and routes that data to targets, including Amazon SNS. EventBridge was formerly called CloudWatch Events.	Receive notifications of EventBridge events. For more information, see Amazon EventBrid ge targets in the *Amazon EventBridge User Guide*.

AWS service	Benefit of using with Amazon SNS
AWS Step Functions – Lets you combine AWS Lambda functions and other AWS services to build business-critical applications.	Receive notification of Step Functions events. For more information, see Call Amazon SNS with Step Functions in the *AWS Step Functions Developer Guide*.

Billing & cost management services

The following table describes how AWS Billing and Cost Management integrates with Amazon SNS to provide notifications for budgets, price changes, and cost anomalies.

You can leverage this integration to set-up Amazon SNS topics to receive real-time alerts about your AWS spending, helping you monitor costs and respond to unexpected charges efficiently.

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

Business applications services

The following table describes how Amazon Chime integrates with Amazon SNS to send notifications for important meeting events, enabling you to stay informed about your communications and scheduling.

You can leverage this integration to utilize Amazon Chime SDK event notifications to enhance your collaboration tools within and outside your organization.

AWS service	Benefit of using with Amazon SNS
Amazon Chime – Lets you meet, chat, and place business calls inside and outside of your organization.	Receive important meeting event notifications. For more information, see Amazon Chime SDK event notifications in the *Amazon Chime Developer Guide*.

Compute services

The following table describes how Amazon SNS integrates with various AWS compute services, enabling you to receive notifications for key events such as Auto Scaling actions, EC2 Image Builder completions, Elastic Beanstalk environment changes, Lambda function outputs, and Lightsail metric thresholds.

You can leverage these integrations to efficiently manage your applications and resources by staying informed about critical updates and actions across AWS services.

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

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

Containers services

The following table describes how Amazon SNS integrates with AWS container services such as Amazon EKS Distro and Amazon ECS, allowing you to track updates and security patches for Amazon EKS clusters and receive notifications for new ECS-optimized AMI releases.

You can leverage these integrations to maintain the security and efficiency of your container deployments by staying informed about important updates and changes.

AWS service	Benefit of using with Amazon SNS
Amazon EKS Distro – Lets you create reliable and secure clusters wherever your applications are deployed.	Track updates and security patches for clusters created with Amazon EKS Distro. For more information, see Introducing Amazon EKS Distro - an open source Kubernetes distribution used by Amazon EKS.

AWS service	Benefit of using with Amazon SNS
Amazon Elastic Container Service (Amazon ECS) – Enables you to run, stop, and manage containers on a cluster.	Receive notifications when a new Amazon ECS-optimized AMI is available. For more information, see Subscribing to Amazon ECS-optimized AMI update notifications in the *Amazon Elastic Container Service Developer Guide*.

Customer engagement services

The following table describes how Amazon SNS enhances customer engagement services by integrating with Amazon Connect, AWS End User Messaging SMS, and Amazon Simple Email Service (SES), enabling you to receive alerts and validations, configure two-way SMS messaging, and monitor email notifications for bounces, complaints, and deliveries.

These integrations help you manage customer communications across multiple channels.

AWS service	Benefit of using with Amazon SNS
Amazon Connect – Lets you set up an omnichannel cloud contact center to engage with your customers.	Receive alerts and validations. For more information, see The power of AWS with Amazon Connect in the *Amazon Connect Administrator Guide*.
AWS End User Messaging SMS – Helps you engage your customers by sending them email, SMS and voice messages, and push notifications.	Configure two-way SMS, which allows you to receive messages from your customers . For more information, see Two-way SMS messaging in the *AWS End User Messaging SMS User Guide*.
Amazon Simple Email Service (Amazon SES) – Provides cost-effective way for you to send and receive email using your own email addresses and domains.	Receive notifications of bounces, complaint s, and deliveries. For more information, see Configuring Amazon SNS notifications for Amazon SES in the *Amazon Simple Email Service Developer Guide*.

Database services

The following table describes how Amazon SNS integrates with AWS database services such as AWS Database Migration Service (DMS), Amazon DynamoDB, Amazon ElastiCache, Amazon Neptune, Amazon Redshift, and Amazon Relational Database Service (RDS) to send notifications about important events such as data migrations, maintenance activities, cache updates, and database changes.

These integrations help you to monitor and manage your database environments more effectively by providing timely alerts on key operational events.

AWS service	Benefit of using with Amazon SNS
AWS Database Migration Service – Migrates data from on-premises databases into the AWS Cloud.	Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see Working with events an d notifications in AWS Database Migration Service in the *AWS Database Migration Service User Guide*.
Amazon DynamoDB – Provides fast and predictable performance with seamless scalability in this fully managed NoSQL database service.	Receive notifications when maintenance events occur. For more information, see Customizing DAX cluster settings in the *Amazon DynamoDB Developer Guide*.
Amazon ElastiCache – Provides a high performance, resizeable, and cost-effective in-memory cache, while removing complexit y associated with deploying and managing a distributed cache environment.	Receive notifications when significant events occur. For more information, see Event notifications and Amazon SNS in the *Amazon ElastiCache (Memcached) User Guide*.
Amazon Neptune – Enables you to build and run applications that work with highly con nected datasets.	Receive notifications when a Neptune event occurs. For more information, see Using Nep tune event notification in the *Neptune User Guide*.

AWS service	Benefit of using with Amazon SNS
Amazon Redshift – Manages all of the work of setting up, operating, and scaling a data warehouse.	Receive notifications of Amazon Redshift events. For more information, see Amazon Redshift event notifications in the *Amazon Redshift Management Guide*.
Amazon Relational Database Service – Makes it easier to set up, operate, and scale a relational database in the AWS Cloud.	Receive notifications of Amazon RDS events. For more information, see Using Amazon RDS event notification in the *Amazon RDS User Guide*.

Developer tools services

The following table describes how Amazon SNS integrates with AWS developer tools services, such as AWS CodeBuild, AWS CodeCommit, AWS CodeDeploy, Amazon CodeGuru, and AWS CodePipeline to provide notifications for critical events such as build status changes, repository updates, deployment progress, performance anomalies, and pipeline actions.

These integrations helps you efficiently monitor and manage your software development workflows by receiving timely alerts on important events.

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

AWS service	Benefit of using with Amazon SNS
AWS CodeDeploy – Automates applicati on deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.	Receive notifications for CodeDeploy deployments or instance events. For more information, see Create a trigger for a CodeDeploy event in the *AWS CodeDeploy User Guide*.
Amazon CodeGuru – Collects runtime performance data from your live applications, and provides recommendations that can help you fine-tune your application performance.	Receive notifications when anomalies occur. For more information, see Working with anomalies and recommendation reports in the *Amazon CodeGuru User Guide*.
AWS CodePipeline – Automates the steps required to release software changes co ntinuously.	Receive notifications about approval actions. For more information, see Manage approval actions in CodePipeline in the *AWS CodePipel ine User Guide*.

Front-end web & mobile services

The following table describes how Amazon SNS integrates with AWS End User Messaging SMS to enhance customer engagement by sending emails, SMS, voice messages, and push notifications, including the ability to configure two-way SMS for receiving customer messages.

This integration allows you to interact more effectively with your customers across various communication channels.

AWS service	Benefit of using with Amazon SNS
AWS End User Messaging SMS – Helps you engage your customers by sending them email, SMS and voice messages, and push notifications.	Configure two-way SMS, which allows you to receive messages from your customers . For more information, see Two-way SMS messaging in the *AWS End User Messaging SMS User Guide*.

Game development services

The following table describes how Amazon SNS integrates with Amazon GameLift Servers to provide notifications for matchmaking and queue events in session-based multiplayer game servers.

This integration helps game developers automate and monitor the deployment, operation, and scaling of their game servers, ensuring a seamless gaming experience.

AWS service	Benefit of using with Amazon SNS
Amazon GameLift Servers – Provides solutions for hosting session-based multiplayer game servers in the cloud, including a fully managed service for deploying, operating, and scaling game servers.	Receive matchmaking and queue event notifications. For more information, see the following pages: • For matchmaking notifications, see Set up FlexMatch event notification in the *Amazon GameLift Servers FlexMatch Developer Guide*. • For queue notifications, see Set up event notification for game session placement in the *Amazon GameLift Servers Developer Guide*.

Internet of Things services

The following table descrives how Amazon SNS integrates with AWS IoT services, such as AWS IoT Core, AWS IoT Device Defender, AWS IoT Events, and AWS IoT Greengrass, to provide notifications for IoT events and alerts.

These integrations allow you to effectively monitor device behavior, receive alerts for abnormal activities, and manage IoT devices with real-time updates and actions.

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

Machine learning services

The following table describes how Amazon SNS integrates with AWS machine learning services, such as Amazon CodeGuru, Amazon DevOps Guru, Amazon Lookout for Metrics, Amazon Rekognition, and Amazon SageMaker AI, to provide notifications for anomalies, operational insights, and data labeling activities.

These integrations allow you to monitor application performance, receive alerts for data irregularities, and streamline the deployment of machine learning models with real-time updates.

AWS service	Benefit of using with Amazon SNS
Amazon CodeGuru – Collects runtime performance data from your live applications,	Receive notifications when anomalies occur. For more information, see Working with

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

Management & governance services

The following table describes how Amazon SNS integrates with AWS management and governance services such as Amazon Q Developer in chat applications, AWS CloudFormation, CloudTrail, CloudWatch, AWS Config, AWS Control Tower, AWS License Manager, AWS Service Catalog, and AWS Systems Manager, providing notifications for key events like infrastructure changes, compliance alerts, and operational insights.

These integrations help you monitor and manage your AWS environments efficiently by delivering timely alerts and updates to relevant teams and systems.

AWS service	Benefit of using with Amazon SNS
Amazon Q Developer in chat applications – Enables DevOps and software development teams to use Amazon Chime and Slack chat rooms to monitor and respond to operational events in the AWS Cloud.	Deliver notifications to chat rooms. For more information, see Setting up Amazon Q Developer in chat applications in the *Amazon Q Developer in chat applications Administrator Guide*.
AWS CloudFormation – Enables you to create and provision AWS infrastructure deployments predictably and repeatedly.	Receive notifications when stacks are created and updated. For more information, see Setting AWS CloudFormation stack options in the *AWS CloudFormation User Guide*.
AWS CloudTrail – Provides event history of your AWS account activity.	Receive notifications when CloudTrail publishes new log files to your Amazon S3 bucket. For more information, see Configuring Amazon SNS notifications for CloudTrail in the *AWS CloudTrail User Guide*.
Amazon CloudWatch – Monitors your AWS resources and the applications you run on AWS in real time.	Receive notifications when alarms change state. For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.
AWS Config – Provides a detailed view of the configuration of AWS resources in your AWS account.	Receive notifications when resources are updated, or when AWS Config evaluates custom or managed rules against your resources. For more information, see Notifications that AWS Config sends to an SNS topic and Example configuration item change notifications in the *AWS Config Developer Guide*.
AWS Control Tower – Enables you to set up and govern a secure, compliant, multi-account AWS environment.	Use alerts to help you prevent drift within your landing zone, and receive compliance notifications. For more information, see Tracking alerts through Amazon Simple

AWS service	Benefit of using with Amazon SNS
	Notification Service in the *AWS Control Tower User Guide*.
AWS License Manager – Helps you manage your software licenses from software vendors centrally across AWS and your on-premises environments.	Receive License Manager notifications and alerts. For more information, see Settings in License Manager in the *License Manager User Guide* and Creating ServiceNow incidents for AWS License Manager notifications on the *AWS Management & Governance Blog*.
AWS Service Catalog – Enables IT administrators to create, manage, and distribute portfolios of approved products to end users, who can then access the products they need in a personalized portal.	Receive notifications about stack events. For more information, see AWS Service Catalog notification constraints in the *Service Catalog Administrator Guide*.
AWS Systems Manager – Lets you view and control your infrastructure on AWS.	Receive notifications about the status of commands. For more information, see Monitoring Systems Manager status changes using Amazon SNS notifications in the *AWS Systems Manager User Guide*.

Media services

The following table describes how Amazon SNS integrates with Amazon Elastic Transcoder to send notifications when media transcoding jobs change status, enabling you to efficiently monitor and manage the conversion of media files stored in Amazon S3 into formats suitable for consumer playback devices.

This integration helps you streamline media processing workflows by providing real-time alerts on job status.

AWS service	Benefit of using with Amazon SNS
Amazon Elastic Transcoder – Lets you convert media files that you stored in Amazon S3	Receive notifications when jobs change status. For more information, see Notifications of

AWS service	Benefit of using with Amazon SNS
into media files in the formats required by consumer playback devices.	job status in the *Amazon Elastic Transcoder Developer Guide*.

Migration & transfer services

The following table describes how Amazon SNS integrates with AWS migration and transfer services, such as AWS Application Discovery Service, AWS Database Migration Service (DMS), and AWS Snowball Edge, to provide notifications for events like server data collection, database migration activities, and data transfer jobs.

These integrations help you to effectively manage and monitor your Cloud migration processes by offering real-time alerts and updates on critical migration tasks.

AWS service	Benefit of using with Amazon SNS
AWS Application Discovery Service – Helps you plan your migration to the AWS Cloud by collecting usage and configuration data about your on-premises servers.	Receive notifications of events through AWS CloudTrail. For more information, see Logging Application Discovery Service API calls with AWS CloudTrail in the *Application Discovery Service User Guide*.
AWS Database Migration Service – Migrates data from on-premises databases into the AWS Cloud.	Receive notifications when AWS DMS events occur; for example, when a replication instance is created or deleted. For more information, see Working with events an d notifications in AWS Database Migration Service in the *AWS Database Migration Service User Guide*.
AWS Snowball Edge – Uses physical storage devices to transfer large amounts of data between Amazon S3 and your onsite data storage location at faster-than-internet speeds.	Receive notifications for Snowball Edge jobs. For more information, see Notifications for Snow Family devices in the *AWS Snowball Edge User Guide*.

Networking & content delivery services

The following table describes how Amazon SNS integrates with AWS networking and content delivery services, such as Amazon API Gateway, Amazon CloudFront, AWS Direct Connect, Elastic Load Balancing, Amazon Route 53, and Amazon VPC, to send notifications for events like API messages, CloudFront metric alarms, connection state changes, load balancer events, health check statuses, and VPC endpoint activities.

These integrations help you to monitor and manage your network and content delivery operations by providing timely alerts and updates.

AWS service	Benefit of using with Amazon SNS
Amazon API Gateway – Enables you to create and deploy your own REST and WebSocket APIs at any scale.	Receive messages posted to an API Gateway endpoint. For more information, see Tutorial: Build an API Gateway REST API with AWS integration in the *API Gateway Developer Guide*.
Amazon CloudFront – Speeds up distribution of your static and dynamic web content, such as .html, .css, .php, image, and media files.	Receive notifications when alarms based on specified CloudFront metrics occur. For more information, see Setting alarms to receiv e notifications in the *Amazon CloudFront Developer Guide*.
AWS Direct Connect – Links your internal network to an AWS Direct Connect location over a standard Ethernet fiber-optic cable.	Receive notifications when alarms that monitor the state of an AWS Direct Connect connection change state. For more informati on, see Creating CloudWatch alarms to monitor AWS Direct Connect connections in the *AWS Direct Connect User Guide*.
Elastic Load Balancing – Automatically distributes your incoming traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in more or more Availability Zones.	Receive notifications of alarms you've created for load balancer events. For more informati on, see Create CloudWatch alarms for your load balancer in the *User Guide for Classic Load Balancers*.

AWS service	Benefit of using with Amazon SNS
Amazon Route 53 – Provides domain registration, DNS routing, and health checking.	Receive notifications when health check status is unhealthy. For more information, see To receive an Amazon SNS notification when a health check status is unhealthy (console) in the *Amazon Route 53 Developer Guide*.
Amazon Virtual Private Cloud (Amazon VPC) – Enables you to launch AWS resources into a virtual network that you've defined.	Receive notifications for specific events that occur on interface endpoints. For more information, see Create and manage a notification for an endpoint service in the *Amazon VPC User Guide*.

Security, identity, & compliance services

The following table describes how Amazon SNS integrates with AWS security, identity, and compliance services, such as AWS Directory Service, Amazon GuardDuty, Amazon Inspector, and AWS Security Hub, to provide notifications for directory status changes, security findings, Inspector events, and security hub announcements.

These integrations help you to maintain robust security practices by offering timely alerts and updates on security and compliance events.

AWS service	Benefit of using with Amazon SNS
AWS Directory Service – Provides multiple ways to use Microsoft Active Directory (AD) with other AWS services.	Receive email or text (SMS) messages when the status of your directory changes. For more information, see Configure directory status notifications in the *AWS Directory Service Administration Guide*.
Amazon GuardDuty – Provides continuous security monitoring to help to identify unexpected and potentially unauthorized or malicious activity in your AWS environment.	Receive notifications about newly released finding types, updates to the existing finding types, and other functionality changes. For more information, see Subscribing to

AWS service	Benefit of using with Amazon SNS
	GuardDuty announcements SNS topic in the *Amazon GuardDuty User Guide*.
Amazon Inspector – Tests the network accessibility of your Amazon EC2 instances and the security state of your applications that run on those instances.	Receive notifications for Amazon Inspector events. For more information, see Setting up an SNS topic for Amazon Inspector notifications in the *Amazon Inspector User Guide*.
AWS Security Hub – Automates AWS security checks and centralizes security alerts.	Receive notifications about AWS Security Hub announcements, including notifications about AWS Security Hub controls or standards that have been added, edited, or retired. For more information, see Subscribing to AWS Security Hub announcements with Amazon SNS.

Serverless services

The following table describes how Amazon SNS integrates with services like Amazon DynamoDB, Amazon EventBridge, and Lambda to send notifications for key events such as maintenance updates, real-time data streams, and Lambda function outputs.

These integrations help you to efficiently monitor and manage your applications by receiving timely alerts on critical operations and automating responses to these events.

AWS service	Benefit of using with Amazon SNS
Amazon DynamoDB – Provides fast and predictable performance with seamless scalability in this fully managed NoSQL database service.	Receive notifications when maintenance events occur. For more information, see Customizing DAX cluster settings in the *Amazon DynamoDB Developer Guide*.
Amazon EventBridge – Delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services and routes that data to	Receive notifications of EventBridge events. For more information, see Amazon EventBridge targets in the *Amazon EventBridge User Guide*.

AWS service	Benefit of using with Amazon SNS
targets, including Amazon SNS. EventBridge was formerly called CloudWatch Events.	
AWS Lambda – Lets you run code without provisioning or managing servers.	Receive function output data by setting an SNS topic as a Lambda dead-letter queue or a Lambda destination. For more informati on, see Asynchronous invocation in the *AWS Lambda Developer Guide*.

Storage services

The following table describes how Amazon SNS integrates with AWS storage services like AWS Backup, Amazon Elastic File System (EFS), Amazon S3 Glacier, Amazon S3, and AWS Snowball Edge to provide notifications for various events such as backup activities, file system alarms, data retrieval jobs, bucket changes, and data transfer operations.

These integrations help you to efficiently monitor and manage your storage solutions by receiving timely alerts on critical storage events.

AWS service	Benefit of using with Amazon SNS
AWS Backup – Helps you centralize and automate the backup of data across AWS services in the Cloud and on premises	Receive notifications of AWS Backup events. For more information, see Using Amazon SNS to track AWS Backup events in the *AWS Backup Developer Guide*.
Amazon Elastic File System – Provides file storage for your Amazon EC2 instances.	Receive notifications of alarms you've created for Amazon EFS events. For more informati on, see Automated monitoring tools in the *Amazon Elastic File System User Guide*.
Amazon S3 Glacier – Provides storage for infrequently used data.	Set a notification configuration on a vault so that when a job completes, a message is sent to an SNS topic. For more information, see Configuring vault notifications in Amazon S3

AWS service	Benefit of using with Amazon SNS
	Glacier in the *Amazon S3 Glacier Developer Guide*.
Amazon Simple Storage Service (Amazon S3) – Provides object storage.	Receive notifications when changes occur to an Amazon S3 bucket or in the rare instance when objects don't replicate to their destination Region. For more information, see Walkthrough: Configure a bucket for notifications (SNS topic or SQS queue) and Monitoring progress with replication metrics and Amazon S3 event notifications in the *Amazon Simple Storage Service User Guide*.
AWS Snowball Edge – Uses physical storage devices to transfer large amounts of data between Amazon S3 and your onsite data storage location at faster-than-internet speeds.	Receive notifications for Snowball Edge jobs. For more information, see Notifications for Snow Family devices in the *AWS Snowball Edge User Guide*.

Additional event sources

The following table describes how Amazon SNS can be used to receive timely notifications about AWS daily feature updates and changes to AWS IP address ranges, ensuring that you are informed about the latest AWS service releases, instance types, VPC endpoints, and public IP address changes.

These integrations help you stay up-to-date with AWS infrastructure changes and manage your resources effectively.

Source	Benefit of using with Amazon SNS
AWS Daily Feature Updates	Receive timely detailed information about releases and updates to AWS via an Amazon SNS topic. These releases include AWS Regions, AWS services, Amazon VPC endpoints

Source	Benefit of using with Amazon SNS
	, AWS services integrated with AWS Service Quotas, Amazon EC2 instance types, Amazon SageMaker AI instance types, Amazon Nimble Studio instance types, Amazon RDS database engine versions, and Amazon MSK Apache Kafka versions. For more information, see Subscribe to AWS Daily Feature Updates via Amazon SNS in the *AWS News Blog*.
AWS IP address ranges	Receive notifications of changes to AWS IP ranges via an Amazon SNS topic. For more information, see AWS IP address ranges notifications in the *Amazon Web Services General Reference*, and Subscribe to AWS Public IP Address Changes via Amazon SNS in the *AWS News Blog*.

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

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

Amazon SNS event destinations

This topic lists all event destinations, organized by application-to-application (A2A) messaging and application-to-person (A2P) notifications.

Note

Amazon SNS introduced FIFO topics in October, 2020. Currently, most AWS services support receiving events from SNS standard topics only. Amazon SQS supports receiving events from both SNS standard and FIFO topics.

A2A destinations

The following table describes how Amazon SNS can deliver events to various application-to-application (A2A) destinations such as , Lambda, Amazon SQS, AWS Event Fork Pipelines, and HTTP/S endpoints.

These integrations allow you to archive and analyze data, trigger custom business logic, facilitate application integration, and route events to external webhooks, enhancing the efficiency and flexibility of event-driven architectures.

Event destination	Benefit of using with Amazon SNS
https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html	Deliver events to delivery streams for archiving and analysis purposes. Through delivery streams, you can deliver events to AWS destinations like Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and Amazon OpenSearch Service (OpenSearch Service), or to third-party destinations such as Datadog, New Relic, MongoDB, and Splunk. For more information, see Fanout to Firehose delivery streams.
AWS Lambda	Deliver events to functions for triggering the execution of custom business logic. For more information, see Fanout Amazon SNS notificat ions to Lambda functions for automated processing.
Amazon SQS	Deliver events to queues for application integration purposes. For more informati on, see Fanout Amazon SNS notifications to Amazon SQS queues for asynchronous processing.
AWS Event Fork Pipelines	Deliver events to event backup and storage, event search and analytics, or event replay pipelines. For more information, see Fanout

Event destination	Benefit of using with Amazon SNS
	Amazon SNS events to AWS Event Fork Pipelines.
HTTP/S	Deliver events to external webhooks. For more information, see Fanout Amazon SNS notifications to HTTPS endpoints.

A2P destinations

The following table describes how Amazon SNS delivers application-to-person (A2P) notifications to various destinations, including mobile phones via SMS and native push notifications, email inboxes, Amazon Chime chat rooms, Slack channels, and operational insights to on-call teams via PagerDuty.

These integrations enhance communication and operational efficiency by enabling real-time alerts and updates across multiple platforms and communication channels.

Event destination	Benefit of using with Amazon SNS
SMS	Deliver events to mobile phones as text messages. For more information, see Mobile text messaging with Amazon SNS.
Email	Deliver events to inboxes as email messages. For more information, see Amazon SNS email subscription setup and management.
Platform endpoint	Deliver events to mobile phones as native push notifications. For more information, see Sending mobile push notifications with Amazon SNS.
Amazon Q Developer in chat applications	Deliver events to Amazon Chime chat rooms or Slack channels. For more information, see the following pages in the *Amazon Q*

Event destination	Benefit of using with Amazon SNS
	Developer in chat applications Administrator Guide: - Setting up Amazon Q Developer in chat applications with Amazon Chime - Setting up Amazon Q Developer in chat applications with Slack - Using Amazon Q Developer in chat applications with other AWS services
PagerDuty	Deliver operational insights to on-call teams. For more information, see Deliver ML-powered operational insights to your on-call teams via PagerDuty with Amazon DevOps Guru on the *AWS Management & Governance Blog*.

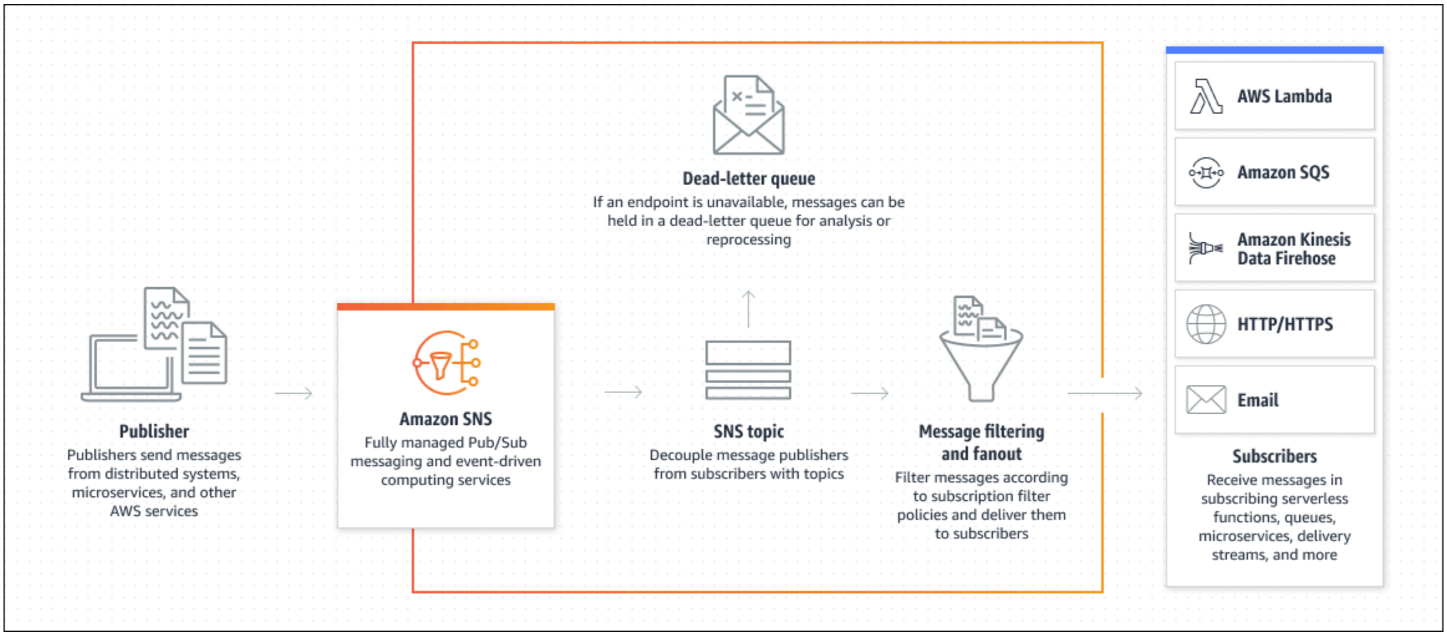
Note

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

- **Native AWS events** – You can use Amazon Q Developer in chat applications to send native AWS events, through Amazon SNS topics, to Amazon Chime and Slack. The supported set of native AWS events includes events from AWS Billing and Cost Management, AWS Health, AWS CloudFormation, Amazon CloudWatch, and more. For more information, see Using Amazon Q Developer in chat applications with other services in the *Amazon Q Developer in chat applications Administrator Guide*.
- **Custom events** – You can also send your custom events, through Amazon SNS topics, to Amazon Chime, Slack, and Microsoft Teams. To do this, you publish custom events to an SNS topic, which delivers the events to a subscribed Lambda function. The Lambda function then uses the chat app's webhook to deliver the events to recipients. For more information, see How do I use webhooks to publish Amazon SNS messages to Amazon Chime, Slack, or Microsoft Teams?

Using Amazon SNS for application-to-application messaging

Amazon SNS simplifies application-to-application (A2A) messaging by separating publishers from subscribers, which supports microservices, distributed systems, and serverless applications. Messages are sent to Amazon SNS topics, where they can be filtered and delivered to subscribers like Lambda, Amazon SQS, or HTTP endpoints. If delivery fails, the messages are stored in a dead-letter queue for further analysis or reprocessing.


Dead-letter queue
If an endpoint is unavailable, messages can be held in a dead-letter queue for analysis or reprocessing
Publisher
Publishers send messages from distributed systems, microservices, and other AWS services
Amazon SNS
Fully managed Pub/Sub messaging and event-driven computing services
SNS topic
Decouple message publishers from subscribers with topics
Message filtering and fanout
Filter messages according to subscription filter policies and deliver them to subscribers
AWS Lambda
Amazon SQS
Amazon Kinesis Data Firehose
HTTP/HTTPS
Email
Subscribers
Receive messages in subscribing serverless functions, queues, microservices, delivery streams, and more

Fanout to Firehose delivery streams

You can subscribe delivery streams to Amazon SNS topics, allowing you to send notifications to additional storage and analytics endpoints. Messages published to an Amazon SNS topic are sent to the subscribed Firehose delivery stream, and delivered to the destination as configured in Firehose. A subscription owner can subscribe up to five Firehose delivery streams to an Amazon SNS topic. Each Firehose delivery stream has a default quota for requests and throughput per second. This limit could result in more messages published (inbound traffic) than delivered (outbound traffic). When there's more inbound than outbound traffic, your subscription can accumulate a large message backlog, causing high message delivery latency. You can request an increase in quota based on the publish rate to avoid adverse impact on your workload.

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

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

Amazon SNS also supports message delivery status logging for notifications sent to Firehose endpoints. For more information, see Amazon SNS message delivery status.

Prerequisites for subscribing Firehose delivery streams to Amazon SNS topics

To subscribe an delivery stream to an SNS topic, your AWS account must have:

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

  The following example policy shows the recommended permissions:

  JSON

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

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

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

JSON

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

For more information on creating roles, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

After you've completed these requirements, you can subscribe the delivery stream to the SNS topic.

Subscribing a Firehose delivery stream to an Amazon SNS topic

To deliver Amazon SNS notifications to delivery streams, first make sure that you've addressed all the prerequisites. For a list of supported endpoints, see endpoints and quotas in the *Amazon Web Services General Reference*.

To subscribe a Firehose delivery stream to a topic

1. Sign in to the Amazon SNS console.
2. In the navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:

   a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a standard topic.

   b. For **Protocol**, choose **Firehose**.

   c. For **Endpoint**, choose the ARN of a Firehose delivery stream that can receive notifications from Amazon SNS.

   d. For **Subscription role ARN**, specify the ARN of the AWS Identity and Access Management (IAM) role that you created for writing to Firehose delivery streams. For more information, see Prerequisites for subscribing Firehose delivery streams to Amazon SNS topics.

   e. (Optional) To remove any Amazon SNS metadata from published messages, choose **Enable raw message delivery**. For more information, see Amazon SNS raw message delivery.
5. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see Amazon SNS subscription filter policies.
6. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see Amazon SNS dead-letter queues.
7. Choose **Create subscription**.

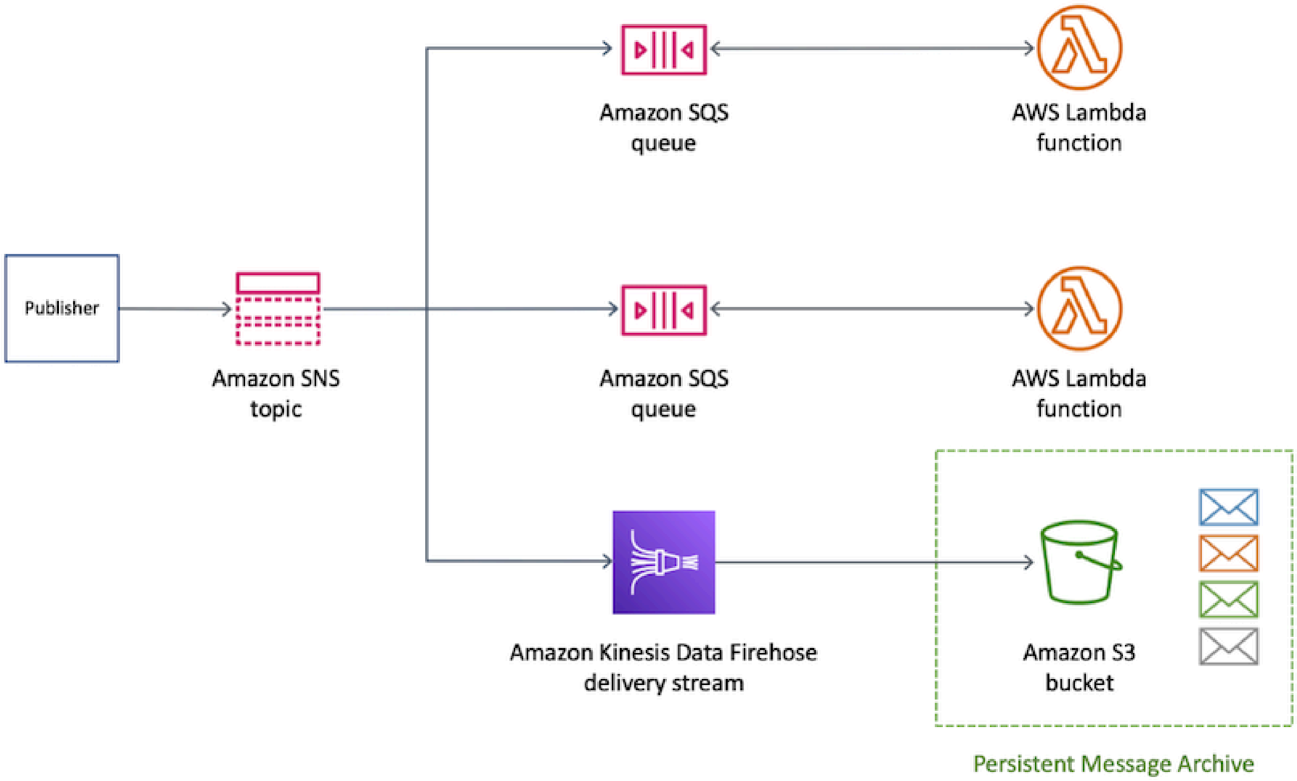
The console creates the subscription and opens the subscription's **Details** page.

Managing Amazon SNS messages across multiple delivery stream destinations

delivery streams allow you to manage Amazon SNS messages across multiple destinations, enabling integration with Amazon S3, Amazon OpenSearch Service, Amazon Redshift, and HTTP endpoints for storage, indexing, and analysis. By properly configuring message formatting and delivery, you can store Amazon SNS notifications in Amazon S3 for later processing, analyze structured message data using Amazon Athena, index messages in OpenSearch for real-time search and visualization, and structure archives in Amazon Redshift for advanced querying.

Storing and analyzing Amazon SNS messages in Amazon S3 destinations

This topic explains how delivery streams publish data to Amazon Simple Storage Service (Amazon S3).



Topics

- Formatting Amazon SNS notifications for storage in Amazon S3 destinations
- Analyzing Amazon SNS messages stored in Amazon S3 using Athena

Formatting Amazon SNS notifications for storage in Amazon S3 destinations

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

Note

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

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

For more information about raw delivery, see Amazon SNS raw message delivery.

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

The following example shows three SNS messages sent through an delivery stream to the same Amazon S3 bucket. Buffering is applied, and line breaks separate each message.

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

Analyzing Amazon SNS messages stored in Amazon S3 using Athena

This page explains how to analyze Amazon SNS messages that are sent through delivery streams to Amazon Simple Storage Service (Amazon S3) destinations.

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

1. Configure your Amazon S3 resources. For instructions, see Creating a bucket in the *Amazon Simple Storage Service User Guide* and Working with Amazon S3 Buckets in the *Amazon Simple Storage Service User Guide*.
2. Configure your delivery stream. For instructions, see Choose Amazon S3 for Your Destination in the *Amazon Data Firehose Developer Guide*.
3. Use Amazon Athena to query the Amazon S3 objects using standard SQL. For more information, see Getting Started in the *Amazon Athena User Guide*.

Example query

For this example query, assume the following:

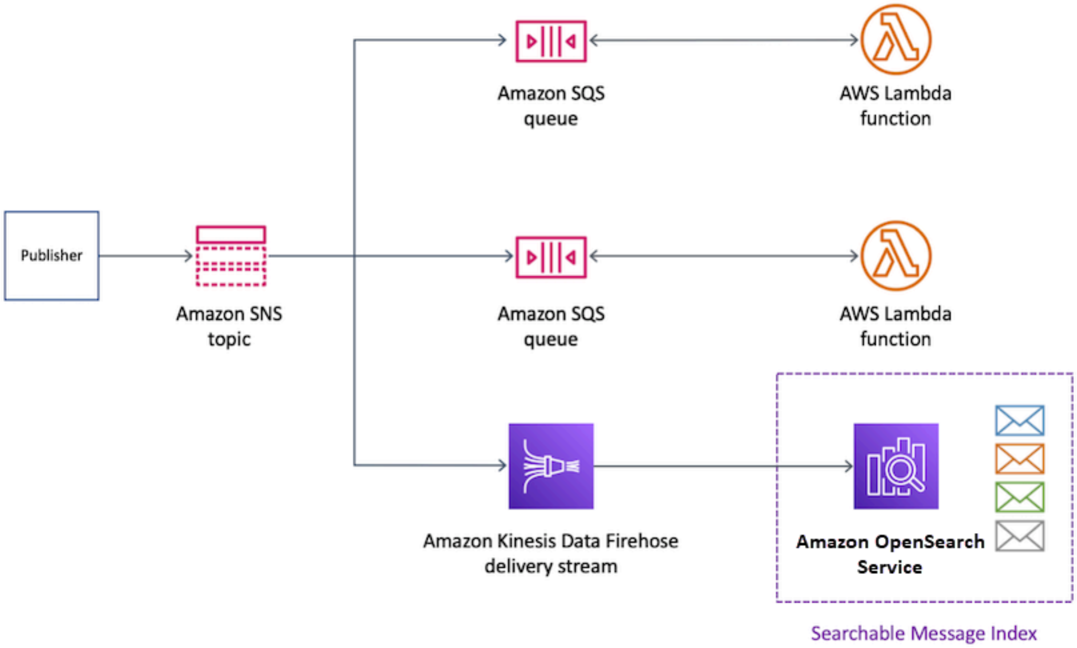
- Messages are stored in the `notifications` table in the `default` schema.
- The `notifications` table includes a `timestamp` column with a type of `string`.

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

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

Integrating Amazon SNS messages with Amazon OpenSearch Service destinations

This section explains how delivery streams publish data to Amazon OpenSearch Service (OpenSearch Service).



Topics

- [Storing and formatting Amazon SNS Notifications in OpenSearch Service indices]
- [Analyzing Amazon SNS messages for OpenSearch Service destinations]

Storing and formatting Amazon SNS Notifications in OpenSearch Service indices

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

> **Note**
>
> In this example, raw message delivery is disabled for the published message. When raw message delivery is disabled, Amazon SNS adds JSON metadata to the message, including these properties:
>
> - `Type`
> - `MessageId`
> - `TopicArn`
> - `Subject`
> - `Timestamp`
> - `UnsubscribeURL`
> - `MessageAttributes`
>
> For more information about raw delivery, see Amazon SNS raw message delivery.

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

Analyzing Amazon SNS messages for OpenSearch Service destinations

This topic explains how to analyze Amazon SNS messages sent through delivery streams to Amazon OpenSearch Service (OpenSearch Service) destinations.

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

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

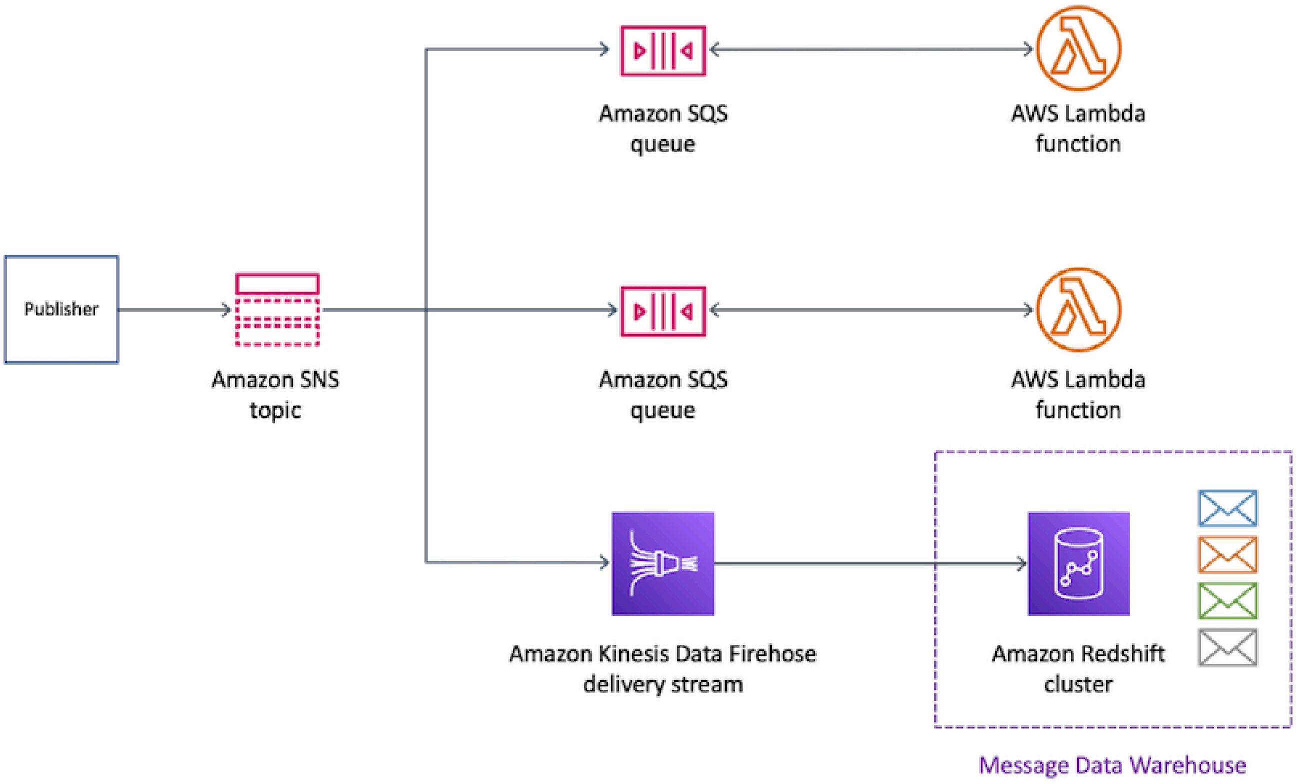
Example query

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

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

Configuring Amazon SNS message delivery and analysis in Amazon Redshift destinations

This topic explains how to fan out Amazon SNS notifications to an delivery stream, which then publishes data to Amazon Redshift. With this setup, you can connect to the Amazon Redshift database and use a SQL query tool to retrieve Amazon SNS messages that match specific criteria.



Topics

- Structuring Amazon SNS message archives in Amazon Redshift tables
- Analyzing Amazon SNS messages stored in Amazon Redshift destinations

Structuring Amazon SNS message archives in Amazon Redshift tables

For Amazon Redshift endpoints, Amazon SNS messages are archived as rows in a table. Here's an example of how the data is stored:

> **Note**
>
> In this example, raw message delivery is disabled for the published message. When raw message delivery is disabled, Amazon SNS adds JSON metadata to the message, including these properties:
>
> - `Type`
> - `MessageId`
> - `TopicArn`

- `Subject`
- `Message`
- `Timestamp`
- `UnsubscribeURL`
- `MessageAttributes`

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

type	messageid	topicarn	subject	message	timestamp	unsubscribeurl	messageattributes
Notification	ea544832-a0d8-581d-9275-108243c46103	arn:aws:sns:us-east-1:111111111111:my-topic	Sample subject	Sample message	2020-12-02T00:33:32.272Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-	{\"my_attribute\":{\"Type\":\"String\",\"Value\":\"my_value\"}}

type	messageid	topicarn	subject	message	timestamp	unsubscribeurl	messageattributes
						ca77a247813b	
Notification	ab124832-a0d8-581d-9275-108243c46114	arn:aws:sns:us-east-1:111111111111:my-topic	Sample subject 2	Sample message 2	2020-12-03T00:18:11.129Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b	{\"my_attribute2\":{\"Type\":\"String\",\"Value\":\"my_value\"}}

type	messageid	topicarn	subject	message	timestamp	unsubscribeurl	messageattributes
Notification	ce644832-a0d8-581d-9275-108243c46125	arn:aws:sns:us-east-1:111111111111:my-topic	Sample subject 3	Sample message 3	2020-12-09T00:08:44.405Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b	{\"my_attribute3\":{\"Type\":\"String\",\"Value\":\"my_value\"}}

For more information about fanning out notifications to Amazon Redshift endpoints, see Configuring Amazon SNS message delivery and analysis in Amazon Redshift destinations.

Analyzing Amazon SNS messages stored in Amazon Redshift destinations

This topic describes how to analyze Amazon SNS messages that are sent through delivery streams to Amazon Redshift destinations.

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

1. Configure your Amazon Redshift resources. For instructions, see Getting started with Amazon Redshift in the *Amazon Redshift Getting Started Guide*.

2. Configure your delivery stream. For instructions, see Choose Amazon Redshift for Your Destination in the *Amazon Data Firehose Developer Guide*.
3. Run a query. For more information, see Querying a database using the query editor in the *Amazon Redshift Management Guide*.

Example query

For this example query, assume the following:

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

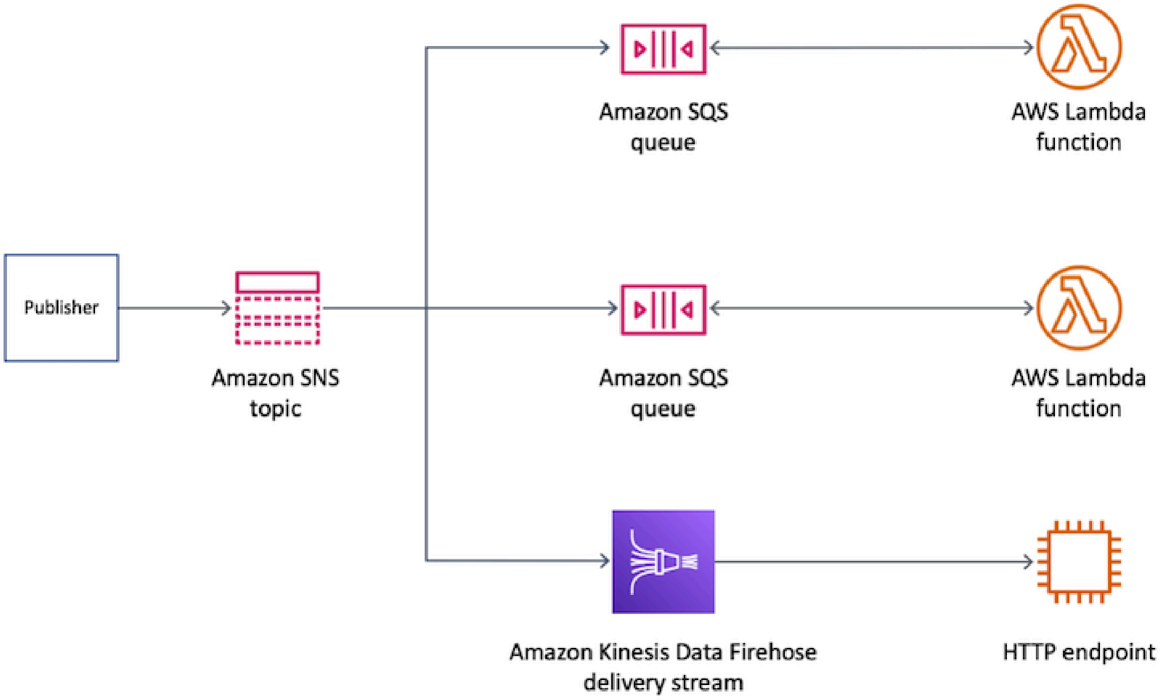
> **Note**
>
> To transform the JSON metadata for the Amazon Redshift endpoint, you can use the SQL `COPY` command. For more information, see Copy from JSON examples and Load from JSON data using the 'auto ignorecase' option in the *Amazon Redshift Database Developer Guide*.

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

```
SELECT *
FROM public.notifications
WHERE timestamp > '2020-12-01T09:00:00.000Z' AND timestamp <
 '2020-12-02T09:00:00.000Z';
```

Configuring Amazon SNS message delivery to HTTP destinations using

This topic explains how delivery streams publish data to HTTP endpoints.



Topics

- Amazon SNS notification format for delivery to HTTP destinations

Amazon SNS notification format for delivery to HTTP destinations

Here's an example of an HTTP POST request body from Amazon SNS, sent through an delivery stream to an HTTP endpoint. The Amazon SNS notification is encoded as a base64 payload within the records property.

> **Note**
>
> In this example, raw message delivery is disabled for the published message. For more information about raw delivery, see Amazon SNS raw message delivery.

```
"body": {
    "requestId": "ebc9e8b2-fce3-4aef-a8f1-71698bf8175f",
    "timestamp": 1606255960435,
    "records": [
      {
        "data":
 "eyJUeXBlIjoiTm90aWZpY2F0aW9uIiwiTWVzc2FnZUlkIjoiMjFkMmUzOGQtMmNhYi01ZjYxLTliYTItYmJiYWFhYzg0M
      }
    ]
  }
```

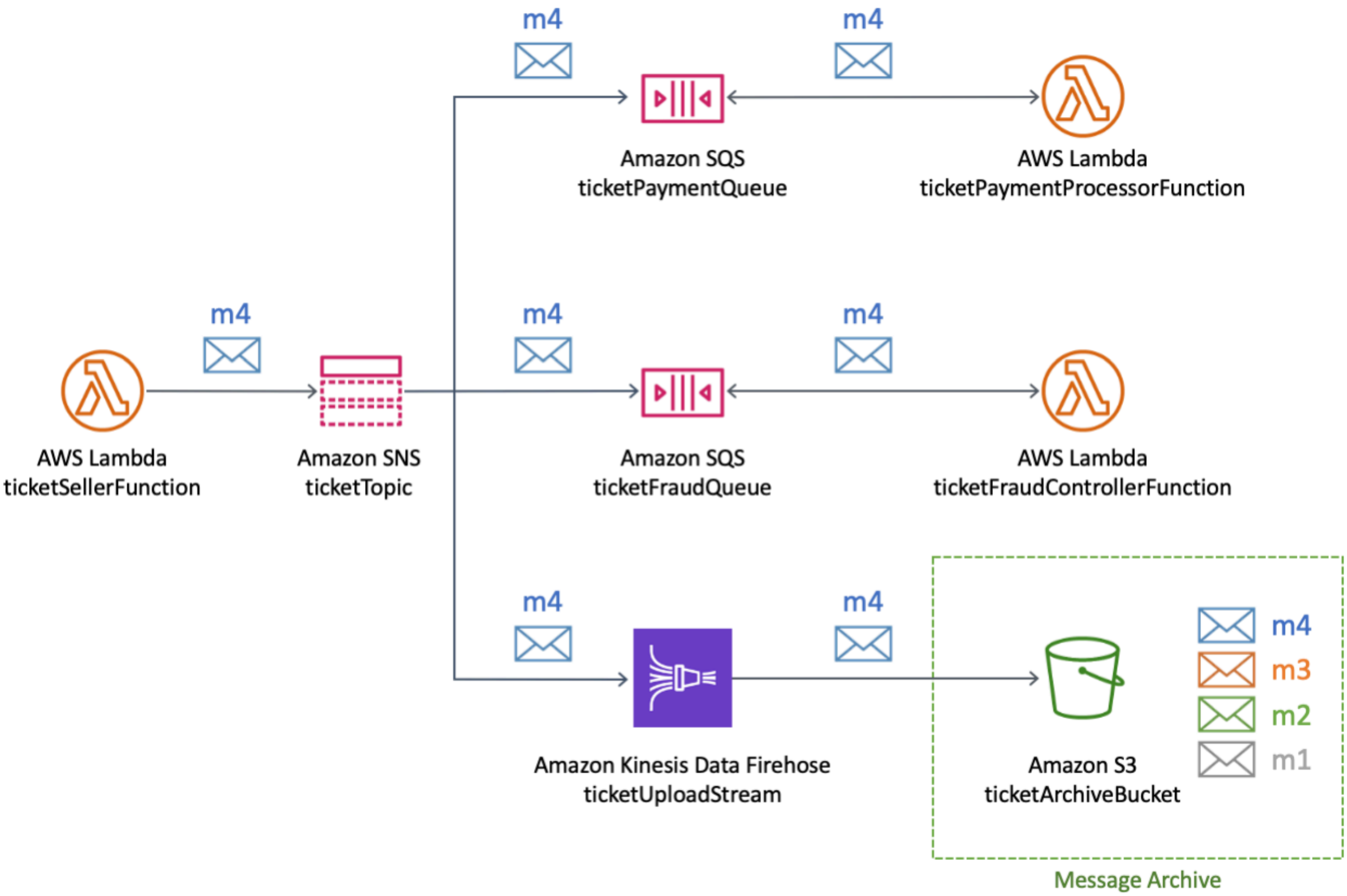
Amazon SNS message archiving and analytics: An example use case for airline ticketing platforms

This topic provides a tutorial for a common use case of archiving and analyzing Amazon SNS messages.

The setting of this use case is an airline ticketing platform that operates in a regulated environment.

1. The platform is subject to a compliance framework that requires the company to archive all ticket sales for at least five years.
2. To meet the compliance goal on data retention, the company subscribes an delivery stream to an existing Amazon SNS topic.
3. The destination for the delivery stream is an Amazon Simple Storage Service (Amazon S3) bucket. With this configuration, all events published to the SNS topic are archived in the Amazon S3 bucket.

The following diagram shows the architecture of this configuration:



To run analytics and gain insights on ticket sales, the company runs SQL queries using Amazon Athena. For example, the company can query to learn about the most popular destinations and the most frequent flyers.

To create the AWS resources for this use case, you can use the AWS Management Console or an AWS CloudFormation template.

Topics

Setting-up initial AWS resources for Amazon SNS message archiving and analytics

This topic describes how to create the resources needed for the message archiving and analytics example use case:

- An Amazon Simple Storage Service (Amazon S3) bucket
- Two Amazon Simple Queue Service (Amazon SQS) queues
- An Amazon SNS topic
- Two Amazon SQS subscriptions to the Amazon SNS topic

To create the initial resources

1. Create the Amazon S3 bucket:

   a. Open the Amazon S3 console.

   b. Choose **Create bucket**.

   c. For **Bucket name**, enter a globally unique name. Keep the other fields as the defaults.

   d. Choose **Create bucket**.

   For more information about Amazon S3 buckets, see Creating a bucket in the *Amazon Simple Storage Service User Guide* and Working with Amazon S3 Buckets in the *Amazon Simple Storage Service User Guide*.

2. Create the two Amazon SQS queues:

   a. Open the Amazon SQS console.

   b. Choose **Create queue**.

   c. For **Type**, choose **Standard**.

   d. For **Name**, enter **`ticketPaymentQueue`**.

   e. Under **Access policy**, for **Choose method**, choose **Advanced**.

   f. In the JSON policy box, paste the following policy:

   JSON

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-
east-1:123456789012:ticketTopic"
        }
      }
    }
  ]
}
```

In this access policy, replace the AWS account number (*123456789012*) with your own, and change the AWS Region (*us-east-1*) accordingly.

g. Choose **Create queue**.

h. Repeat these steps to create a second SQS queue named **ticketFraudQueue**.

For more information on creating SQS queues, see Creating an Amazon SQS queue (console) in the *Amazon Simple Queue Service Developer Guide*.

3. Create the SNS topic:

   a. Open the Topics page of the Amazon SNS console.

   b. Choose **Create topic**.

   c. Under **Details**, for **Type**, choose **Standard**.

   d. For **Name**, enter **ticketTopic**.

   e. Choose **Create topic**.

   For more information on creating SNS topics, see Creating an Amazon SNS topic.

4. Subscribe both SQS queues to the SNS topic:

   a. In the Amazon SNS console, on the **ticketTopic** topic's details page, choose **Create subscription**.

   b. Under **Details**, for **Protocol**, choose **Amazon SQS**.

   c. For **Endpoint**, choose the Amazon Resource Name (ARN) of the **ticketPaymentQueue** queue.

   d. Choose **Create subscription**.

   e. Repeat these steps to create a second subscription using the ARN of the **ticketFraudQueue** queue.

      For more information on subscribing to SNS topics, see Creating a subscription to an Amazon SNS topic. You can also subscribe SQS queues to SNS topics from the Amazon SQS console. For more information, see Subscribing an Amazon SQS queue to an Amazon SNS topic (console) in the *Amazon Simple Queue Service Developer Guide*.

You've created the initial resources for this example use case. To continue, see Setting-up a Amazon Data Firehose delivery stream for Amazon SNS message archiving.

Setting-up a Amazon Data Firehose delivery stream for Amazon SNS message archiving

This topic explains how to create the Amazon Data Firehose delivery stream for the message archiving and analytics example use case.

To create the Amazon Data Firehose delivery stream

1. Open the Amazon Data Firehose services console.
2. Choose **Firehose** and then choose **Create delivery stream**.
3. On the **New delivery stream** page, for **Delivery stream name**, enter **`ticketUploadStream`**, and then choose **Next**.
4. On the **Process records** page, choose **Next**.
5. On the **Choose a destination** page, do the following:

   a. For **Destination**, choose **Amazon S3**.

   b. Under **S3 destination**, for **S3 bucket**, choose the S3 bucket that you created initially.

   c. Choose **Next**.

6. On the **Configure settings** page, for **S3 buffer conditions**, do the following:

   - For **Buffer size**, enter **1**.
   - For **Buffer interval**, enter **`60`**.

   Using these values for the Amazon S3 buffer lets you quickly test the configuration. The first condition that is satisfied triggers data delivery to the S3 bucket.
7. On the **Configure settings** page, for **Permissions**, choose to create an AWS Identity and Access Management (IAM) role with the required permissions assigned automatically. Then choose **Next**.
8. On the **Review** page, choose **Create delivery stream**.
9. From the **Amazon Data Firehose delivery streams page,** choose the delivery stream you just created (**ticketUploadStream**). On the **Details** tab, note the stream's Amazon Resource Name (ARN) for later.

For more information on creating delivery streams, see Creating an Amazon Data Firehose Delivery Stream in the *Amazon Data Firehose Developer Guide*. For more information on creating IAM roles, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

You've created the Firehose delivery stream with the required permissions. To continue, see Subscribing the Firehose delivery stream to the Amazon SNS topic.

Subscribing the Firehose delivery stream to the Amazon SNS topic

This topic explains how to create the following resources for the message archiving and analytics example use case:

- The AWS Identity and Access Management (IAM) role that allows the Amazon SNS subscription to put records on the delivery stream.
- The Firehose delivery stream subscription to the Amazon SNS topic.

To create the IAM role for the Amazon SNS subscription

1. Open the Roles page of the IAM console.
2. Choose **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**.

4. For **Choose a use case**, choose **SNS**. Then choose **Next: Permissions**.

5. Choose **Next: Tags**.

6. Choose **Next: Review**.

7. On the **Review** page, for **Role name**, enter **`ticketUploadStreamSubscriptionRole`**. Then choose **Create role**.

8. When the role is created, choose its name (**ticketUploadStreamSubscriptionRole**).

9. On the role's **Summary** page, choose **Add inline policy**.

10. On the **Create policy** page, choose the **JSON** tab, and then paste the following policy into the box:

    JSON

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Action": [
                    "firehose:DescribeDeliveryStream",
                    "firehose:ListDeliveryStreams",
                    "firehose:ListTagsForDeliveryStream",
                    "firehose:PutRecord",
                    "firehose:PutRecordBatch"
                ],
                "Resource": [
                    "arn:aws:firehose:us-east-1:123456789012:deliverystream/
    ticketUploadStream"
                ],
                "Effect": "Allow"
            }
        ]
    }
    ```

    In this policy, replace the AWS account number (*`123456789012`*) with your own, and change the AWS Region (*`us-east-1`*) accordingly.

11. Choose **Review policy**.

12. On the **Review policy** page, for **Name**, enter **`FirehoseSnsPolicy`**. Then choose **Create policy**.
13. On the role's **Summary** page, note the **Role ARN** for later.

For more information on creating IAM roles, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

To subscribe the Firehose delivery stream to the SNS topic

1. Open the Topics page of the Amazon SNS console.
2. On the **Subscriptions**, tab, choose **Create subscription**.
3. Under **Details**, for **Protocol**, choose .
4. For **Endpoint**, enter the Amazon Resource Name (ARN) of the **ticketUploadStream** delivery stream that you created earlier. For example, enter **`arn:aws:firehose:us-east-1:123456789012:deliverystream/ticketUploadStream`**.
5. For **Subscription role ARN**, enter the ARN of the **ticketUploadStreamSubscriptionRole** IAM role that you created earlier. For example, enter **`arn:aws:iam::123456789012:role/ticketUploadStreamSubscriptionRole`**.
6. Select the **Enable raw message delivery** check box.
7. Choose **Create subscription**.

You've created the IAM role and SNS topic subscription. To continue, see Testing and querying an Amazon SNS configuration for effective data management.

Testing and querying an Amazon SNS configuration for effective data management

This topic explains how to test the message archiving and analytics example use case by publishing a message to the Amazon SNS topic. The instructions include an example query that you can run and adapt to your own needs.

To test your configuration

1. Open the Topics page of the Amazon SNS console.
2. Choose the **`ticketTopic`** topic.
3. Choose **Publish message**.

4. On the **Publish message to topic** page, enter the following for the message body. Add a newline character at the end of the message.

   ```
   {"BookingDate":"2020-12-15","BookingTime":"2020-12-15
    04:15:05","Destination":"Miami","FlyingFrom":"Vancouver","TicketNumber":"abcd1234"}
   ```

   Keep all other options as their defaults.

5. Choose **Publish message**.

   For more information on publishing messages, see Publishing an Amazon SNS message.

6. After the delivery stream interval of 60 seconds, open the Amazon Simple Storage Service (Amazon S3) console and choose the Amazon S3 bucket that you created initially.

   The published message appears in the bucket.

To query the data

1. Open the Amazon Athena console.

2. Run a query.

   For example, assume that the `notifications` table in the `default` schema contains the following data:

   ```
   {"BookingDate":"2020-12-15","BookingTime":"2020-12-15
    04:15:05","Destination":"Miami","FlyingFrom":"Vancouver","TicketNumber":"abcd1234"}
   {"BookingDate":"2020-12-15","BookingTime":"2020-12-15
    11:30:15","Destination":"Miami","FlyingFrom":"Omaha","TicketNumber":"efgh5678"}
   {"BookingDate":"2020-12-15","BookingTime":"2020-12-15
    3:30:10","Destination":"Miami","FlyingFrom":"NewYork","TicketNumber":"ijkl9012"}
   {"BookingDate":"2020-12-15","BookingTime":"2020-12-15
    12:30:05","Destination":"Delhi","FlyingFrom":"Omaha","TicketNumber":"mnop3456"}
   ```

   To find the top destination, run the following query:

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

To query for tickets sold during a specific date and time range, run a query like the following:

```
SELECT *
FROM default.notifications
WHERE bookingtime
  BETWEEN TIMESTAMP '2020-12-15 10:00:00'
  AND TIMESTAMP '2020-12-15 12:00:00';
```

You can adapt both sample queries for your own needs. For more information on using Athena to run queries, see Getting Started in the *Amazon Athena User Guide*.

Cleaning up

To avoid incurring usage charges after you're done testing, delete the following resources that you created during the tutorial:

- Amazon SNS subscriptions
- Amazon SNS topic
- Amazon Simple Queue Service (Amazon SQS) queues
- Amazon S3 bucket
- delivery stream
- AWS Identity and Access Management (IAM) roles and policies

Automating Amazon SNS message archiving with an AWS CloudFormation template

To automate the deployment of the Amazon SNS message archiving and analytics example use case, you can use the following YAML template:

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Template for creating an SNS archiving use case
Resources:
  ticketUploadStream:
    DependsOn:
    - ticketUploadStreamRolePolicy
    Type: AWS::KinesisFirehose::DeliveryStream
    Properties:
      S3DestinationConfiguration:
        BucketARN: !Sub 'arn:${AWS::Partition}:s3:::${ticketArchiveBucket}'
        BufferingHints:
          IntervalInSeconds: 60
          SizeInMBs: 1
        CompressionFormat: UNCOMPRESSED
        RoleARN: !GetAtt ticketUploadStreamRole.Arn
  ticketArchiveBucket:
    Type: AWS::S3::Bucket
  ticketTopic:
    Type: AWS::SNS::Topic
  ticketPaymentQueue:
    Type: AWS::SQS::Queue
  ticketFraudQueue:
    Type: AWS::SQS::Queue
  ticketQueuePolicy:
    Type: AWS::SQS::QueuePolicy
    Properties:
      PolicyDocument:
        Statement:
          Effect: Allow
          Principal:
            Service: sns.amazonaws.com
          Action:
            - sqs:SendMessage
          Resource: '*'
          Condition:
            ArnEquals:
              aws:SourceArn: !Ref ticketTopic
      Queues:
        - !Ref ticketPaymentQueue
        - !Ref ticketFraudQueue
  ticketUploadStreamSubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref ticketTopic
      Endpoint: !GetAtt ticketUploadStream.Arn
      Protocol: firehose
      SubscriptionRoleArn: !GetAtt ticketUploadStreamSubscriptionRole.Arn
  ticketPaymentQueueSubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref ticketTopic
      Endpoint: !GetAtt ticketPaymentQueue.Arn
      Protocol: sqs
  ticketFraudQueueSubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref ticketTopic
      Endpoint: !GetAtt ticketFraudQueue.Arn
      Protocol: sqs
  ticketUploadStreamRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Sid: ''
          Effect: Allow
          Principal:
            Service: firehose.amazonaws.com
          Action: sts:AssumeRole
  ticketUploadStreamRolePolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: FirehoseticketUploadStreamRolePolicy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Action:
          - s3:AbortMultipartUpload
          - s3:GetBucketLocation
          - s3:GetObject
          - s3:ListBucket
          - s3:ListBucketMultipartUploads
          - s3:PutObject
          Resource:
          - !Sub 'arn:aws:s3:::${ticketArchiveBucket}'
          - !Sub 'arn:aws:s3:::${ticketArchiveBucket}/*'
      Roles:
      - !Ref ticketUploadStreamRole
  ticketUploadStreamSubscriptionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal:
            Service:
            - sns.amazonaws.com
          Action:
          - sts:AssumeRole
      Policies:
      - PolicyName: SNSKinesisFirehoseAccessPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
          - Action:
            - firehose:DescribeDeliveryStream
            - firehose:ListDeliveryStreams
            - firehose:ListTagsForDeliveryStream
            - firehose:PutRecord
            - firehose:PutRecordBatch
            Effect: Allow
            Resource:
            - !GetAtt ticketUploadStream.Arn
```

Fanout Amazon SNS notifications to Lambda functions for automated processing

Amazon SNS integrates with AWS Lambda, allowing you to trigger Lambda functions in response to Amazon SNS notifications. When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message. The Lambda function receives the message payload as an input parameter and can manipulate the information in the message, publish the message to other SNS topics, or send the message to other AWS services.

Amazon SNS also supports message delivery status attributes for message notifications sent to Lambda endpoints. For more information, see Amazon SNS message delivery status.

Topics

- Prerequisites for integrating Amazon SNS with Lambda functions across regions
- Subscribing a Lambda function to an Amazon SNS topic

Prerequisites for integrating Amazon SNS with Lambda functions across regions

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

- A Lambda function
- An Amazon SNS topic

For information about creating a Lambda function to use with Amazon SNS, see Using Lambda with Amazon SNS. For information about creating an Amazon SNS topic, see Create a topic.

When you use Amazon SNS to deliver messages from opt-in regions to regions which are enabled by default, you must alter the policy created in the AWS Lambda function by replacing the principal `sns.amazonaws.com` with `sns.<opt-in-region>.amazonaws.com`.

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

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

Subscribing a Lambda function to an Amazon SNS topic

This topic explains how to subscribe a Lambda function to an Amazon SNS topic, enabling the function to be triggered by published messages.

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic.
4. In the **Subscriptions** section, choose **Create subscription**.

5. On the **Create subscription** page, in the **Details** section, do the following:

   a. Verify the chosen **Topic ARN**.

   b. For **Protocol** choose AWS Lambda.

   c. For **Endpoint** enter the ARN of a function.

   d. Choose **Create subscription**.

When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message. For information about how to use AWS Lambda with Amazon SNS, including a tutorial, see Using AWS Lambda with Amazon SNS.

Fanout Amazon SNS notifications to Amazon SQS queues for asynchronous processing

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

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

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

Subscribing an Amazon SQS queue to an Amazon SNS topic

To enable an Amazon SNS topic to send messages to an Amazon SQS queue, choose one of the following:

- Use the Amazon SQS console, which simplifies the process. For more information, see Subscribing an Amazon SQS queue to an Amazon SNS topic in the *Amazon Simple Queue Service Developer Guide*.
- Use the following 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.
  2. Give `sqs:SendMessage` permission to the Amazon SNS topic so that it can send messages to the queue.
  3. Subscribe the queue to the Amazon SNS topic.
  4. Give IAM users or AWS accounts the appropriate permissions to publish to the Amazon SNS topic and read messages from the Amazon SQS queue.
  5. Test it out by publishing a message to the topic and reading the message from the queue.

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.

To see an AWS CloudFormation template that creates a topic that sends messages to two queues, see Automate Amazon SNS to Amazon SQS messaging with AWS CloudFormation.

Step 1: Get the ARN of the queue and topic

When subscribing a queue to your topic, you'll need a copy of the ARN for the queue. Similarly, when giving permission for the topic to send messages to the queue, you'll need a copy of the ARN for the topic.

To get the queue ARN, you can use the Amazon SQS console or the GetQueueAttributes API action.

To get the queue ARN from the Amazon SQS console

1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Select the box for the queue whose ARN you want to get.
3. From the **Details** section, copy the ARN value so that you can use it to subscribe to the Amazon SNS topic.

To get the topic ARN, you can use the Amazon SNS console, the `sns-get-topic-attributes` command, or the `GetQueueAttributes` API action.

To get the topic ARN from the Amazon SNS console

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose the topic whose ARN you want to get.
3. From the **Details** section, copy the **ARN** value so that you can use it to give permission for the Amazon SNS topic to send messages to the queue.

Step 2: Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue

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

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

To set a policy on a queue, you can use the Amazon SQS console or the SetQueueAttributes API action. Before you start, make sure you have the ARN for the topic that you want to allow to send messages to the queue. If you are subscribing a queue to multiple topics, your policy must contain one `Statement` element for each topic.

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

1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Select the box for the queue whose policy you want to set, choose the **Access policy** tab, and then choose **Edit**.
3. In the **Access policy** section, define who can access your queue.
   - Add a condition that allows the action for the topic.
   - Set `Principal` to be the Amazon SNS service, as shown in the example below.
   - Use the `aws:SourceArn` or `aws:SourceAccount` global condition keys to protect against the confused deputy scenario. To use these condition keys, set the value to the ARN of your topic. If your queue is subscribed to multiple topics, you can use `aws:SourceAccount` instead.

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

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

Step 3: Subscribe the queue to the Amazon SNS topic

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

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Topics**.
3. On the **Topics** page, choose a topic.
4. On the ***MyTopic*** page, in the **Subscriptions** page, choose **Create subscription**.
5. On the **Create subscription** page, in the **Details** section, do the following:

    a. Verify the **Topic ARN**.

    b. For **Protocol**, choose **Amazon SQS**.

    c. For **Endpoint**, enter the ARN of an Amazon SQS queue.

    d. Choose **Create Subscription**.

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

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

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

You should use AWS Identity and Access Management (IAM) to allow only appropriate users to publish to the Amazon SNS topic and to read/delete messages from the Amazon SQS queue. For more information about controlling actions on topics and queues for IAM users, see Using identity-based policies with Amazon SNS, and Identity and access management in Amazon SQS in the Amazon Simple Queue Service Developer Guide.

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

- Add a policy to an IAM user or group. 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. If you want to give permissions to a topic or queue to another AWS account, the only way you can do that is by adding a policy that has as its principal the AWS account you want to give permissions to.

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

Adding a policy to an IAM user or group

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

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

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

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

Adding a policy to a topic or queue

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

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

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

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

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

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

Step 5: Test the topic's queue subscriptions

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

To publish to a topic using the Amazon SNS console

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

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

1. Using the credentials of the AWS account or IAM user with permission to view messages in the queue, sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Choose a **queue** that is subscribed to the topic.
3. Choose **Send and receive messages**, and then choose **Poll for messages**. A message with a type of **Notification** appears.
4. In the **Body** column, choose **More Details**. The **Message Details** box contains a JSON document that contains the subject and message that you published to the topic. The message looks similar to the following JSON document.

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

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

Automate Amazon SNS to Amazon SQS messaging with AWS CloudFormation

AWS CloudFormation enables you to use a template file to create and configure a collection of AWS resources together as a single unit. This section has an example template that makes it easy to deploy topics that publish to queues. The templates take care of the setup steps for you by creating two queues, creating a topic with subscriptions to the queues, adding a policy to the queues so that the topic can send messages to the queues, and creating IAM users and groups to control access to those resources.

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

Using an AWS CloudFormation template to set up topics and queues within an AWS account

The example template creates an Amazon SNS topic that can send messages to two Amazon SQS queues with appropriate permissions for members of one IAM group to publish to the topic and another to read messages from the queues. The template also creates IAM users that are added to each group.

You copy the template contents into a file. You can also download the template from the AWS CloudFormation Templates page. On the templates page, choose **Browse sample templates by AWS service**and then choose **Amazon Simple Queue Service**.

MySNSTopic is set up to publish to two subscribed endpoints, which are two Amazon SQS queues (MyQueue1 and MyQueue2). MyPublishTopicGroup is an IAM group whose members have permission to publish to MySNSTopic using the Publish API action or sns-publish command. The template creates the IAM users MyPublishUser and MyQueueUser and gives them login profiles and access keys. The user who creates a stack with this template specifies the passwords for the login profiles as input parameters. The template creates access keys for the two IAM users with MyPublishUserKey and MyQueueUserKey. AddUserToMyPublishTopicGroup adds MyPublishUser to the MyPublishTopicGroup so that the user will have the permissions assigned to the group.

MyRDMessageQueueGroup is an IAM group whose members have permission to read and delete messages from the two Amazon SQS queues using the ReceiveMessage and DeleteMessage API actions. AddUserToMyQueueGroup adds MyQueueUser to the MyRDMessageQueueGroup so that the user will have the permissions assigned to the group. MyQueuePolicy assigns permission for MySNSTopic to publish its notifications to the two queues.

The following listing shows the AWS CloudFormation template contents.

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

  "Description" : "AWS CloudFormation Sample Template SNSToSQS: This Template creates
 an SNS topic that can send messages to
  two SQS queues with appropriate permissions for one IAM user to publish to the topic
 and another to read messages from the queues.
  MySNSTopic is set up to publish to two subscribed endpoints, which are two SQS queues
 (MyQueue1 and MyQueue2). MyPublishUser is an IAM user
  that can publish to MySNSTopic using the Publish API. MyTopicPolicy assigns that
 permission to MyPublishUser. MyQueueUser is an IAM user
  that can read messages from the two SQS queues. MyQueuePolicy assigns those
 permissions to MyQueueUser. It also assigns permission for
  MySNSTopic to publish its notifications to the two queues. The template creates
 access keys for the two IAM users with MyPublishUserKey
  and MyQueueUserKey. ***Warning*** you will be billed for the AWS resources used if
 you create a stack from this template.",

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


  "Resources": {
    "MySNSTopic": {
      "Type": "AWS::SNS::Topic",
      "Properties": {
        "Subscription": [{
            "Endpoint": {
              "Fn::GetAtt": ["MyQueue1", "Arn"]
            },
            "Protocol": "sqs"
```

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

```
         ],
         "Resource": {
          "Ref": "MySNSTopic"
         }
        }]
       }
      }]
     }
    },
    "AddUserToMyPublishTopicGroup": {
     "Type": "AWS::IAM::UserToGroupAddition",
     "Properties": {
      "GroupName": {
       "Ref": "MyPublishTopicGroup"
      },
      "Users": [{
       "Ref": "MyPublishUser"
      }]
     }
    },
    "MyQueueUser": {
     "Type": "AWS::IAM::User",
     "Properties": {
      "LoginProfile": {
       "Password": {
        "Ref": "MyQueueUserPassword"
       }
      }
     }
    },
    "MyQueueUserKey": {
     "Type": "AWS::IAM::AccessKey",
     "Properties": {
      "UserName": {
       "Ref": "MyQueueUser"
      }
     }
    },
    "MyRDMessageQueueGroup": {
     "Type": "AWS::IAM::Group",
     "Properties": {
      "Policies": [{
       "PolicyName": "MyQueueGroupPolicy",
       "PolicyDocument": {
            "Statement": [{
              "Effect": "Allow",
              "Action": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage"
              ],
              "Resource": [{
                  "Fn::GetAtt": ["MyQueue1", "Arn"]
                },
                {
                  "Fn::GetAtt": ["MyQueue2", "Arn"]
                }
              ]
            }]
          }
        }]
      }
    },
    "AddUserToMyQueueGroup": {
      "Type": "AWS::IAM::UserToGroupAddition",
      "Properties": {
        "GroupName": {
          "Ref": "MyRDMessageQueueGroup"
        },
        "Users": [{
          "Ref": "MyQueueUser"
        }]
      }
    },
    "MyQueuePolicy": {
      "Type": "AWS::SQS::QueuePolicy",
      "Properties": {
        "PolicyDocument": {
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": "sns.amazonaws.com"
            },
            "Action": ["sqs:SendMessage"],
            "Resource": "*",
            "Condition": {
              "ArnEquals": {
                "aws:SourceArn": {
                  "Ref": "MySNSTopic"
        }
       }
      }
     }]
    },
    "Queues": [{
     "Ref": "MyQueue1"
    }, {
     "Ref": "MyQueue2"
    }]
   }
  }
 },
 "Outputs": {
  "MySNSTopicTopicARN": {
   "Value": {
    "Ref": "MySNSTopic"
   }
  },
  "MyQueue1Info": {
   "Value": {
    "Fn::Join": [
     " ",
     [
      "ARN:",
      {
       "Fn::GetAtt": ["MyQueue1", "Arn"]
      },
      "URL:",
      {
       "Ref": "MyQueue1"
      }
     ]
    ]
   }
  },
  "MyQueue2Info": {
   "Value": {
    "Fn::Join": [
     " ",
     [
      "ARN:",
      {
       "Fn::GetAtt": ["MyQueue2", "Arn"]
```

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

Fanout Amazon SNS notifications to 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 choose whether Amazon SNS uses HTTP or HTTPS to send the POST request to the endpoint. If you use HTTPS, then you can take advantage of the support in Amazon SNS for the following:

- **Server Name Indication (SNI)**—This allows Amazon SNS to support HTTPS endpoints that require SNI, such as a server requiring multiple certificates for hosting multiple domains. For more information about SNI, see Server Name Indication.

- **Basic and Digest Access Authentication**—This allows you to specify a username and password in the HTTPS URL for the HTTP POST request, such as `https://user:password@domain.com` or `https://user@domain.com` The username and password are encrypted over the SSL connection established when using HTTPS. Only the domain name is sent in plaintext. For more information about Basic and Digest Access Authentication, see RFC-2617.

> ⚠ **Important**
>
> Amazon SNS does not currently support private HTTP(S) endpoints.
> HTTPS URLs are only retrievable from the Amazon SNS `GetSubscriptionAttributes` API action, for principals to which you have granted API access.

Note

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

The request contains the subject and message that were published to the topic along with metadata about the notification in a JSON document. The request will look similar to the following HTTP POST request. For details about the HTTP header and the JSON format of the request body, see HTTP/HTTPS headers and HTTP/HTTPS notification JSON format.

Note

Amazon SNS considers all 5XX errors and 429 (too many requests sent) errors as retryable. These errors are subject to the delivery policy. All other errors are considered as permanent failures and retries will not be attempted.

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

{
  "Type" : "Notification",
  "MessageId" : "da41e39f-ea4d-435a-b922-c6aae3915ebe",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "test",
  "Message" : "test message",
  "Timestamp" : "2012-04-25T21:49:25.719Z",
  "SignatureVersion" : "1",
  "Signature" :
 "EXAMPLElDMXvB8r9R83tGoNn0ecwd5UjllzsvSvbItzfaMpN2nk5HVSw7XnOn/49IkxDKz8YrlH2qJXj2iZB0Zo2O71c4
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
   "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55"
}
```

Subscribing an HTTPS endpoint to an Amazon SNS topic

This topic explains how to subscribe HTTP/S endpoints to Amazon SNS topics.

Topics

- Step 1: Make sure your endpoint is ready to process Amazon SNS messages
- Step 2: Subscribe the HTTP/HTTPS endpoint to the Amazon SNS topic
- Step 3: Confirm your Amazon SNS subscription
- Step 4: Optional - Set the delivery policy for the Amazon SNS subscription
- Step 5: Optional - Give users permissions to publish to the Amazon SNS topic
- Step 6: Send Amazon SNS messages to the HTTP/HTTPS endpoint

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

Before you subscribe your HTTP or HTTPS endpoint to a topic, you must make sure that the HTTP or HTTPS endpoint has the capability to handle the HTTP POST requests that Amazon SNS uses to send the subscription confirmation and notification messages. Usually, this means creating and deploying a web application (for example, a Java servlet if your endpoint host is running Linux with Apache and Tomcat) that processes the HTTP requests from Amazon SNS. When you subscribe an HTTP endpoint, Amazon SNS sends it a subscription confirmation request. Your endpoint must be prepared to receive and process this request when you create the subscription because Amazon SNS sends this request at that time. Amazon SNS will not send notifications to the endpoint until you confirm the subscription. Once you confirm the subscription, Amazon SNS will send notifications to the endpoint when a publish action is performed on the subscribed topic.

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

1. Your code should read the HTTP headers of the HTTP POST requests that Amazon SNS sends to your endpoint. Your code should look for the header field `x-amz-sns-message-`

type, which tells you the type of message that Amazon SNS has sent to you. By looking at the header, you can determine the message type without having to parse the body of the HTTP request. There are two types that you need to handle: SubscriptionConfirmation and Notification. The UnsubscribeConfirmation message is used only when the subscription is deleted from the topic.

For details about the HTTP header, see HTTP/HTTPS headers. The following HTTP POST request is an example of a subscription confirmation message.

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

{
  "Type" : "SubscriptionConfirmation",
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37f...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL
 included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
west-2:123456789012:MyTopic&Token=2336412f37...",
  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH+...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

2. Your code should parse the JSON document in the body of the HTTP POST request and content-type text/plain to read the name-value pairs that make up the Amazon SNS message. Use a JSON parser that handles converting the escaped representation of control characters back to their ASCII character values (for example, converting \n to a newline character). You can use an existing JSON parser such as the Jackson JSON Processor or write your own. In

order to send the text in the subject and message fields as valid JSON, Amazon SNS must convert some control characters to escaped representations that can be included in the JSON document. When you receive the JSON document in the body of the POST request sent to your endpoint, you must convert the escaped characters back to their original character values if you want an exact representation of the original subject and messages published to the topic. This is critical if you want to verify the signature of a notification because the signature uses the message and subject in their original forms as part of the string to sign.

3. Your code should verify the authenticity of a notification, subscription confirmation, or unsubscribe confirmation message sent by Amazon SNS. Using information contained in the Amazon SNS message, your endpoint can recreate the signature so that you can verify the contents of the message by matching your signature with the signature that Amazon SNS sent with the message. For more information about verifying the signature of a message, see Verifying the signatures of Amazon SNS messages.

4. Based on the type specified by the header field `x-amz-sns-message-type`, your code should read the JSON document contained in the body of the HTTP request and process the message. Here are the guidelines for handling the two primary types of messages:

   SubscriptionConfirmation

   Read the value for `SubscribeURL` and visit that URL. To confirm the subscription and start receiving notifications at the endpoint, you must visit the `SubscribeURL`URL (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. When you visit the URL, you will get back a response that looks like the following XML document. The document returns the subscription ARN for the endpoint within the `ConfirmSubscriptionResult` element.

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

As an alternative to visiting the `SubscribeURL`, you can confirm the subscription using the ConfirmSubscription action with the `Token` set to its corresponding value in the `SubscriptionConfirmation` message. If you want to allow only the topic owner and subscription owner to be able to unsubscribe the endpoint, you call the `ConfirmSubscription` action with an AWS signature.

Notification

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

For details about the format of the `Notification` message, see HTTP/HTTPS headers. The following HTTP POST request is an example of a notification message sent to the endpoint example.com.

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

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

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

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

   Amazon SNS uses the `x-amz-sns-message-id` header field to uniquely identify each message published to an Amazon SNS topic. By comparing the IDs of the messages you have processed with incoming messages, you can determine whether the message is a retry attempt.
7. If you are subscribing an HTTPS endpoint, make sure that your endpoint has a server certificate from a trusted Certificate Authority (CA). Amazon SNS will only send messages to HTTPS endpoints that have a server certificate signed by a CA trusted by Amazon SNS.
8. Deploy the code that you have created to receive Amazon SNS messages. When you subscribe the endpoint, the endpoint must be ready to receive at least the subscription confirmation message.

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

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

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

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Subscriptions**.
3. Choose the **Create subscription**.
4. In the **Protocol** drop-down list, select **HTTP** or **HTTPS**.
5. In the **Endpoint** box, paste in the URL for the endpoint that you want the topic to send messages to and then choose **Create subscription**.
6. The confirmation message is displayed. Choose **Close**.

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

Step 3: Confirm your Amazon SNS subscription

To confirm your Amazon SNS subscription, follow these steps to ensure your endpoint can successfully receive messages. This process involves setting up your endpoint to handle incoming confirmation messages, retrieving the confirmation URL, and confirming the subscription. You can confirm the subscription either automatically or manually, depending on your setup.

1. After subscribing to an Amazon SNS topic, Amazon SNS sends a confirmation message to your endpoint. This message contains a `SubscribeURL` that you must use to confirm the subscription.
2. Your endpoint must be set up to listen for incoming messages from Amazon SNS. When the confirmation message arrives, extract the **`SubscribeURL`** from the message.
3. Once you have the `SubscribeURL`, you can confirm the subscription in one of two ways:

   - **Automatic confirmation** – Your endpoint can automatically confirm the subscription by sending an **HTTP GET request** to the `SubscribeURL`.

     This method does not require manual intervention.

   - **Manual confirmation** – If automatic confirmation is not set up, **copy** the **`SubscribeURL`** from the confirmation message and **paste** it into your browser’s address bar.

This will confirm the subscription manually.

4. You can also verify the **subscription status** using the Amazon SNS console:

   a. Sign in to the Amazon SNS console.

   b. In the navigation pane, choose **Subscriptions**.

   c. Find your **subscription** in the list.

      - If confirmed, the `SubscriptionArn` will be displayed.
      - If still unconfirmed, it will show as `PendingConfirmation`.

Step 4: Optional - Set the delivery policy for the Amazon SNS subscription

By default, if the initial delivery of the message fails, Amazon SNS attempts up to three retries with a delay between failed attempts set at 20 seconds. As discussed in Step 1, 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 also specify the content type for your HTTP/S notifications in `DeliveryPolicy`. For more information, see Creating an HTTP/S delivery policy.

Step 5: Optional - Give users permissions to publish to the Amazon SNS topic

By default, the topic owner has permissions to publish the topic. To enable other users or applications to publish to the topic, you should use AWS Identity and Access Management (IAM) to give publish permission to the topic. For more information about giving permissions for Amazon SNS actions to IAM users, see Using identity-based policies with Amazon SNS.

There are two ways to control access to a topic:

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

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

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

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

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

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

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

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

Step 6: Send Amazon SNS messages to the HTTP/HTTPS endpoint

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

If you followed Step 1, the code that you deployed at your endpoint should process the notification.

To publish to a topic using the Amazon SNS console

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

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

Verifying the signatures of Amazon SNS messages

Amazon SNS uses message signatures to confirm the authenticity of messages sent to your HTTP endpoint. To ensure message integrity and prevent spoofing, you **must** verify the signature before processing any Amazon SNS messages.

When should you verify Amazon SNS signatures?

You should verify Amazon SNS message signatures in the following scenarios:

- When Amazon SNS sends a notification message to your HTTP(S) endpoint.
- When Amazon SNS sends a confirmation message to your endpoint after a `Subscribe` or `Unsubscribe` API call.

Amazon SNS supports two signature versions:

- SignatureVersion1 – Uses an SHA1 hash of the message.
- SignatureVersion2 – Uses an SHA256 hash of the message. This provides stronger security and is the recommended option.

To correctly verify SNS message signatures, follow these best practices:

- Always retrieve the signing certificate using HTTPS to prevent unauthorized interception attacks.
- Check that the certificate is issued by Amazon SNS.
- Confirm that the certificate's chain of trust is valid.
- The certificate should come from an SNS-signed URL.
- Don't trust any certificates provided in the message without validation.
- Reject any message with an unexpected `TopicArn` to prevent spoofing.
- The AWS SDKs for Amazon SNS provide built-in validation logic, reducing the risk of misimplementation.

Configuring the message signature version on Amazon SNS topics

Configuring the message signature version on Amazon SNS topics allows you to enhance the security and compatibility of your message verification process.

Select between `SignatureVersion`**1** (SHA1) and `SignatureVersion`**2** (SHA256) to control the hashing algorithm used for signing messages. Amazon SNS topics default to `SignatureVersion`**1**. You can configure this setting using the `SetTopicAttributes` API action.

Use the following example to set the topic attribute `SignatureVersion` using the AWS CLI:

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-east-2:123456789012:MyTopic \
    --attribute-name SignatureVersion \
    --attribute-value 2
```

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

Verifying the signature of an Amazon SNS message when using HTTP query-based requests ensures the message's authenticity and integrity. This process confirms that the message originates from Amazon SNS and has not been tampered with during transit. By parsing the message, constructing the correct string to sign, and validating the signature against a trusted public key, you safeguard your system against spoofing and unauthorized message alterations.

1. Extract **key-value pairs** from the JSON document in the HTTP POST request body sent by Amazon SNS. These fields are required to construct the **string to sign**.

   - `Message`
   - `Subject` (if present)
   - `MessageId`
   - `Timestamp`
   - `TopicArn`
   - `Type`

   For example:

   ```
   MESSAGE_FILE="message.json"
   FIELDS=("Message" "MessageId" "Subject" "Timestamp" "TopicArn" "Type")
   ```

   > **Note**
   >
   > If any field contains escaped characters (for example, \n), convert them to their **original form** to ensure an exact match.

2. Locate the `SigningCertURL` field in the Amazon SNS message. This certificate contains the public key needed to verify the message signature. For example:

   ```
   SIGNING_CERT_URL=$(jq -r '.SigningCertURL' "$MESSAGE_FILE")
   ```

3. Ensure the `SigningCertURL` is from a trusted AWS domain (for example, https://sns.us-east-1.amazonaws.com). Reject any URLs **outside AWS domains** for security reasons.

4. Download the **X.509 certificate** from the provided URL. For example:

```
curl -s "$SIGNING_CERT_URL" -o signing_cert.pem
```

5. Extract the **public key** from the downloaded X.509 certificate. The public key allows you to decrypt the message's signature and verify its integrity. For example:

```
openssl x509 -pubkey -noout -in signing_cert.pem > public_key.pem
```

6. Different message types require different key-value pairs in the string to sign. Identify the **message type** (`Type` field in the Amazon SNS message) to determine which **key-value pairs** to include:

   - **Notification message** – Includes `Message`, `MessageId`, `Subject` (if present), `Timestamp`, `TopicArn`, and `Type`.
   - **SubscriptionConfirmation** or **UnsubscribeConfirmation message** – Includes `Message`, `MessageId`, `SubscribeURL`, `Timestamp`, `Token`, `TopicArn`, and `Type`.

7. Amazon SNS requires the string to sign to follow a strict, fixed field order for verification. **Only the explicitly required fields must be included**—no extra fields can be added. Optional fields, such as `Subject`, must be included only if present in the message and must appear in the exact position defined by the required field order. For example:

```
KeyNameOne\nValueOne\nKeyNameTwo\nValueTwo
```

> **Important**
>
> Do not add a newline character at the end of the string.

8. Arrange the **key-value pairs** in byte-sort order (alphabetical by key name).

9. Construct the **string to sign** using the following format example:

```
STRING_TO_SIGN=""
for FIELD in "${FIELDS[@]}"; do
    VALUE=$(jq -r --arg field "$FIELD" '.[$field]' "$MESSAGE_FILE")
    STRING_TO_SIGN+="$FIELD\n$VALUE"
    # Append a newline after each field except the last one
    if [[ "$FIELD" != "Type" ]]; then
        STRING_TO_SIGN+="\n"
    fi
```

```
done
```

Notification message example:

```
Message
My Test Message
MessageId
4d4dc071-ddbf-465d-bba8-08f81c89da64
Subject
My subject
Timestamp
2019-01-31T04:37:04.321Z
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFCOXTC0P
Type
Notification
```

SubscriptionConfirmation example:

```
Message
Please confirm your subscription
MessageId
3d891288-136d-417f-bc05-901c108273ee
SubscribeURL
https://sns.us-east-2.amazonaws.com/...
Timestamp
2024-01-01T00:00:00.000Z
Token
abc123...
TopicArn
arn:aws:sns:us-east-2:123456789012:MyTopic
Type
SubscriptionConfirmation
```

10. The `Signature` field in the message is Base64-encoded. You need to **decode** it to compare its **raw binary form** with the **derived hash**. For example:

    ```
    SIGNATURE=$(jq -r '.Signature' "$MESSAGE_FILE")
    echo "$SIGNATURE" | base64 -d > signature.bin
    ```

11. Use the `SignatureVersion` field to select the hash algorithm:

- For `SignatureVersion`**1**, use **SHA1** (for example, `-sha1`).
- For `SignatureVersion`**2**, use **SHA256** (for example, `-sha256`).

12. To confirm the authenticity of the Amazon SNS message, generate a **hash** of the constructed string and verify the signature using the **public key**.

    ```
    openssl dgst -sha256 -verify public_key.pem -signature signature.bin <<<
     "$STRING_TO_SIGN"
    ```

    If the signature is valid, the output is `Verified OK`. Otherwise, the output is `Verification Failure`.

Example script with error handling

The following example script automates the verification process:

```
#!/bin/bash

# Path to the local message file
MESSAGE_FILE="message.json"

# Extract the SigningCertURL and Signature from the message
SIGNING_CERT_URL=$(jq -r '.SigningCertURL' "$MESSAGE_FILE")
SIGNATURE=$(jq -r '.Signature' "$MESSAGE_FILE")

# Fetch the X.509 certificate
curl -s "$SIGNING_CERT_URL" -o signing_cert.pem

# Extract the public key from the certificate
openssl x509 -pubkey -noout -in signing_cert.pem > public_key.pem

# Define the fields to include in the string to sign
FIELDS=("Message" "MessageId" "Subject" "Timestamp" "TopicArn" "Type")

# Initialize the string to sign
STRING_TO_SIGN=""

# Iterate over the fields to construct the string to sign
for FIELD in "${FIELDS[@]}"; do
    VALUE=$(jq -r --arg field "$FIELD" '.[$field]' "$MESSAGE_FILE")
    STRING_TO_SIGN+="$FIELD\n$VALUE"
```

```
    # Append a newline after each field except the last one
    if [[ "$FIELD" != "Type" ]]; then
        STRING_TO_SIGN+="\n"
    fi
done

# Verify the signature
echo -e "$STRING_TO_SIGN" | openssl dgst -sha256 -verify public_key.pem -signature
 <(echo "$SIGNATURE" | base64 -d)
```

Parsing Amazon SNS message formats

When Amazon SNS sends messages to HTTP/HTTPS endpoints, they contain both HTTP headers and a JSON message body. These messages follow a structured format that includes metadata such as the message type, topic ARN, timestamps, and digital signatures. By correctly parsing Amazon SNS messages, you can determine whether a message is a subscription confirmation, notification, or unsubscribe confirmation, extract relevant data, and verify authenticity using signature validation.

HTTP/HTTPS headers

When Amazon SNS sends a subscription confirmation, notification, or unsubscribe confirmation message to HTTP/HTTPS endpoints, it sends a POST message with a number of Amazon SNS-specific header values. You can use header values for such tasks as identifying the message type without having to parse the JSON message body to read the `Type` value. By default, Amazon SNS sends all the notification to HTTP/S endpoints with `Content-Type` set to `text/plain; charset=UTF-8`. To choose a `Content-Type` other than text/plain (default), see `headerContentType` in [Creating an HTTP/S delivery policy](#).

`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 (UUID), unique for each message published. For a notification that Amazon SNS resends during a retry, the message ID of the original message is used.

`x-amz-sns-topic-arn`

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

x-amz-sns-subscription-arn

The ARN for the subscription to this endpoint.

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

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

HTTP/HTTPS subscription confirmation JSON format

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

Note

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

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

Type

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

MessageId

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

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.

Message

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

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

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.

SignatureVersion

Version of the Amazon SNS signature used.

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

Signature

Base64-encoded `SHA1withRSA` or `SHA256withRSA` signature of the `Message`, `MessageId`, `Type`, `Timestamp`, and `TopicArn` values.

SigningCertURL

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

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

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

{
  "Type" : "SubscriptionConfirmation",
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL
 included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
west-2:123456789012:MyTopic&Token=2336412f37...",
  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH
+DcEwjAPg8O9mY8dReBSwksfg2S7WKQcikcNKWLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkkOxYDVrY0Ad8L
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

HTTP/HTTPS notification JSON format

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

`Type`

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

`MessageId`

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

`TopicArn`

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

`Subject`

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

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

`Message`

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

`Timestamp`

The time (GMT) when the notification was published.

`SignatureVersion`

Version of the Amazon SNS signature used.

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

`Signature`

Base64-encoded `SHA1withRSA` or `SHA256withRSA` signature of the `Message`, `MessageId`, `Subject` (if present), `Type`, `Timestamp`, and `TopicArn` values.

`SigningCertURL`

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

`UnsubscribeURL`

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

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

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

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

HTTP/HTTPS unsubscribe confirmation JSON format

After an HTTP/HTTPS endpoint is unsubscribed from a topic, Amazon SNS sends an unsubscribe confirmation message to the endpoint.

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

`Type`

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

`MessageId`

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

`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.

`Message`

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

```
You have chosen to deactivate subscription arn:aws:sns:us-
east-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this
 operation and restore the subscription, visit the SubscribeURL included in this
 message.
```

`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.

`SignatureVersion`

Version of the Amazon SNS signature used.

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

`Signature`

Base64-encoded `SHA1withRSA` or `SHA256withRSA` signature of the `Message`, `MessageId`, `Type`, `Timestamp`, and `TopicArn` values.

`SigningCertURL`

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

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

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

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

SetSubscriptionAttributes delivery policy JSON format

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

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

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

```
{
    "healthyRetryPolicy" : {
        "minDelayTarget" :  int,
        "maxDelayTarget" : int,
        "numRetries" : int,
        "numMaxDelayRetries" : int,
        "backoffFunction" : "linear|arithmetic|geometric|exponential"
    },
    "throttlePolicy" : {
        "maxReceivesPerSecond" : int
    },
    "requestPolicy" : {
        "headerContentType" : "text/plain | application/json | application/xml"
    }
}
```

For more information about the `SetSubscriptionAttribute` action, go to SetSubscriptionAttributes in the *Amazon Simple Notification Service API Reference*. For more information on the supported HTTP content-type headers, see Creating an HTTP/S delivery policy.

SetTopicAttributes delivery policy JSON format

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

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

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

```
{
    "http" : {
        "defaultHealthyRetryPolicy" : {
            "minDelayTarget":  int,
            "maxDelayTarget": int,
            "numRetries": int,
            "numMaxDelayRetries": int,
            "backoffFunction": "linear|arithmetic|geometric|exponential"
        },
        "disableSubscriptionOverrides" : Boolean,
        "defaultThrottlePolicy" : {
            "maxReceivesPerSecond" : int
        },
        "defaultRequestPolicy" : {
            "headerContentType" : "text/plain | application/json | application/xml"
        }
    }
}
```

For more information about the `SetTopicAttribute` action, go to SetTopicAttributes in the *Amazon Simple Notification Service API Reference*. For more information on the supported HTTP content-type headers, see Creating an HTTP/S delivery policy.

Fanout Amazon SNS events to AWS Event Fork Pipelines

For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Data Firehose. You can subscribe Firehose delivery streams to SNS topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see Fanout to Firehose delivery streams.

You can use Amazon SNS to build event-driven applications which use subscriber services to perform work automatically in response to events triggered by publisher services. This architectural pattern can make services more reusable, interoperable, and scalable. However, it can be labor-intensive to fork the processing of events into pipelines that address common event handling requirements, such as event storage, backup, search, analytics, and replay.

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

For an AWS Event Fork Pipelines use case, see Deploying and testing the Amazon SNS event fork pipelines sample application.

Topics

- How AWS Event Fork Pipelines works
- Deploying AWS Event Fork Pipelines
- Deploying and testing the Amazon SNS event fork pipelines sample application
- Subscribing AWS Event Fork Pipelines to an Amazon SNS topic

How AWS Event Fork Pipelines works

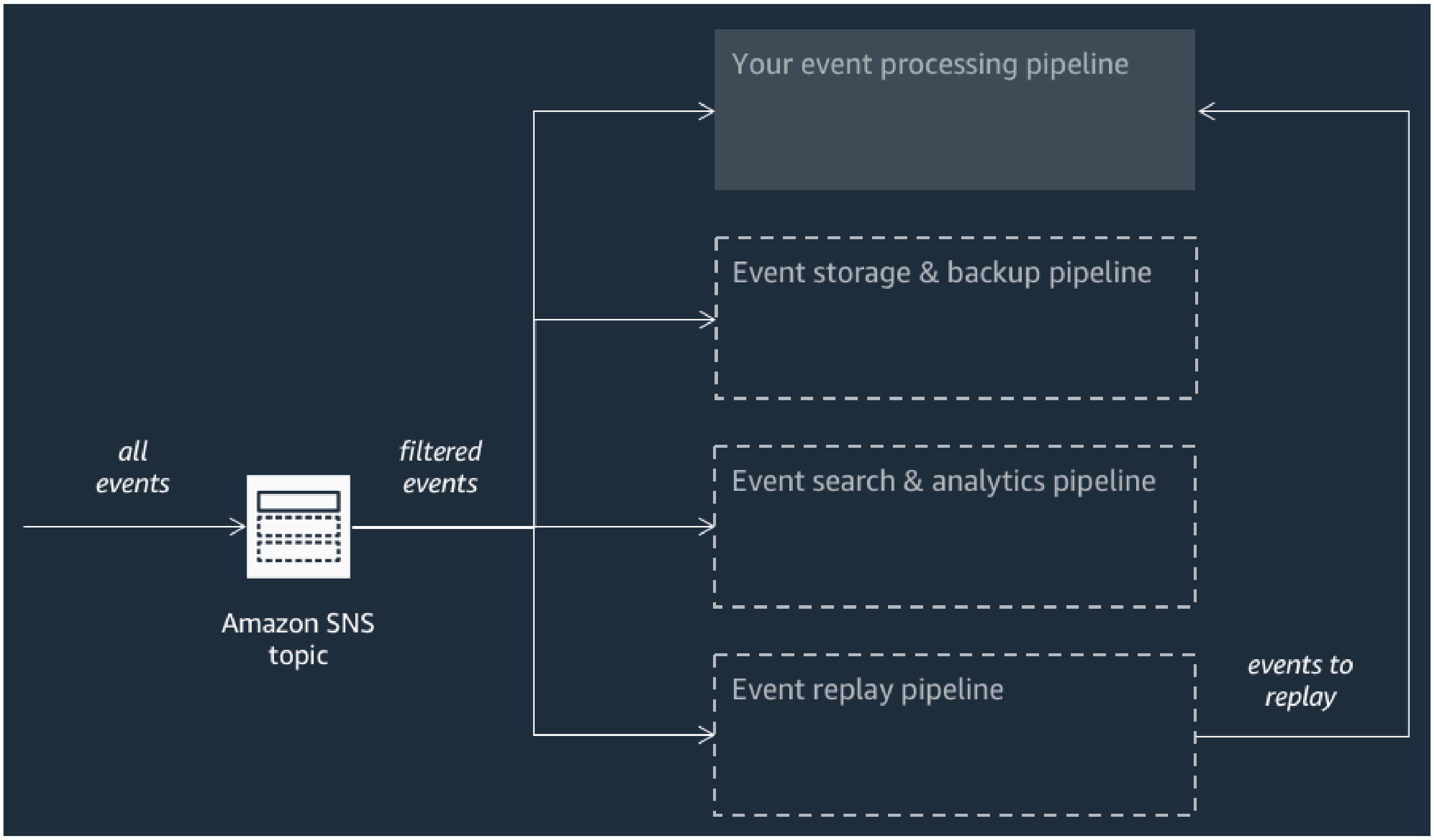
AWS Event Fork Pipelines is a serverless design pattern. However, it is also a suite of nested serverless applications based on AWS SAM (which you can deploy directly from the AWS Serverless

Application Repository (AWS SAR) to your AWS account in order to enrich your event-driven platforms). You can deploy these nested applications individually, as your architecture requires.

Topics

- The event storage and backup pipeline
- The event search and analytics pipeline
- The event replay pipeline

The following diagram shows an AWS Event Fork Pipelines application supplemented by three nested applications. You can deploy any of the pipelines from the AWS Event Fork Pipelines suite on the AWS SAR independently, as your architecture requires.


Your event processing pipeline
Event storage & backup pipeline
all events
filtered events
Event search & analytics pipeline
Amazon SNS topic
events to replay
Event replay pipeline

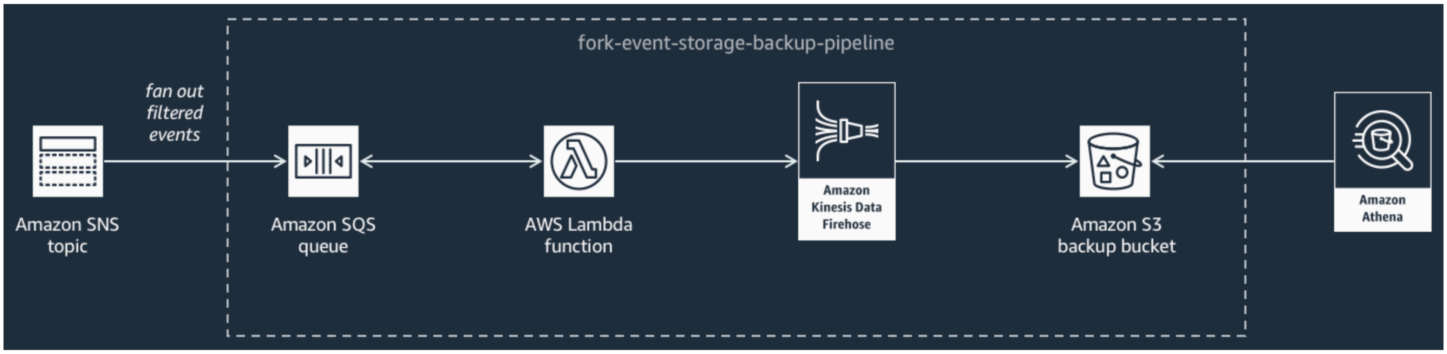
Each pipeline is subscribed to the same Amazon SNS topic, allowing itself to process events in parallel as these events are published to the topic. Each pipeline is independent and can set its own Subscription Filter Policy. This allows a pipeline to process only a subset of the events that it is interested in (rather than all events published to the topic).

> **Note**
>
> Because you place the three AWS Event Fork Pipelines alongside your regular event processing pipelines (possibly already subscribed to your Amazon SNS topic), you don't need to change any portion of your current message publisher to take advantage of AWS Event Fork Pipelines in your existing workloads.

The event storage and backup pipeline

The following diagram shows the Event Storage and Backup Pipeline. You can subscribe this pipeline to your Amazon SNS topic to automatically back up the events flowing through your system.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, an AWS Lambda function that automatically polls for these events in the queue and pushes them into an stream, and an Amazon S3 bucket that durably backs up the events loaded by the stream.

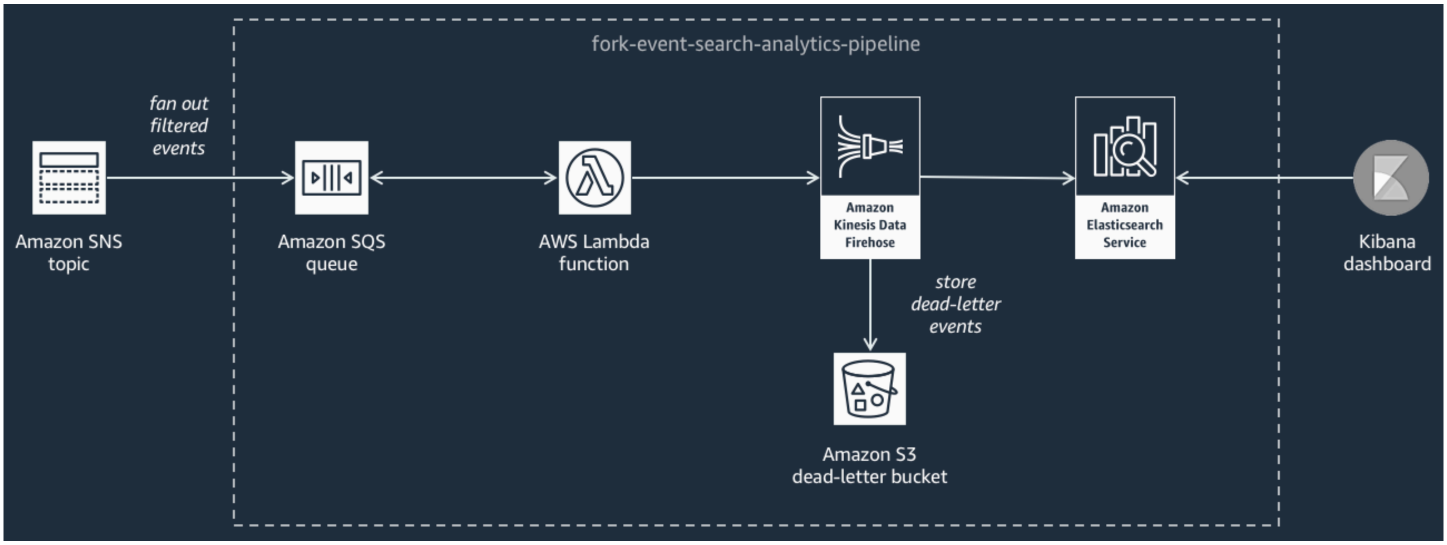


To fine-tune the behavior of your Firehose stream, you can configure it to buffer, transform, and compress your events prior to loading them into the bucket. As events are loaded, you can use Amazon Athena to query the bucket using standard SQL queries. You can also configure the pipeline to reuse an existing Amazon S3 bucket or create a new one.

The event search and analytics pipeline

The following diagram shows the Event Search and Analytics Pipeline. You can subscribe this pipeline to your Amazon SNS topic to index the events that flow through your system in a search domain and then run analytics on them.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, an AWS Lambda function that polls events from the queue and pushes them into an stream, an Amazon OpenSearch Service domain that indexes the events loaded by the Firehose stream, and an Amazon S3 bucket that stores the dead-letter events that can't be indexed in the search domain.


fork-event-search-analytics-pipeline
fan out filtered events
Amazon SNS topic
Amazon SQS queue
AWS Lambda function
Amazon Kinesis Data Firehose
Amazon Elasticsearch Service
Kibana dashboard
store dead-letter events
Amazon S3 dead-letter bucket

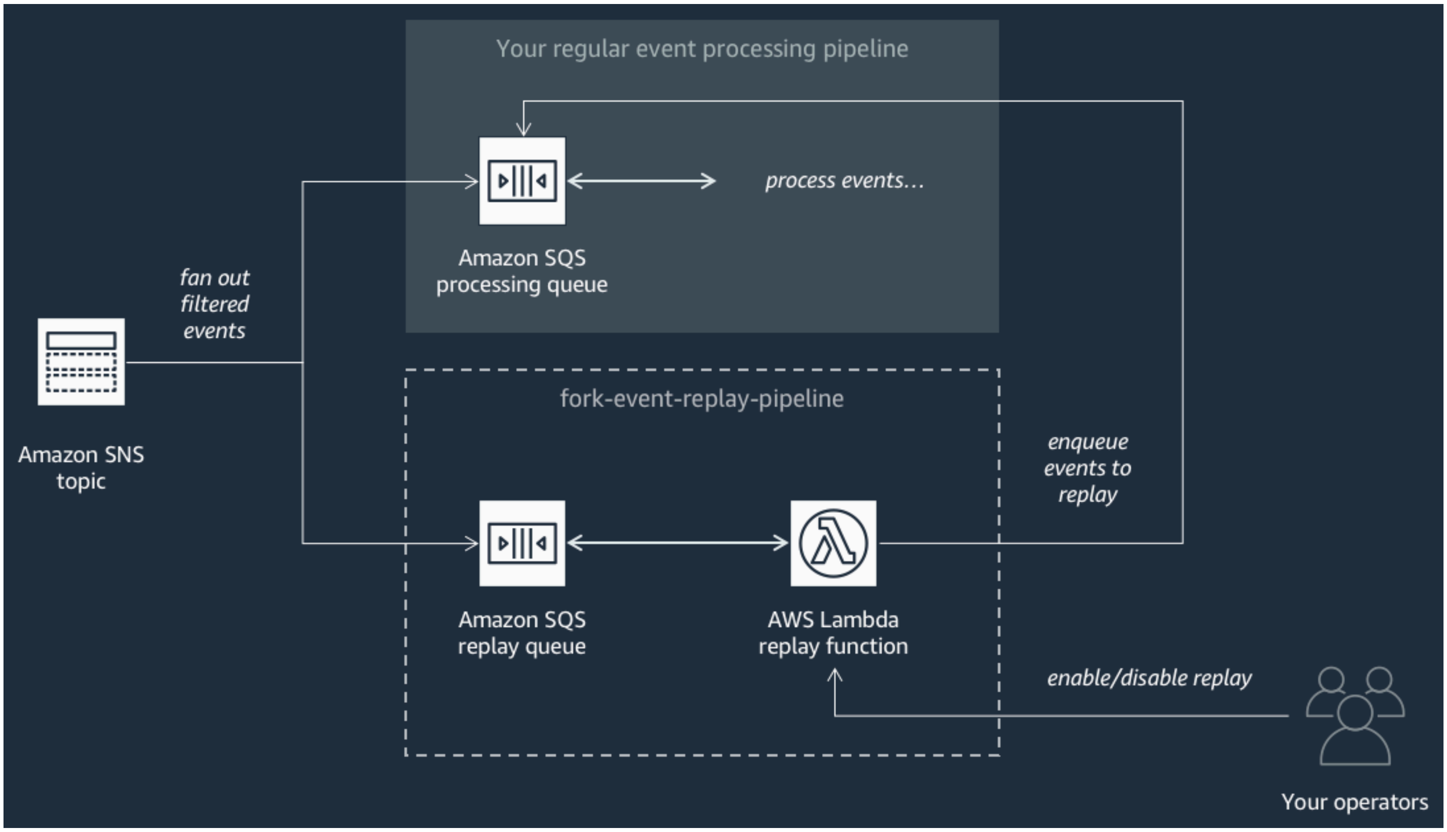
To fine-tune your Firehose stream in terms of event buffering, transformation, and compression, you can configure this pipeline.

You can also configure whether the pipeline should reuse an existing OpenSearch domain in your AWS account or create a new one for you. As events are indexed in the search domain, you can use Kibana to run analytics on your events and update visual dashboards in real-time.

The event replay pipeline

The following diagram shows the Event Replay Pipeline. To record the events that have been processed by your system for the past 14 days (for example when your platform needs to recover from failure), you can subscribe this pipeline to your Amazon SNS topic and then reprocess the events.

This pipeline is comprised of an Amazon SQS queue that buffers the events delivered by the Amazon SNS topic, and an AWS Lambda function that polls events from the queue and redrives them into your regular event processing pipeline, which is also subscribed to your topic.



Note

By default, the replay function is disabled, not redriving your events. If you need to reprocess events, you must enable the Amazon SQS replay queue as an event source for the AWS Lambda replay function.

Deploying AWS Event Fork Pipelines

The AWS Event Fork Pipelines suite (choose **Show apps that create custom IAM roles or resource policies**) is available as a group of public applications in the AWS Serverless Application Repository, from where you can deploy and test them manually using the AWS Lambda console. For information about deploying pipelines using the AWS Lambda console, see Subscribing AWS Event Fork Pipelines to an Amazon SNS topic.

In a production scenario, we recommend embedding AWS Event Fork Pipelines within your overall application's AWS SAM template. The nested-application feature lets you do this by adding the resource `AWS::Serverless::Application` to your AWS SAM template, referencing the AWS SAR `ApplicationId` and the `SemanticVersion` of the nested application.

For example, you can use the Event Storage and Backup Pipeline as a nested application by adding the following YAML snippet to the `Resources` section of your AWS SAM template.

```
Backup:
    Type: AWS::Serverless::Application
  Properties:
    Location:
      ApplicationId: arn:aws:serverlessrepo:us-east-2:123456789012:applications/fork-
event-storage-backup-pipeline
      SemanticVersion: 1.0.0
    Parameters:
      #The ARN of the Amazon SNS topic whose messages should be backed up to the Amazon
 S3 bucket.
      TopicArn: !Ref MySNSTopic
```

When you specify parameter values, you can use AWS CloudFormation intrinsic functions to reference other resources in your template. For example, in the YAML snippet above, the `TopicArn` parameter references the `AWS::SNS::Topic` resource `MySNSTopic`, defined elsewhere in the AWS SAM template. For more information, see the Intrinsic Function Reference in the *AWS CloudFormation User Guide*.

> **Note**
>
> The AWS Lambda console page for your AWS SAR application includes the **Copy as SAM Resource** button, which copies the YAML required for nesting an AWS SAR application to the clipboard.

Deploying and testing the Amazon SNS event fork pipelines sample application

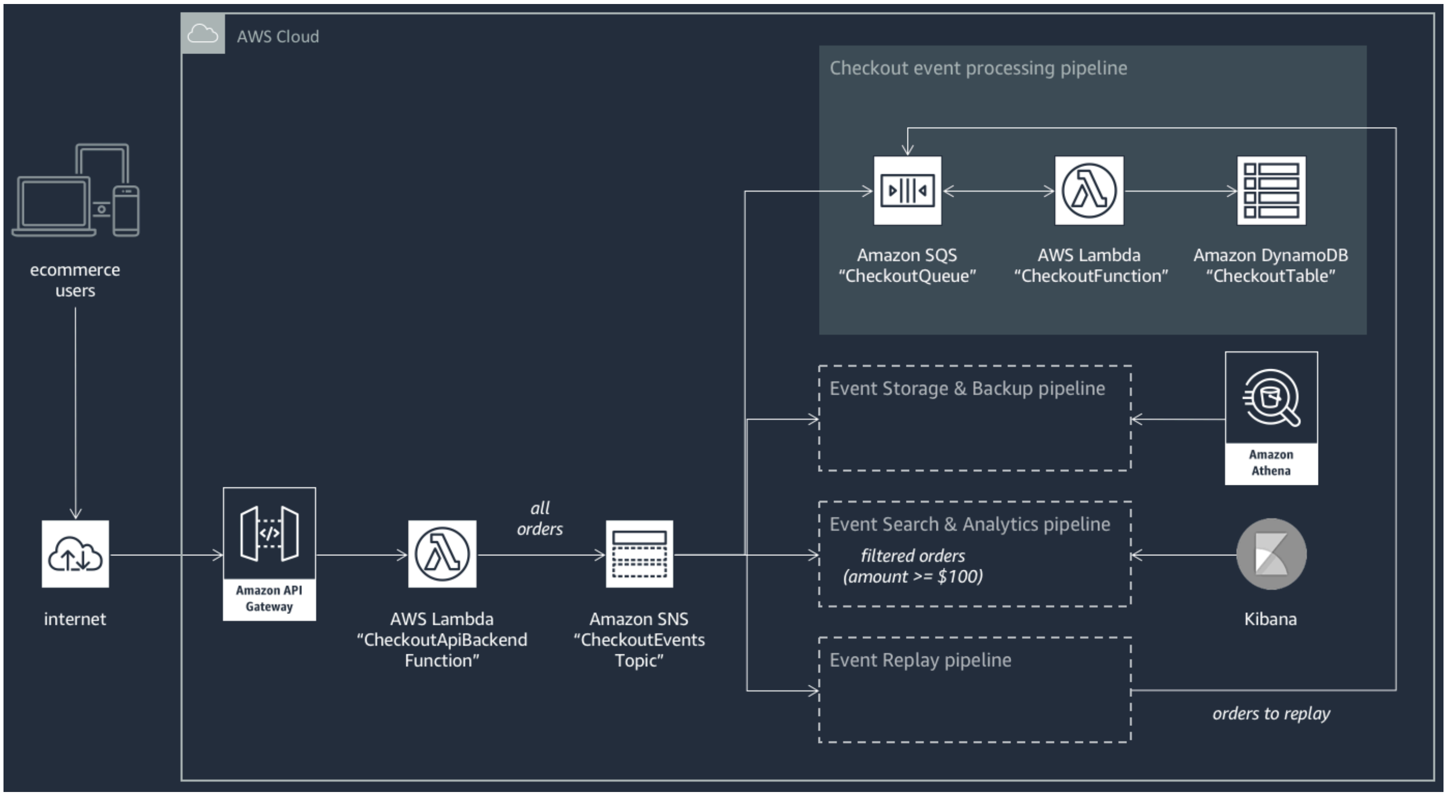
To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source nested applications, based on the AWS Serverless Application Model (AWS SAM), which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

This page shows how you can use the AWS Management Console to deploy and test the AWS Event Fork Pipelines sample application.

> **Important**
>
> To avoid incurring unwanted costs after you finish deploying the AWS Event Fork Pipelines sample application, delete its AWS CloudFormation stack. For more information, see Deleting a Stack on the AWS CloudFormation Console in the *AWS CloudFormation User Guide*.

AWS Event Fork Pipelines use case example

The following scenario describes an event-driven, serverless e-commerce application that uses AWS Event Fork Pipelines. You can use this example e-commerce application in the AWS Serverless Application Repository and then deploy it in your AWS account using the AWS Lambda console, where you can test it and examine its source code in GitHub.


AWS Cloud
Checkout event processing pipeline
Amazon SQS "CheckoutQueue"
AWS Lambda "CheckoutFunction"
Amazon DynamoDB "CheckoutTable"
ecommerce users
Event Storage & Backup pipeline
Amazon Athena
Event Search & Analytics pipeline
filtered orders (amount >= $100)
Kibana
internet
Amazon API Gateway
AWS Lambda "CheckoutApiBackend Function"
all orders
Amazon SNS "CheckoutEvents Topic"
Event Replay pipeline
orders to replay

This e-commerce application takes orders from buyers through a RESTful API hosted by API Gateway and backed by the AWS Lambda function `CheckoutApiBackendFunction`. This function publishes all received orders to an Amazon SNS topic named CheckoutEventsTopic which, in turn, fans out the orders to four different pipelines.

The first pipeline is the regular checkout-processing pipeline designed and implemented by the owner of the e-commerce application. This pipeline has the Amazon SQS queue CheckoutQueue that buffers all received orders, an AWS Lambda function named CheckoutFunction that polls the queue to process these orders, and the DynamoDB table CheckoutTable that securely saves all placed orders.

Applying AWS Event Fork Pipelines

The components of the e-commerce application handle the core business logic. However, the e-commerce application owner also needs to address the following:

- **Compliance**—secure, compressed backups encrypted at rest and sanitization of sensitive information
- **Resiliency**—replay of most recent orders in case of the disruption of the fulfillment process
- **Searchability**—running analytics and generating metrics on placed orders

Instead of implementing this event processing logic, the application owner can subscribe AWS Event Fork Pipelines to the CheckoutEventsTopic Amazon SNS topic

- The event storage and backup pipeline is configured to transform data to remove credit card details, buffer data for 60 seconds, compress it using GZIP, and encrypt it using the default customer managed key for Amazon S3. This key is managed by AWS and powered by the AWS Key Management Service (AWS KMS).

  For more information, see Choose Amazon S3 For Your Destination, Amazon Data Firehose Data Transformation, and Configure Settings in the *Amazon Data Firehose Developer Guide*.
- The event search and analytics pipeline is configured with an index retry duration of 30 seconds, a bucket for storing orders that fail to be indexed in the search domain, and a filter policy to restrict the set of indexed orders.

  For more information, see Choose OpenSearch Service for your Destination in the *Amazon Data Firehose Developer Guide*.
- The event replay pipeline is configured with the Amazon SQS queue part of the regular order-processing pipeline designed and implemented by the e-commerce application owner.

For more information, see Queue Name and URL in the *Amazon Simple Queue Service Developer Guide*.

The following JSON filter policy is set in the configuration for the Event Search and Analytics Pipeline. It matches only incoming orders in which the total amount is $100 or higher. For more information, see Amazon SNS message filtering.

```
{
   "amount": [{ "numeric": [ ">=", 100 ] }]
}
```

Using the AWS Event Fork Pipelines pattern, the e-commerce application owner can avoid the development overhead that often follows coding undifferentiating logic for event handling. Instead, she can deploy AWS Event Fork Pipelines directly from the AWS Serverless Application Repository into her AWS account.

Step 1: Deploying the sample Amazon SNS application

1. Sign in to the AWS Lambda console.
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:

    a. Choose **Browse serverless app repository**, **Public applications**, **Show apps that create custom IAM roles or resource policies**.

    b. Search for `fork-example-ecommerce-checkout-api` and then choose the application.

4. On the **fork-example-ecommerce-checkout-api** page, do the following:

    a. In the **Application settings** section, enter an **Application name** (for example, `fork-example-ecommerce-my-app`).

    > **Note**
    >
    > - To find your resources easily later, keep the prefix `fork-example-ecommerce`.
    > - For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

b. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:

- `DEBUG`
- `ERROR`
- `INFO` (default)
- `WARNING`

5. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then, at the bottom of the page, choose **Deploy**.

On the **Deployment status for fork-example-ecommerce-*my-app*** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE_IN_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

> **Note**
>
> It might take 20-30 minutes for all resources to be deployed.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Step 2: Executing the SNS-linked sample application

1. In the AWS Lambda console, on the navigation panel, choose **Applications**.
2. On the **Applications** page, in the search field, search for `serverlessrepo-fork-example-ecommerce-`*`my-app`* and then choose the application.
3. In the **Resources** section, do the following:

   a. To find the resource whose type is **ApiGateway RestApi**, sort the resources by **Type**, for example `ServerlessRestApi`, and then expand the resource.

   b. Two nested resources are displayed, of types **ApiGateway Deployment** and **ApiGateway Stage**.

c. Copy the link **Prod API endpoint** and append `/checkout` to it, for example:

```
https://abcdefghij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

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

```
{
   "id": 15311,
   "date": "2019-03-25T23:41:11-08:00",
   "status": "confirmed",
   "customer": {
      "id": 65144,
  "quantity": 2,
      "price": 25.00,
      "subtotal": 50.00
   }]
}
```

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

```
curl -d "$(cat test_event.json)" https://abcdefghij.execute-api.us-
east-2.amazonaws.com/Prod/checkout
```

The API returns the following empty response, indicating a successful execution:

```
{ }
```

Step 3: Verifying Amazon SNS application and pipeline performance

Step 1: Verifying the execution of the sample checkout pipeline

1. Sign in to the Amazon DynamoDB console.
2. On the navigation panel, choose **Tables**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutTable`.
4. On the table details page, choose **Items** and then choose the created item.

   The stored attributes are displayed.

Step 2: Verifying the execution of the event storage and backup pipeline

1. Sign in to the Amazon S3 console.
2. On the navigation panel, choose **Buckets**.
3. Search for `serverlessrepo-fork-example` and then choose `CheckoutBucket`.
4. Navigate the directory hierarchy until you find a file with the extension `.gz`.
5. To download the file, choose **Actions**, **Open**.
6. The pipeline is configured with a Lambda function that sanitizes credit card information for compliance reasons.

   To verify that the stored JSON payload doesn't contain any credit card information, decompress the file.

Step 3: Verifying the execution of the event search and analytics pipeline

1. Sign in to the OpenSearch Service console.
2. On the navigation panel, under **My domains**, choose the domain prefixed with `serverl-analyt`.
3. The pipeline is configured with an Amazon SNS subscription filter policy that sets a numeric matching condition.

   To verify that the event is indexed because it refers to an order whose value is higher than USD $100, on the **serverl-analyt-*`abcdefgh1ijk`*** page, choose **Indices**, **checkout_events**.

Step 4: Verifying the execution of the event replay pipeline

1. Sign in to the Amazon SQS console.
2. In the list of queues, search for `serverlessrepo-fork-example` and choose `ReplayQueue`.
3. Choose **Send and receive messages**.
4. In the **Send and receive messages in fork-example-ecommerce-*`my-app`*...ReplayP-ReplayQueue-*`123ABCD4E5F6`*** dialog box, choose **Poll for messages**.
5. To verify that the event is enqueued, choose **More Details** next to the message that appears in the queue.

Step 4: Simulating an issue and replay events for recovery

Step 1: Enable the simulated issue and send a second API request

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

```
{
    "id": 9917,
    "date": "2019-03-26T21:11:10-08:00",
    "status": "confirmed",
    "customer": {
       "id": 56999,
"quantity": 1,
       "price": 75.00,
       "subtotal": 75.00
    }]
 }
```

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

```
curl -d "$(cat test_event_2.json)" https://abcdefghij.execute-api.us-
east-2.amazonaws.com/Prod/checkout
```

   The API returns the following empty response, indicating a successful execution:

```
{ }
```

Step 2: Verify simulated data corruption

1. Sign in to the Amazon DynamoDB console.
2. On the navigation panel, choose **Tables**.
3. Search for `serverlessrepo-fork-example` and choose `CheckoutTable`.

4. On the table details page, choose **Items** and then choose the created item.

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

Step 3: Disable the simulated issue

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

Step 4: Enable replay to recover from the issue

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

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

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

Subscribing AWS Event Fork Pipelines to an Amazon SNS topic

To accelerate the development of your event-driven applications, you can subscribe event-handling pipelines—powered by AWS Event Fork Pipelines—to Amazon SNS topics. AWS Event Fork Pipelines is a suite of open-source nested applications, based on the AWS Serverless Application Model (AWS SAM), which you can deploy directly from the AWS Event Fork Pipelines suite (choose

Show apps that create custom IAM roles or resource policies) into your AWS account. For more information, see How AWS Event Fork Pipelines works.

This section show how you can use the AWS Management Console to deploy a pipeline and then subscribe AWS Event Fork Pipelines to an Amazon SNS topic. Before you begin, create an Amazon SNS topic.

To delete the resources that comprise a pipeline, find the pipeline on the **Applications** page of on the AWS Lambda console, expand the **SAM template section**, choose **CloudFormation stack**, and then choose **Other Actions**, **Delete Stack**.

Deploying and subscribing the Event Storage and Backup Pipeline to Amazon SNS

> For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Data Firehose. You can subscribe Firehose delivery streams to SNS topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see Fanout to Firehose delivery streams.

This page shows how to deploy the Event Storage and Backup Pipeline and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Storage and Backup Pipeline, including the following:

- Amazon SQS queue
- Lambda function
- Firehose delivery stream
- Amazon S3 backup bucket

For more information about configuring a stream with an Amazon S3 bucket as a destination, see `S3DestinationConfiguration` in the *Amazon Data Firehose API Reference*.

For more information about transforming events and about configuring event buffering, event compression, and event encryption, see Creating an Delivery Stream in the *Amazon Data Firehose Developer Guide*.

For more information about filtering events, see Amazon SNS subscription filter policies in this guide.

1. Sign in to the AWS Lambda console.
2. On the navigation panel, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, do the following:

    a. Choose **Browse serverless app repository**, **Public applications**, **Show apps that create custom IAM roles or resource policies**.

    b. Search for `fork-event-storage-backup-pipeline` and then choose the application.

4. On the **fork-event-storage-backup-pipeline** page, do the following:

    a. In the **Application settings** section, enter an **Application name** (for example, `my-app-backup`).

    > **Note**
    >
    > - For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

    b. (Optional) For **BucketArn**, enter the ARN of the Amazon S3 bucket into which incoming events are loaded. If you don't enter a value, a new Amazon S3 bucket is created in your AWS account.

    c. (Optional) For **DataTransformationFunctionArn**, enter the ARN of the Lambda function through which the incoming events are transformed. If you don't enter a value, data transformation is disabled.

    d. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:

    - `DEBUG`
    - `ERROR`
    - `INFO` (default)

   - `WARNING`

e. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.

f. (Optional) For **StreamBufferingIntervalInSeconds** and **StreamBufferingSizeInMBs**, enter the values for configuring the buffering of incoming events. If you don't enter any values, 300 seconds and 5 MB are used.

g. (Optional) Enter one of the following **StreamCompressionFormat** settings for compressing incoming events:

   - `GZIP`
   - `SNAPPY`
   - `UNCOMPRESSED` (default)
   - `ZIP`

h. (Optional) For **StreamPrefix**, enter the string prefix to name files stored in the Amazon S3 backup bucket. If you don't enter a value, no prefix is used.

i. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are indexed in the OpenSearch Service index. If you don't enter a value, no filtering is used (all events are indexed).

j. (Optional) For **SubscriptionFilterPolicyScope**, enter the string `MessageBody` or `MessageAttributes` to enable payload-based or attribute-based message filtering.

k. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status for *`my-app`*** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE_IN_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are stored in the Amazon S3 backup bucket provisioned by the Event Storage and Backup pipeline automatically.

Deploying and subscribing the Event Search and Analytics Pipeline to Amazon SNS

> For event archiving and analytics, Amazon SNS now recommends using its native integration with Amazon Data Firehose. You can subscribe Firehose delivery streams to SNS topics, which allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more. Using Amazon SNS with Firehose delivery streams is a fully-managed and codeless solution that doesn't require you to use AWS Lambda functions. For more information, see Fanout to Firehose delivery streams.

This page shows how to deploy the Event Search and Analytics Pipeline and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Search and Analytics Pipeline, including the following:

- Amazon SQS queue
- Lambda function
- Firehose delivery stream
- Amazon OpenSearch Service domain
- Amazon S3 dead-letter bucket

For more information about configuring a stream with an index as a destination, see `ElasticsearchDestinationConfiguration` in the *Amazon Data Firehose API Reference*.

For more information about transforming events and about configuring event buffering, event compression, and event encryption, see Creating an Delivery Stream in the *Amazon Data Firehose Developer Guide*.

For more information about filtering events, see Amazon SNS subscription filter policies in this guide.

1. Sign in to the AWS Lambda console.
2. On the navigation panel, choose **Functions** and then choose **Create function**.

3. On the **Create function** page, do the following:

   a. Choose **Browse serverless app repository**, **Public applications**, **Show apps that create custom IAM roles or resource policies**.

   b. Search for `fork-event-search-analytics-pipeline` and then choose the application.

4. On the **fork-event-search-analytics-pipeline** page, do the following:

   a. In the **Application settings** section, enter an **Application name** (for example, `my-app-search`).

   > **Note**
   >
   > For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

   b. (Optional) For **DataTransformationFunctionArn**, enter the ARN of the Lambda function used for transforming incoming events. If you don't enter a value, data transformation is disabled.

   c. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:

   - `DEBUG`
   - `ERROR`
   - `INFO` (default)
   - `WARNING`

   d. (Optional) For **SearchDomainArn**, enter the ARN of the OpenSearch Service domain, a cluster that configures the needed compute and storage functionality. If you don't enter a value, a new domain is created with the default configuration.

   e. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.

   f. For **SearchIndexName**, enter the name of the OpenSearch Service index for event search and analytics.

Note

The following quotas apply to index names:

- Can't include uppercase letters
- Can't include the following characters: `\ / * ? " < > | ` , #`
- Can't begin with the following characters: `- + _`
- Can't be the following: `. ..`
- Can't be longer than 80 characters
- Can't be longer than 255 bytes
- Can't contain a colon (from OpenSearch Service 7.0)

g. (Optional) Enter one of the following **SearchIndexRotationPeriod** settings for the rotation period of the OpenSearch Service index:

- `NoRotation` (default)
- `OneDay`
- `OneHour`
- `OneMonth`
- `OneWeek`

Index rotation appends a timestamp to the index name, facilitating the expiration of old data.

h. For **SearchTypeName**, enter the name of the OpenSearch Service type for organizing the events in an index.

Note

- OpenSearch Service type names can contain any character (except null bytes) but can't begin with `_`.
- For OpenSearch Service 6.x, there can be only one type per index. If you specify a new type for an existing index that already has another type, Firehose returns a runtime error.

i. (Optional) For **StreamBufferingIntervalInSeconds** and **StreamBufferingSizeInMBs**, enter the values for configuring the buffering of incoming events. If you don't enter any values, 300 seconds and 5 MB are used.

j. (Optional) Enter one of the following **StreamCompressionFormat** settings for compressing incoming events:

   - `GZIP`
   - `SNAPPY`
   - `UNCOMPRESSED` (default)
   - `ZIP`

k. (Optional) For **StreamPrefix**, enter the string prefix to name files stored in the Amazon S3 dead-letter bucket. If you don't enter a value, no prefix is used.

l. (Optional) For **StreamRetryDurationInSecons**, enter the retry duration for cases when Firehose can't index events in the OpenSearch Service index. If you don't enter a value, then 300 seconds is used.

m. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are indexed in the OpenSearch Service index. If you don't enter a value, no filtering is used (all events are indexed).

n. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status for *`my-app-search`*** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE_IN_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are indexed in the OpenSearch Service index provisioned by the Event Search and Analytics pipeline automatically. If the pipeline can't index an event, it stores it in a Amazon S3 dead-letter bucket.

Deploying the Event Replay Pipeline with Amazon SNS integration

This page shows how to deploy the Event Replay Pipeline and subscribe it to an Amazon SNS topic. This process automatically turns the AWS SAM template associated with the pipeline into an AWS CloudFormation stack, and then deploys the stack into your AWS account. This process also creates and configures the set of resources that comprise the Event Replay Pipeline, including an Amazon SQS queue and a Lambda function.

For more information about filtering events, see Amazon SNS subscription filter policies in this guide.

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

   > **Note**
   >
   > For each deployment, the application name must be unique. If you reuse an application name, the deployment will update only the previously deployed AWS CloudFormation stack (rather than create a new one).

   b. (Optional) Enter one of the following **LogLevel** settings for the execution of your application's Lambda function:
      - `DEBUG`
      - `ERROR`
      - `INFO` (default)
      - `WARNING`

c. (Optional) For **ReplayQueueRetentionPeriodInSeconds**, enter the amount of time, in seconds, for which the Amazon SQS replay queue keeps the message. If you don't enter a value, 1,209,600 seconds (14 days) is used.

d. For **TopicArn**, enter the ARN of the Amazon SNS topic to which this instance of the fork pipeline is to be subscribed.

e. For **DestinationQueueName**, enter the name of the Amazon SQS queue to which the Lambda replay function forwards messages.

f. (Optional) For **SubscriptionFilterPolicy**, enter the Amazon SNS subscription filter policy, in JSON format, to be used for filtering incoming events. The filter policy decides which events are buffered for replay. If you don't enter a value, no filtering is used (all events are buffered for replay).

g. Choose **I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.** and then choose **Deploy**.

On the **Deployment status for *`my-app-replay`*** page, Lambda displays the **Your application is being deployed** status.

In the **Resources** section, AWS CloudFormation begins to create the stack and displays the **CREATE_IN_PROGRESS** status for each resource. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

When the deployment is complete, Lambda displays the **Your application has been deployed** status.

Messages published to your Amazon SNS topic are buffered for replay in the Amazon SQS queue provisioned by the Event Replay Pipeline automatically.

> **Note**
>
> By default, replay is disabled. To enable replay, navigate to the function's page on the Lambda console, expand the **Designer** section, choose the **SQS** tile and then, in the **SQS** section, choose **Enabled**.

Using Amazon EventBridge Scheduler with Amazon SNS

Amazon EventBridge Scheduler is a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. With EventBridge Scheduler, you can create schedules using Cron and rate expressions for recurring patterns, or configure one-time invocations. You can set up flexible time windows for delivery, define retry limits, and set the maximum retention time for failed API invocations.

This page explains how to use EventBridge Scheduler to publish a message from an Amazon SNS topic on a schedule.

Setting-up the execution role

When you create a new schedule, EventBridge Scheduler must have permission to invoke its target API operation on your behalf. You grant these permissions to EventBridge Scheduler using an *execution role*. The permission policy you attach to your schedule's execution role defines the required permissions. These permissions depend on the target API you want EventBridge Scheduler to invoke.

When you use the EventBridge Scheduler console to create a schedule, as in the following procedure, EventBridge Scheduler automatically sets up an execution role based on your selected target. If you want to create a schedule using one of the EventBridge Scheduler SDKs, the AWS CLI, or AWS CloudFormation, you must have an existing execution role that grants the permissions EventBridge Scheduler requires to invoke a target. For more information about manually setting-up an execution role for your schedule, see Setting-up an execution role in the *EventBridge Scheduler User Guide*.

Create a schedule

To create a schedule by using the console

1. Open the Amazon EventBridge Scheduler console at https://console.aws.amazon.com/scheduler/home.
2. On the **Schedules** page, choose **Create schedule**.
3. On the **Specify schedule detail** page, in the **Schedule name and description** section, do the following:

    a. For **Schedule name**, enter a name for your schedule. For example, **`MyTestSchedule`**.

b. (Optional) For **Description**, enter a description for your schedule. For example, **`My first schedule`**.

c. For **Schedule group**, choose a schedule group from the dropdown list. If you don't have a group, choose **default**. To create a schedule group, choose **create your own schedule**.

   You use schedule groups to add tags to groups of schedules.

4. - Choose your schedule options.

Occurrence	Do this...	
One-time schedule A one-time schedule invokes a target only once at the date and time that you specify.	For **Date and time**, do the following: • Enter a valid date in `YYYY/MM/DD` format. • Enter a timestamp in 24-hour hh`:mm` format. • For **Timezone**, choose the timezone.	
Recurring schedule A recurring schedule invokes a target at a rate that you specify using a **cron** expression or rate expression.	a. For **Schedule type**, do one of the following: • To use a cron expression to define the schedule, choose **Cron-based schedule** and enter the cron expression. • To use a rate expression to define the schedule, choose **Rate-based schedule** and enter the rate expression.	

Occurrence	Do this...	
	For more information about cron and rate expressions, see Schedule types on EventBridge Scheduler in the *Amazon EventBridge Scheduler User Guide*. b. For **Flexible time window**, choose **Off** to turn off the option, or choose one of the pre-defined time windows. For example, if you choose **15 minutes** and you set a recurring schedule to invoke its target once every hour, the schedule runs within 15 minutes after the start of every hour.	

5. (Optional) If you chose **Recurring schedule** in the previous step, in the **Timeframe** section, do the following:

   a. For **Timezone**, choose a timezone.

   b. For **Start date and time**, enter a valid date in `YYYY/MM/DD` format, and then specify a timestamp in 24-hour `hh:mm` format.

   c. For **End date and time**, enter a valid date in `YYYY/MM/DD` format, and then specify a timestamp in 24-hour `hh:mm` format.

6. Choose **Next**.

7. On the **Select target** page, choose the AWS API operation that EventBridge Scheduler invokes:

   a. Choose **Amazon SNS Publish**.

   b. In the **Publish** section, select an SNS topic or choose **Create new SNS topic**.

   c. (Optional) Enter a JSON payload. If you don't enter a payload, EventBridge Scheduler uses an empty event to invoke the function.

8. Choose **Next**.

9. On the **Settings** page, do the following:

   a. To turn on the schedule, under **Schedule state**, toggle **Enable schedule**.

   b. To configure a retry policy for your schedule, under **Retry policy and dead-letter queue (DLQ)**, do the following:

      - Toggle **Retry**.
      - For **Maximum age of event**, enter the maximum **hour(s)** and **min(s)** that EventBridge Scheduler must keep an unprocessed event.
      - The maximum time is 24 hours.
      - For **Maximum retries**, enter the maximum number of times EventBridge Scheduler retries the schedule if the target returns an error.

        The maximum value is 185 retries.

      With retry policies, if a schedule fails to invoke its target, EventBridge Scheduler re-runs the schedule. If configured, you must set the maximum retention time and retries for the schedule.

   c. Choose where EventBridge Scheduler stores undelivered events.

Dead-letter queue (DLQ) option	Do this...	
Don't store	Choose **None**.	
Store the event in the same AWS account where you're creating the schedule	a. Choose **Select an Amazon SQS queue in my AWS account as a DLQ**.	

Dead-letter queue (DLQ) option	Do this...
	b. Choose the Amazon Resource Name (ARN) of the Amazon SQS queue.
Store the event in a different AWS account from where you're creating the schedule	a. Choose **Specify an Amazon SQS queue in other AWS accounts as a DLQ**. b. Enter the Amazon Resource Name (ARN) of the Amazon SQS queue.

d. To use a customer managed key to encrypt your target input, under **Encryption**, choose **Customize encryption settings (advanced)**.

   If you choose this option, enter an existing KMS key ARN or choose **Create an AWS KMS key** to navigate to the AWS KMS console. For more information about how EventBridge Scheduler encrypts your data at rest, see Encryption at rest in the *Amazon EventBridge Scheduler User Guide*.

e. To have EventBridge Scheduler create a new execution role for you, choose **Create new role for this schedule**. Then, enter a name for **Role name**. If you choose this option, EventBridge Scheduler attaches the required permissions necessary for your templated target to the role.

10. Choose **Next**.
11. In the **Review and create schedule** page, review the details of your schedule. In each section, choose **Edit** to go back to that step and edit its details.
12. Choose **Create schedule**.

    You can view a list of your new and existing schedules on the **Schedules** page. Under the **Status** column, verify that your new schedule is **Enabled**.

Related resources

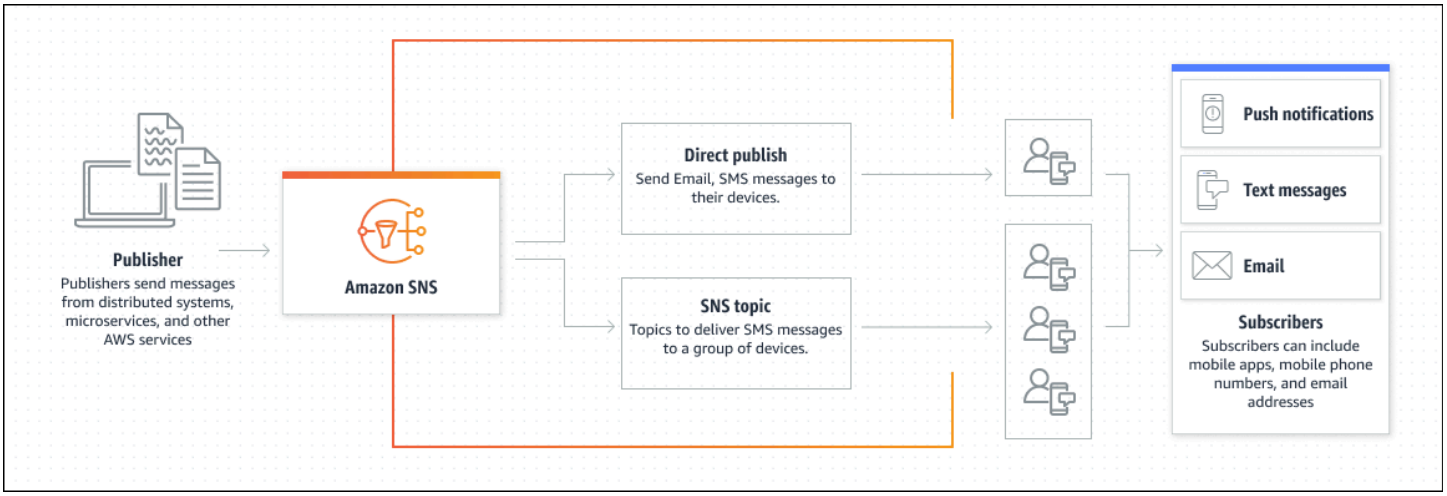
For more information about EventBridge Scheduler, see the following:

- EventBridge Scheduler User Guide
- EventBridge Scheduler API Reference
- EventBridge Scheduler Pricing

Using Amazon SNS for application-to-person messaging

Amazon SNS application-to-person (A2P) messaging lets you to deliver notifications and alerts directly to your customers' mobile devices through SMS (Short Message Service). Using this feature, you can send push notifications to mobile apps, text messages to mobile phone numbers, and plain-text emails to email addresses. Additionally, you have the flexibility to distribute messages by using topics to reach multiple recipients, or publish messages directly to individual mobile endpoints for personalized communication.

This topic explains how to use Amazon SNS for user notifications with subscribers such as mobile applications, mobile phone numbers, and email addresses.


Publisher
Publishers send messages from distributed systems, microservices, and other AWS services
Amazon SNS
Direct publish
Send Email, SMS messages to their devices.
SNS topic
Topics to deliver SMS messages to a group of devices.
Push notifications
Text messages
Email
Subscribers
Subscribers can include mobile apps, mobile phone numbers, and email addresses

Mobile text messaging with Amazon SNS

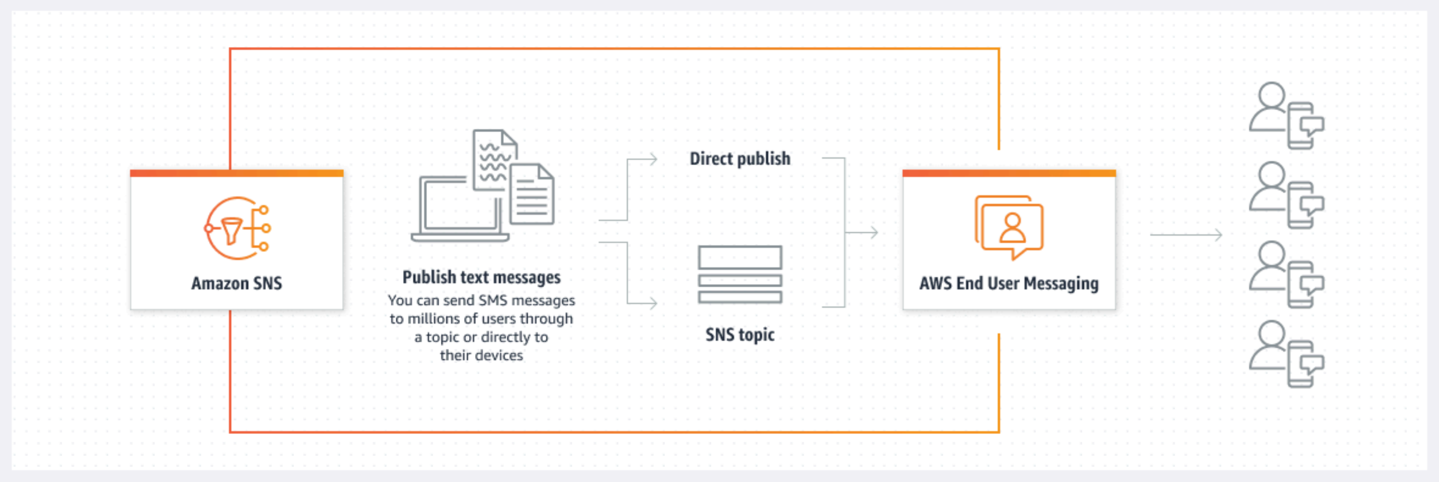
Important

The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

Amazon SNS mobile text messaging (SMS) is designed to facilitate message delivery to various platforms, such as web, mobile, and business applications that support SMS. Users can send messages to one or multiple phone numbers by subscribing them to a topic, simplifying the distribution process.

Amazon SNS messages are delivered by AWS End User Messaging SMS, which ensures reliable message transmission. Within Amazon SNS APIs, you can set various properties such as message types (promotional or transactional), monthly spending limits, opt-out lists, and message delivery optimization.

AWS End User Messaging SMS handles the transmission of messages to the destination phone number through its global SMS supply network. It manages the routing, delivery status, and any required compliance with regional regulations. To access additional SMS features such as granular permissions, phone pools, configurations sets, SMS simulator, and country rule, see the AWS End User Messaging SMS User Guide.


Amazon SNS
Publish text messages
You can send SMS messages to millions of users through a topic or directly to their devices
Direct publish
SNS topic
AWS End User Messaging

The following key features help you send Amazon SNS SMS messages that are scalable and easily extensible:

Customize message preferences

Customize SMS deliveries for your AWS account by setting up SMS preferences based on your budget and use case. For example, you can choose whether your messages prioritize cost efficiency or reliable delivery.

Set spending quotas

Tailor your SMS deliveries by specifying spending quotas or for individual message deliveries and monthly spending quotas for your AWS account. Where required by local laws and regulations (such as the US and Canada), SMS recipients can opt-out, which means that they choose to stop receiving SMS messages from your AWS account. After a recipient opts-out of receiving messages, you can, with limitations, opt-in the phone number again so that you can resume sending messages.

Send SMS messages globally

Amazon SNS supports SMS messaging in multiple regions, allowing you to send messages to over 240 countries and regions.

How does Amazon SNS deliver my SMS messages?

When you request Amazon SNS to send SMS on your behalf, the messages are dispatched using AWS End User Messaging SMS. The integration between Amazon SNS and AWS End User Messaging SMS offers the following benefits:

IAM policies

You can leverage IAM and resource policies to control and distribute access to your SMS resources across other AWS services and regions.

AWS End User Messaging SMS configurations

All origination ID related configurations (creation, configuration updating, provisioning new origination IDs, changing registration templates) use AWS End User Messaging SMS.

AWS End User Messaging SMS billing

All SMS billing is done though AWS End User Messaging SMS. You can consolidate your AWS spend for your SMS workloads, while procuring and managing your SMS resources in one central location.

Getting started with Amazon SNS SMS

> ⚠ **Important**
>
> The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

This topic guides you through managing your SMS sandbox and configuring IAM and resource-based policies to grant Amazon SNS the necessary permissions for accessing and utilizing the AWS End User Messaging SMS APIs.

Prerequisites

Amazon SNS recommends updating your IAM policy to include the following actions to ensure comprehensive control and visibility over your Amazon SNS resources:

- `AmazonSNSFullAccess`
- `AmazonSNSReadOnly`

Using the Amazon SNS SMS sandbox

Newly created Amazon SNS SMS accounts are automatically placed into the SMS sandbox to ensure the security of both AWS customers and recipients by mitigating the risk of fraud and abuse. This environment serves as a secure space for testing and development purposes. While operating within the SMS sandbox, you have access to all Amazon SNS features but are subject to certain limitations:

- You can only send SMS messages to verified destination phone numbers.
- You can have up to 10 verified destination phone numbers.
- You can delete destination phone numbers only after a minimum of 24 hours have passed since verification, or the last verification attempt.

Once your account transitions out of the sandbox, these restrictions are removed, and you can send SMS messages to any recipient.

First steps

New Amazon SNS SMS accounts are placed into an SMS sandbox. Use the following steps to create and manage phone numbers in your sandbox, create origination numbers and sender IDs, and register your company.

1. Add a **destination phone number** to the SMS sandbox. For details on adding, managing and moving phone numbers out of the Amazon SNS SMS sandbox, see Adding and verifying phone numbers in the Amazon SNS SMS sandbox.
2. Create an **origination identity** that your recipients see on their devices when you send them an SMS message. To learn more about origination identities, including the different types you can use, see the Origination identities for Amazon SNS SMS messages documentation.

3. **Register** your company. Some countries require you to register your company's identity to be able to purchase phone numbers or sender IDs and review the messages you send to recipients in their country. For information on which countries require registration, see Supported countries and regions for SMS messaging with AWS End User Messaging SMS in the *AWS End User Messaging SMS User Guide*.
4. **Send** your messages to a topic or mobile phone. For more information, see Sending SMS messages using Amazon SNS.

Adding and verifying phone numbers in the Amazon SNS SMS sandbox

Before you can start sending SMS messages from your AWS account while in the SMS sandbox, you must complete the following setup steps. This ensures that your account is ready for SMS messaging and that your destination phone numbers are properly verified.

1. Create an **origination ID**. Similar to accounts outside of the SMS sandbox, an origination ID is necessary before you can send SMS messages to recipients in some countries or regions.
2. Add the **destination phone numbers** you want to send messages to within the SMS sandbox.
3. Verify the **phone numbers** to ensure that the destination phone numbers are valid for use in your SMS messages.

Add and verify destination phone numbers

1. Sign in to the Amazon SNS console.
2. In the console menu, choose a region that supports SMS messaging.
3. In the navigation pane, choose **Text messaging (SMS)**.
4. In the **Sandbox destination phone numbers** section, select **Add phone number**.
5. Under **Destination details**, provide the following information, and then select **Add phone number**:
   - **Country code** and **phone number** of the destination.
   - The **language** you want the verification message to be sent in.
6. After adding the phone number, Amazon SNS will send an OTP to the provided destination phone number. This OTP is required for verification.
7. You will receive the OTP as a standard SMS message on the **destination phone number** you provided.

   - If you don’t receive the OTP within 15 minutes, select **Resend verification** code in the Amazon SNS console.
   - You can resend the OTP up to five times in a 24-hour period.

8. Once you receive the OTP, enter it in the **Verification code** box and select **Verify phone number**.
9. Check the **verification status**.

   - After successfully verifying the phone number, the phone number and its verification status will appear in the **Sandbox destination phone numbers** section.
   - If the status is **Pending**, the verification was unsuccessful. This may happen if, for example, you didn’t enter the country code correctly.
   - You can only delete pending or verified phone numbers after 24 hours or more have passed since the last verification attempt.

10. If you wish to use the same destination phone number in other regions, **repeat** the previous steps for each region where you intend to use it.

Troubleshooting non-receipt of an OTP text

Troubleshoot common problems that may prevent a phone number from receiving OTP texts.

- **Amazon SNS SMS spending limit:** If your AWS account has exceeded the spending limit for sending SMS messages, further messages, including OTP texts, might not be delivered until the limit is increased or the billing issue is resolved.
- **Phone number not opted in for SMS notifications:** In some countries or regions, recipients must opt in to receive SMS messages from short codes, which are commonly used for OTP texts. If the recipient's phone number is not opted in, they will not receive the OTP text.
- **Carrier restrictions or filtering:** Some mobile carriers may have restrictions or filtering mechanisms in place that prevent delivery of certain types of SMS messages, including OTP texts. This could be due to security policies or anti-spam measures implemented by the carrier.
- **Invalid or incorrect phone number:** If the phone number provided by the recipient is incorrect or invalid, the OTP text will not be delivered.
- **Network issues:** Temporary network issues or outages could prevent the delivery of SMS messages, including OTP texts, to the recipient's phone.

- **Delayed delivery:** In some cases, SMS messages may experience delays in delivery due to network congestion or other factors. The OTP text may eventually be delivered, but it could be delayed beyond the expected timeframe.
- **Account suspension or termination:** If there are issues with your AWS account, such as non-payment or violation of AWS terms of service, Amazon SNS messaging capabilities, including OTP texts, may be suspended or terminated.

Deleting phone numbers from the Amazon SNS SMS sandbox

You can delete both pending and verified destination phone numbers from the SMS sandbox.

> ⚠ **Important**
>
> You can only delete a phone number 24 hours after verifying the phone number, or 24 hours after your last verification attempt.

To delete destination phone numbers from the SMS sandbox

1. Sign in to the Amazon SNS console.
2. In the console menu, choose a region that supports SMS messaging where you added a destination phone number.
3. In the navigation pane, select **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, navigate to the **Sandbox destination phone numbers** section.
5. Choose the specific phone number you want to delete, and then choose **Delete phone number**.
6. To confirm that you want to delete the phone number, enter **`delete me`**, and then choose **Delete**.

   Ensure that 24 hours or more have passed since you verified or attempted to verify the destination phone number before proceeding with the deletion.
7. Repeat these steps in each Region where you added the destination phone number and no longer plan to use it.

Moving out of the Amazon SNS SMS sandbox

Moving your AWS account out of the SMS sandbox requires that you first add, verify, and test destination phone numbers. After doing this, create a case with AWS Support.

To request that your AWS account is moved out of the SMS sandbox

1. **Verify phone numbers**

   a. While your AWS account is in the SMS sandbox, open the Amazon SNS console.

   b. In the navigation pane, under Mobile, choose **Text messaging (SMS)**.

   c. In the Sandbox destination phone numbers section, add and verify one or more destination phone numbers. This verification ensures you can successfully send and receive messages.

2. **Test SMS publishing**

   - Confirm that you are able to send and receive messages to at least one verified phone number. For more detailed instructions on how to publish SMS messages, see Publishing SMS messages to a mobile phone using Amazon SNS.

3. **Initiate sandbox edit**

   - On the Amazon SNS console's **Mobile text messaging (SMS)** page, under **Account information**, choose **Exit SMS sandbox**. This action redirects you to the Amazon Support Center and automatically creates a support case with the **Service quota increase** option selected.

4. **Fill out the form**

   - In the support form under **Service quota increase**, do the following:

     i. Choose choose **SNS Text Messaging** as the service.

     ii. Provide the **website URL** or **app name** from which you intend to send SMS messages.

     iii. Specify the type of messages you will send: **One Time Password**, **Promotional**, or **Transactional**.

     iv. Choose the **AWS Region** from which you will send SMS messages.

     v. List the **countries** or **regions** where you plan to send SMS messages.

     vi. Describe how your customers **opt-in to receive messages**.

     vii. Include any **message templates** you intend to use.

5. **Specify quota and Region**

   - Under **Requests**, do the following:

     i. Choose the **AWS Region** where you want to move your AWS account.

     ii. Choose **General Limits** for **Resource Type**.

     iii. Choose **Exit SMS Sandbox** for **Quota**.

     iv. (Optional) To request additional increases or other adjustments, choose **Add another request** and specify the necessary details.

     v. For **New quota value**, enter the **limit** in USD you are requesting.

6. **Additional details**

   a. In the **Case description**, provide any additional details relevant to your request.

   b. Under **Contact options**, choose your **preferred contact language**.

7. **Submit the request**

   - Choose **Submit** to send your request to Support.

The Support team provides an initial response to your request within 24 hours.

To prevent our systems from being used to send unsolicited or malicious content, we consider each request carefully. If we can, we will grant your request within this 24-hour period. However, if we need additional information from you, it might take longer to resolve your request.

If your use case doesn't align with our policies, we might be unable to grant your request.

Origination identities for Amazon SNS SMS messages

> ⚠ **Important**
>
> The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

Origination identities for SMS messages are identifiers used to represent the sender of an SMS message. You can identify yourself to your recipients using the following types of originating identities:

Origination numbers

A numeric string that identifies an SMS message sender's phone number. There are several types of origination numbers, including long codes (standard phone numbers that typically have 10 or more digits), 10 digit long codes (10DLC), toll free numbers (TFN) and short codes (phone numbers that contain between four and seven digits).

Support for origination numbers is not available in countries where local laws require the use of sender IDs instead of origination numbers. When you send an SMS message using an origination number, the recipient's device shows the origination number as the sender's phone number. You can specify different origination numbers by use case.

For additional information, see Phone numbers in the *AWS End User Messaging SMS User Guide*.

> **Tip**
>
> To view a list of all existing origination numbers in your AWS account, in the navigation pane of the Amazon SNS console, choose **Origination numbers**.

Sender IDs

An alphabetic name that identifies the sender of an SMS message. When you send an SMS message using a sender ID, and the recipient is in an area where sender ID authentication is supported, your sender ID appears on the recipient’s device instead of your phone number. A sender ID provides SMS recipients with more information about the sender than a phone number, long code, or short code provides.

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

There's no additional charge for using sender IDs. However, support and requirements for sender ID authentication varies by country. Several major markets (including Canada, China, and the United States) don't support using sender IDs. Some areas require that companies who send SMS messages to individual customers must use a sender ID that's pre-registered with a regulatory agency or industry group.

For additional information, see Sender IDs in the *AWS End User Messaging SMS User Guide*.

Configuring SMS messaging in Amazon SNS

⚠ **Important**

The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

You can use the configurations in Amazon SNS SMS to set SMS preferences to suit your requirements, such as adjusting spending quotas and setting-up delivery status logging. This topic also provides details on how to publish SMS messages to topics using the Amazon SNS console and AWS SDK, efficiently handle quotas, and retrieve detailed statistics on SMS activity.

Sending SMS messages using Amazon SNS

This section describes how to send SMS messages using Amazon SNS, including publishing to a topic, subscribing phone numbers to topics, setting attributes on messages, and publishing directly to mobile phones.

Publishing SMS messages to an Amazon SNS topic

You can publish a single SMS message to many phone numbers at once by subscribing those phone numbers to an Amazon SNS topic. An SNS topic is a communication channel to which you can add subscribers and then publish messages to all of those subscribers. A subscriber receives all messages published to the topic until you cancel the subscription, or until the subscriber opts out of receiving SMS messages from your AWS account.

Sending a message to a topic using the AWS console

To create a topic

Complete the following steps if you don't already have a topic to which you want to send SMS messages.

1. Sign in to the Amazon SNS console.
2. In the console menu, choose a region that supports SMS messaging.
3. In the navigation pane, choose **Topics**.
4. On the **Topics** page, choose **Create topic**.
5. On the **Create topic** page, under **Details**, do the following:

   a. For **Type**, choose **Standard**.

   b. For **Name**, enter a topic name.

   c. (Optional) For **Display name**, enter a custom prefix for your SMS messages. When you send a message to the topic, Amazon SNS prepends the display name followed by a right angle bracket (>) and a space. Display names are not case sensitive, and Amazon SNS converts display names to uppercase characters. For example, if the display name of a topic is `MyTopic` and the message is `Hello World!`, the message appears as:

   ```
   MYTOPIC> Hello World!
   ```

6. Choose **Create topic**. The topic's name and Amazon Resource Name (ARN) appear on the **Topics** page.

To create an SMS subscription

You can use subscriptions to send an SMS message to multiple recipients by publishing the message only once to your topic.

> **Note**
>
> When you start using Amazon SNS to send SMS messages, your AWS account is in the *SMS sandbox*. The SMS sandbox provides a safe environment for you to try Amazon SNS features without risking your reputation as an SMS sender. While your account is in the SMS sandbox, you can use all of the features of Amazon SNS, but you can send SMS messages

only to verified destination phone numbers. For more information, see Using the Amazon SNS SMS sandbox.

1. Sign in to the Amazon SNS console.
2. In the navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, under **Details**, do the following:

    a. For **Topic ARN**, enter or choose the Amazon Resource Name (ARN) of the topic to which you want to send SMS messages.

    b. For **Protocol**, choose **SMS**.

    c. For **Endpoint**, enter the phone number that you want to subscribe to your topic.
5. Choose **Create subscription**. The subscription information appears on the **Subscriptions** page.

    To add more phone numbers, repeat these steps. You can also add other types of subscriptions, such as email.

To send a message

When you publish a message to a topic, Amazon SNS attempts to deliver that message to every phone number that is subscribed to the topic.

1. In the Amazon SNS console, on the **Topics** page, choose the name of the topic to which you want to send SMS messages.
2. On the topic details page, choose **Publish message**.
3. On the **Publish message to topic** page, under **Message details**, do the following:

    a. For **Subject**, keep the field blank unless your topic contains email subscriptions and you want to publish to both email and SMS subscriptions. Amazon SNS uses the **Subject** that you enter as the email subject line.

    b. (Optional) For **Time to Live (TTL)**, enter a number of seconds that Amazon SNS has to send your SMS message to any mobile application endpoint subscribers.
4. Under **Message body**, do the following:

a. For **Message structure**, choose **Identical payload for all delivery protocols** to send the same message to all protocol types subscribed to your topic. Or, choose **Custom payload for each delivery protocol** to customize the message for subscribers of different protocol types. For example, you can enter a default message for phone number subscribers and a custom message for email subscribers.

b. For **Message body to send to the endpoint**, enter your message, or your custom messages per delivery protocol.

   If your topic has a display name, Amazon SNS adds it to the message, which increases the message length. The display name length is the number of characters in the name plus two characters for the right angle bracket (>) and the space that Amazon SNS adds.

   For information about the size quotas for SMS messages, see Publishing SMS messages to a mobile phone using Amazon SNS.

5. (Optional) For **Message attributes**, add message metadata such as timestamps, signatures, and IDs.

6. Choose **Publish message**. Amazon SNS sends the SMS message and displays a success message.

Sending a message to a topic using the AWS SDKs

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

The following code example shows how to:

- Create an Amazon SNS topic.
- Subscribe phone numbers to the topic.
- Publish SMS messages to the topic so that all subscribed phone numbers receive the message at once.

Java

SDK for Java 2.x

Note

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

Create a topic and return its ARN.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicName>

                Where:
                   topicName - The name of the topic to create (for example,
 mytopic).

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
        String topicName = args[0];
        System.out.println("Creating a topic with name: " + topicName);
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnVal = createSNSTopic(snsClient, topicName);
        System.out.println("The topic ARN is" + arnVal);
        snsClient.close();
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                    .name(topicName)
                    .build();

            result = snsClient.createTopic(request);
            return result.topicArn();

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

Subscribe an endpoint to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeTextSMS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn> <phoneNumber>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   phoneNumber - A mobile phone number that receives
 notifications (for example, +1XXX5550100).
                """;

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subTextSNS(snsClient, topicArn, phoneNumber);
        snsClient.close();
    }

    public static void subTextSNS(SnsClient snsClient, String topicArn, String
 phoneNumber) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("sms")
                    .endpoint(phoneNumber)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN: " + result.subscriptionArn() +
 "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

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

Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }
    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
 String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                    .attributes(attributes)
                    .build();

            SetSmsAttributesResponse result =
 snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
 Status was "
                    + result.sdkHttpResponse().statusCode());

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

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <message> <phoneNumber>

                Where:
                   message - The message text to send.
                   phoneNumber - The mobile phone number to which a message is
 sent (for example, +1XXX5550100).\s
                """;

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

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
 phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .phoneNumber(phoneNumber)
                    .build();

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

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

Publishing SMS messages to a mobile phone using Amazon SNS

You can use Amazon SNS to send SMS messages directly to a mobile phone without subscribing the phone number to an Amazon SNS topic.

> **Note**
>
> Subscribing phone numbers to a topic is useful if you want to send one message to multiple phone numbers at once. For instructions on publishing an SMS message to a topic, see Publishing SMS messages to an Amazon SNS topic.

When you send a message, you can control whether the message is optimized for cost or reliable delivery. You can also specify a sender ID or origination number. If you send the message programmatically using the Amazon SNS API or the AWS SDKs, you can specify a maximum price for the message delivery.

Each SMS message can contain up to 140 bytes, and the character quota depends on the encoding scheme. For example, an SMS message can contain:

- 160 GSM characters
- 140 ASCII characters
- 70 UCS-2 characters

If you publish a message that exceeds the size quota, Amazon SNS sends it as multiple messages, each fitting within the size quota. Messages are not cut off in the middle of a word, but instead on whole-word boundaries. The total size quota for a single SMS publish action is 1,600 bytes.

When you send an SMS message, you specify the phone number using the E.164 format, a standard phone numbering structure used for international telecommunication. Phone numbers that follow this format can have a maximum of 15 digits along with the prefix of a plus sign (+) and the country code. For example, a US phone number in E.164 format appears as +1XXX5550100.

Sending a message (console)

1. Sign in to the Amazon SNS console.
2. In the console menu, choose a region that supports SMS messaging.
3. In the navigation pane, choose **Text messaging (SMS)**.

4. On the **Mobile text messaging (SMS)** page, choose **Publish text message**.
5. On the **Publish SMS message** page, for **Message type**, choose one of the following:

   - **Promotional** – Non-critical messages, such as marketing messages.
   - **Transactional** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication.

   > **Note**
   >
   > This message-level setting overrides your account-level default message type. You can set an account-level default message type from the **Text messaging preferences** section of the **Mobile text messaging (SMS)** page.

   For pricing information for promotional and transactional messages, see Worldwide SMS Pricing.
6. For **Destination phone number**, enter the phone number to which you want to send the message.
7. For **Message**, enter the message to send.
8. (Optional) Under **Origination identities**, specify how to identify yourself to your recipients:

   - To specify a **Sender ID**, type a custom ID that contains 3-11 alphanumeric characters, including at least one letter and no spaces. The sender ID is displayed as the message sender on the receiving device. For example, you can use your business brand to make the message source easier to recognize.

     Support for sender IDs varies by country and/or region. For example, messages delivered to U.S. phone numbers will not display the sender ID. For the countries and regions that support sender IDs, see Supported countries and regions for SMS messaging with AWS End User Messaging SMS in the *AWS End User Messaging SMS User Guide*.

     If you do not specify a sender ID, one of the following is displayed as the originating identity:

     - In countries that support long codes, the long code is shown.
     - In countries where only sender IDs are supported, *NOTICE* is shown.

This message-level sender ID overrides your default sender ID, which you set on the **Text messaging preferences** page.

- To specify an **Origination number**, enter a string of 5-14 numbers to display as the sender's phone number on the receiver's device. This string must match an origination number that is configured in your AWS account for the destination country. The origination number can be a 10DLC number, toll-free number, person-to-person long code, or short codes. For more information, see Origination identities for Amazon SNS SMS messages.

  If you don't specify an origination number, Amazon SNS selects an origination number to use for the SMS text message, based on your AWS account configuration.

9. If you're sending SMS messages to recipients in India, expand **Country-specific attributes**, and specify the following attributes:

   - **Entity ID** – The entity ID or principal entity (PE) ID for sending SMS messages to recipients in India. This ID is a unique string of 1–50 characters that the Telecom Regulatory Authority of India (TRAI) provides to identify the entity that you registered with the TRAI.
   - **Template ID** – The template ID for sending SMS messages to recipients in India. This ID is a unique, TRAI-provided string of 1–50 characters that identifies the template that you registered with the TRAI. The template ID must be associated with the sender ID that you specified for the message.

   For more information on sending SMS messages to recipients in India, India sender ID registration process in the *AWS End User Messaging SMS User Guide*.

10. Choose **Publish message**.

> **Tip**
>
> To send SMS messages from an origination number, you can also choose **Origination numbers** in the Amazon SNS console navigation panel. Choose an origination number that includes **SMS** in the **Capabilities** column, and then choose **Publish text message**.

Sending a message (AWS SDKs)

To send an SMS message using one of the AWS SDKs, use the API operation in that SDK that corresponds to the `Publish` request in the Amazon SNS API. With this request, you can send an

SMS message directly to a phone number. You can also use the `MessageAttributes` parameter to set values for the following attribute names:

`AWS.SNS.SMS.SenderID`

A custom ID that contains 3–11 alphanumeric characters or hyphen (-) characters, including at least one letter and no spaces. The sender ID appears as the message sender on the receiving device. For example, you can use your business brand to help make the message source easier to recognize.

Support for sender IDs varies by country or region. For example, messages delivered to US phone numbers don't display the sender ID. For a list of the countries or regions that support sender IDs, see Supported countries and regions for SMS messaging with AWS End User Messaging SMS in the *AWS End User Messaging SMS User Guide*.

If you don't specify a sender ID, a long code appears as the sender ID in supported countries or regions. For countries or regions that require an alphabetic sender ID, *NOTICE* appears as the sender ID.

This message-level attribute overrides the account-level attribute `DefaultSenderID`, which you can set using the `SetSMSAttributes` request.

`AWS.MM.SMS.OriginationNumber`

A custom string of 5–14 numbers, which can include an optional leading plus sign (+). This string of numbers appears as the sender's phone number on the receiving device. The string must match an origination number that's configured in your AWS account for the destination country. The origination number can be a 10DLC number, toll-free number, person-to-person (P2P) long code, or short code. For more information, see Phone numbers in the *AWS End User Messaging SMS User Guide*.

If you don't specify an origination number, Amazon SNS chooses an origination number based on your AWS account configuration.

`AWS.SNS.SMS.MaxPrice`

The maximum price in USD that you're willing to spend to send the SMS message. If Amazon SNS determines that sending the message would incur a cost that exceeds your maximum price, it doesn't send the message.

This attribute has no effect if your month-to-date SMS costs have already exceeded the quota set for the `MonthlySpendLimit` attribute. You can set the `MonthlySpendLimit` attribute using the `SetSMSAttributes` request.

If you're sending the message to an Amazon SNS topic, the maximum price applies to each message delivery to each phone number that is subscribed to the topic.

`AWS.SNS.SMS.SMSType`

The type of message that you're sending:

- **`Promotional`** (default) – Non-critical messages, such as marketing messages.
- **`Transactional`** – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication.

This message-level attribute overrides the account-level attribute `DefaultSMSType`, which you can set using the `SetSMSAttributes` request.

`AWS.MM.SMS.EntityId`

This attribute is required only for sending SMS messages to recipients in India.

This is your entity ID or principal entity (PE) ID for sending SMS messages to recipients in India. This ID is a unique string of 1–50 characters that the Telecom Regulatory Authority of India (TRAI) provides to identify the entity that you registered with the TRAI.

`AWS.MM.SMS.TemplateId`

This attribute is required only for sending SMS messages to recipients in India.

This is your template for sending SMS messages to recipients in India. This ID is a unique, TRAI-provided string of 1–50 characters that identifies the template that you registered with the TRAI. The template ID must be associated with the sender ID that you specified for the message.

Sending a message

The following code examples show how to publish SMS messages using Amazon SNS.

.NET

SDK for .NET

Note

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

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

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

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

            // Now actually send the message.
            var request = new PublishRequest
            {
                Message = text,
                PhoneNumber = phoneNum,
            };

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

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

C++

SDK for C++

Note

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

```
/**
 * Publish SMS: use Amazon Simple Notification Service (Amazon SNS) to send an
 SMS text message to a phone number.
 * Note: This requires additional AWS configuration prior to running example.
 *
 *  NOTE: When you start using Amazon SNS to send SMS messages, your AWS account
 is in the SMS sandbox and you can only
 *  use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
latest/dg/sns-sms-sandbox.html.
 *  NOTE: If destination is in the US, you also have an additional restriction
 that you have use a dedicated
 *  origination ID (phone number). You can request an origination number using
 Amazon Pinpoint for a fee.
 *  See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-
origination-numbers-with-amazon-sns/
 *  for more information.
 *
 *  <phone_number_value> input parameter uses E.164 format.
 *  For example, in United States, this input value should be of the form:
 +12223334444
 */

//! Send an SMS text message to a phone number.
/*!
  \param message: The message to publish.
  \param phoneNumber: The phone number of the recipient in E.164 format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::publishSms(const Aws::String &message,
                             const Aws::String &phoneNumber,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetPhoneNumber(phoneNumber);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with message id, '"
```

```
                  << outcome.GetResult().GetMessageId() << "'."
                  << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

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

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <message> <phoneNumber>

                Where:
                   message - The message text to send.
                   phoneNumber - The mobile phone number to which a message is
 sent (for example, +1XXX5550100).\s
                """;

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

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
 phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .phoneNumber(phoneNumber)
                    .build();

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

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

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

Kotlin

SDK for Kotlin

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

```
suspend fun pubTextSMS(
    messageVal: String?,
    phoneNumberVal: String?,
) {
    val request =
        PublishRequest {
            message = messageVal
            phoneNumber = phoneNumberVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

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

PHP

SDK for PHP

Note

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

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

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



/**
 * Sends a text message (SMS message) directly to a phone number using Amazon
 SNS.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

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

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';

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

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

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

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    def publish_text_message(self, phone_number, message):
        """
        Publishes a text message directly to a phone number without need for a
        subscription.

        :param phone_number: The phone number that receives the message. This
 must be
                             in E.164 format. For example, a United States phone
                             number might be +12065550101.
        :param message: The message to send.
        :return: The ID of the message.
        """
        try:
            response = self.sns_resource.meta.client.publish(
                PhoneNumber=phone_number, Message=message
            )
            message_id = response["MessageId"]
            logger.info("Published message to %s.", phone_number)
        except ClientError:
            logger.exception("Couldn't publish message to %s.", phone_number)
            raise
        else:
            return message_id
```

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

Setting SMS messaging preferences in Amazon SNS

Use Amazon SNS to specify preferences for SMS messaging. For example, you can specify whether to optimize deliveries for cost or reliability, your monthly spending limit, how deliveries are logged, and whether to subscribe to daily SMS usage reports.

These preferences take effect for every SMS message that you send from your account, but you can override some of them when you send an individual message. For more information, see Publishing SMS messages to a mobile phone using Amazon SNS.

Setting SMS messaging preferences using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. Choose a region that supports SMS messaging.
3. On the navigation panel, choose **Mobile** and then **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
5. On the **Edit text messaging preferences** page, in the **Details** section, do the following:

    a. For **Default message type**, choose one of the following:

    - **Promotional –** Non-critical messages (for example, marketing). Amazon SNS optimizes message delivery to incur the lowest cost.

- **Transactional** (default) – Critical messages that support customer transactions, such as one-time passcodes for multi-factor authentication. Amazon SNS optimizes message delivery to achieve the highest reliability.

For pricing information for promotional and transactional messages, see Global SMS Pricing.

b. (Optional) For **Account spend limit**, enter the amount (in USD) that you want to spend on SMS messages each calendar month.

> ⚠ **Important**
>
> - By default, the spend quota is set to 1.00 USD. If you want to raise the service quota, submit a request.
> - If the amount set in the console exceeds your service quota, Amazon SNS stops publishing SMS messages.
> - Because Amazon SNS is a distributed system, it stops sending SMS messages within minutes of the spend quota being exceeded. During this interval, if you continue to send SMS messages, you might incur costs that exceed your quota.

6. (Optional) For **Default sender ID**, enter a custom ID, such as your business brand, which is displayed as the sender of the receiving device.

> ⓘ **Note**
>
> Support for sender IDs varies by country.

7. (Optional) Enter the name of the **Amazon S3 bucket name for usage reports**.

> ⓘ **Note**
>
> The Amazon S3 bucket policy must grant write access to Amazon SNS.

8. Choose **Save changes**.

Setting preferences (AWS SDKs)

To set your SMS preferences using one of the AWS SDKs, use the action in that SDK that corresponds to the `SetSMSAttributes` request in the Amazon SNS API. With this request, you assign values to the different SMS attributes, such as your monthly spend quota and your default SMS type (promotional or transactional). For all SMS attributes, see SetSMSAttributes in the *Amazon Simple Notification Service API Reference*.

The following code examples show how to use `SetSMSAttributes`.

C++

SDK for C++

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

How to use Amazon SNS to set the DefaultSMSType attribute.

```
//! Set the default settings for sending SMS messages.
/*!
  \param smsType: The type of SMS message that you will send by default.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::setSMSType(const Aws::String &smsType,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SetSMSAttributesRequest request;
    request.AddAttributes("DefaultSMSType", smsType);

    const Aws::SNS::Model::SetSMSAttributesOutcome outcome =
 snsClient.SetSMSAttributes(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else {
        std::cerr << "Error while setting SMS Type: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

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

CLI

AWS CLI

To set SMS message attributes

The following `set-sms-attributes` example sets the default sender ID for SMS messages to `MyName`.

```
aws sns set-sms-attributes \
    --attributes DefaultSenderID=MyName
```

This command produces no output.

- For API details, see SetSMSAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

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

    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
 String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                    .attributes(attributes)
                    .build();

            SetSmsAttributesResponse result =
 snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
 Status was "
                    + result.sdkHttpResponse().statusCode());

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing
 messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

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

PHP

SDK for PHP

Note

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

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

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

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see SetSMSAttributes in *AWS SDK for PHP API Reference*.

Setting SMS messaging preferences for country-specific delivery

You can manage and control your SMS traffic by sending messages only to specific destination countries. This ensures that your messages are sent only to approved countries, avoiding unwanted SMS charges. The following instructions use Amazon Pinpoint's Protect configuration to specify the countries you want to allow or block.

1. Open the AWS SMS console at https://console.aws.amazon.com/sms-voice/.
2. In the navigation pane, under **Overview**, in the **Quick start** section, choose **Create a protect configuration**.
3. Under **Protect configuration details**, enter a **business-friendly name** for your protect configuration (for example, Allow-Only-AU).
4. Under **SMS country rules**, select the **Region/Country** checkbox to block sending messages to all supported countries.
5. Deselect the checkboxes for the countries where you want to send messages. For example, to allow messages only to Australia, deselect the checkbox for **Australia**.
6. In the **Protect configuration associations** section, under **Association type**, select **Account default**. This will ensure that the AWS End User Messaging SMS Protect configuration affects all messages sent through Amazon SNS, Amazon Cognito, and the Amazon Pinpoint `SendMessages` API call.
7. Choose **Create protect configuration** to save your settings.

   The following confirmation message is displayed:

```
Success Protect configuration protect-abc0123456789 has been created.
```

8. Sign in to the Amazon SNS console.

9. **Publish a message** to one of the blocked countries, such as India.

   The message will not be delivered. You can verify this in the delivery failure logs using CloudWatch. Search for log group **sns/region/AccountID/DirectPublishToPhoneNumber/Failure** for a response similar to the following example:

```
{
"notification": {
"messageId": "bd59a509-XXXX-XXXX-82f8-fbdb8cb68217",
"timestamp": "YYYY-MM-DD XX:XX:XX.XXXX“
},
"delivery": {
"destination": "+91XXXXXXXXXX",
"smsType": "Transactional",
"providerResponse": "Cannot deliver message to the specified destination country",
"dwellTimeMs": 85
},
"status": "FAILURE"
}
```

Managing Amazon SNS phone numbers and subscriptions

Amazon SNS provides several options for managing who receives SMS messages from your account. With a limited frequency, you can opt in phone numbers that have opted out of receiving SMS messages from your account. To stop sending messages to SMS subscriptions, you can remove subscriptions or the topics that publish to them.

Opting out of receiving SMS messages

Where required by local laws and regulations (such as the US and Canada), SMS recipients can use their devices to opt-out by replying to the message with any of the following:

- ARRET (French)
- CANCEL
- END

- OPT-OUT
- OPTOUT
- QUIT
- REMOVE
- STOP
- TD
- UNSUBSCRIBE

To opt-out, the recipient must reply to the same origination number that Amazon SNS used to deliver the message. After opting-out, the recipient will no longer receive SMS messages delivered from your AWS account unless you opt-in the phone number.

If the phone number is subscribed to an Amazon SNS topic, opting-out does not remove the subscription, but SMS messages will fail to deliver to that subscription unless you opt-in the phone number.

Managing phone numbers and subscriptions using the Amazon SNS console

You can use the Amazon SNS console to control which phone numbers receive SMS messages from your account.

Opting-in a phone number that has been opted-out the Amazon SNS console

You can view which phone numbers have been opted-out of receiving SMS messages from your account, and you can opt-in these phone numbers to resume sending messages to them.

You can opt-in a phone number only once every 30 days.

1. Sign in to the Amazon SNS console.
2. In the console menu, set the region selector to a region that supports SMS messaging.
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, in the **Opted-out phone numbers** section, opted-out phone numbers are displayed.
5. Select the check box for the phone number that you want to opt-in, and choose **Opt in**. The phone number is no longer opted-out and will receive SMS messages that you send to it.

Deleting an SMS subscription the Amazon SNS console

Delete an SMS subscription to stop sending SMS messages to that phone number when you publish to your topics.

1. On the navigation panel, choose **Subscriptions**.
2. Select the check boxes for the subscriptions that you want to delete. Then choose **Actions**, and choose **Delete Subscriptions**.
3. In the **Delete** window, choose **Delete**. Amazon SNS deletes the subscription and displays a success message.

Deleting a topic the Amazon SNS console

Delete a topic when you no longer want to publish messages to its subscribed endpoints.

1. On the navigation panel, choose **Topics**.
2. Select the check boxes for the topics that you want to delete. Then choose **Actions**, and choose **Delete Topics**.
3. In the **Delete** window, choose **Delete**. Amazon SNS deletes the topic and displays a success message.

Managing phone numbers and subscriptions using the AWS SDK

You can use the AWS SDKs to make programmatic requests to Amazon SNS and manage which phone numbers can receive SMS messages from your account.

To use an AWS SDK, you must configure it with your credentials. For more information, see Shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

Viewing all opted-out phone numbers using the AWS SDK

To view all opted-out phone numbers, submit a `ListPhoneNumbersOptedOut` request with the Amazon SNS API.

The following code examples show how to use `ListPhoneNumbersOptedOut`.

CLI

AWS CLI

To list SMS message opt-outs

The following `list-phone-numbers-opted-out` example lists the phone numbers opted out of receiving SMS messages.

```
aws sns list-phone-numbers-opted-out
```

Output:

```
{
    "phoneNumbers": [
        "+15555550100"
    ]
}
```

- For API details, see ListPhoneNumbersOptedOut in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutRequest;
import
 software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListOptOut {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listOpts(snsClient);
        snsClient.close();
    }

    public static void listOpts(SnsClient snsClient) {
        try {
            ListPhoneNumbersOptedOutRequest request =
 ListPhoneNumbersOptedOutRequest.builder().build();
            ListPhoneNumbersOptedOutResponse result =
 snsClient.listPhoneNumbersOptedOut(request);
            System.out.println("Status is " +
 result.sdkHttpResponse().statusCode() + "\n\nPhone Numbers: \n\n"
                    + result.phoneNumbers());

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

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

PHP

SDK for PHP

Note

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

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

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



/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages
 from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

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

try {
    $result = $SnSclient->listPhoneNumbersOptedOut();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.

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

Checking whether a phone number is opted-out using the AWS SDK

To check whether a phone number is opted-out, submit a `CheckIfPhoneNumberIsOptedOut` request with the Amazon SNS API.

The following code examples show how to use `CheckIfPhoneNumberIsOptedOut`.

.NET

SDK for .NET

Note

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

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

   /// <summary>
   /// This example shows how to use the Amazon Simple Notification Service
   /// (Amazon SNS) to check whether a phone number has been opted out.
   /// </summary>
   public class IsPhoneNumOptedOut
   {
       public static async Task Main()
       {
           string phoneNumber = "+15551112222";

           IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

           await CheckIfOptedOutAsync(client, phoneNumber);
       }

       /// <summary>
       /// Checks to see if the supplied phone number has been opted out.
       /// </summary>
       /// <param name="client">The initialized Amazon SNS Client object used
       /// to check if the phone number has been opted out.</param>
       /// <param name="phoneNumber">A string representing the phone number
       /// to check.</param>
       public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string
phoneNumber)
       {
           var request = new CheckIfPhoneNumberIsOptedOutRequest
           {
               PhoneNumber = phoneNumber,
           };

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

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

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

CLI

AWS CLI

To check SMS message opt-out for a phone number

The following `check-if-phone-number-is-opted-out` example checks whether the specified phone number is opted out of receiving SMS messages from the current AWS account.

```
aws sns check-if-phone-number-is-opted-out \
    --phone-number +1555550100
```

Output:

```
{
    "isOptedOut": false
}
```

- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import
 software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutRequest;
import
 software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CheckOptOut {
    public static void main(String[] args) {

        final String usage = """

                Usage:    <phoneNumber>

                Where:
                   phoneNumber - The mobile phone number to look up (for example,
 +1XXX5550100).

                """;

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

        String phoneNumber = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        checkPhone(snsClient, phoneNumber);
        snsClient.close();
    }

    public static void checkPhone(SnsClient snsClient, String phoneNumber) {
        try {
            CheckIfPhoneNumberIsOptedOutRequest request =
 CheckIfPhoneNumberIsOptedOutRequest.builder()
                    .phoneNumber(phoneNumber)
                    .build();

            CheckIfPhoneNumberIsOptedOutResponse result =
 snsClient.checkIfPhoneNumberIsOptedOut(request);
            System.out.println(
                    result.isOptedOut() + "Phone Number " + phoneNumber + " has
 Opted Out of receiving sns messages." +
                            "\n\nStatus was " +
 result.sdkHttpResponse().statusCode());
```

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

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

JavaScript

SDK for JavaScript (v3)

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

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

PHP

SDK for PHP

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

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

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

```
/**
 * Indicates whether the phone number owner has opted out of receiving SMS
 messages from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

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

$phone = '+1XXX5550100';

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

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for PHP API Reference*.

Opting-in a phone number that has been opted-out using the Amazon SNS API

To opt-in a phone number, submit an `OptInPhoneNumber` request with the Amazon SNS API.

You can opt-in a phone number only once every 30 days.

Deleting an SMS subscription using the AWS SDK

To delete an SMS subscription from an Amazon SNS topic, get the subscription ARN by submitting a `ListSubscriptions` request with the Amazon SNS API, and then pass the ARN to an `Unsubscribe` request.

The following code examples show how to use `Unsubscribe`.

.NET

SDK for .NET

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

Unsubscribe from a topic by a subscription ARN.

```
    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

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

C++

SDK for C++

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

```
//! Delete a subscription to an Amazon Simple Notification Service (Amazon SNS)
 topic.
/*!
  \param subscriptionARN: The Amazon Resource Name (ARN) for an Amazon SNS topic
 subscription.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::unsubscribe(const Aws::String &subscriptionARN,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::UnsubscribeRequest request;
    request.SetSubscriptionArn(subscriptionARN);

    const Aws::SNS::Model::UnsubscribeOutcome outcome =
 snsClient.Unsubscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else {
        std::cerr << "Error while unsubscribing " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

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

CLI

AWS CLI

To unsubscribe from a topic

The following `unsubscribe` example deletes the specified subscription from a topic.

```
aws sns unsubscribe \
    --subscription-arn arn:aws:sns:us-west-2:0123456789012:my-
topic:8a21d249-4329-4871-acc6-7be709c6ea7f
```

This command produces no output.

- For API details, see Unsubscribe in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

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

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class Unsubscribe {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <subscriptionArn>

                Where:
                   subscriptionArn - The ARN of the subscription to delete.
                """;

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        unSub(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                    .subscriptionArn(subscriptionArn)
                    .build();

            UnsubscribeResponse result = snsClient.unsubscribe(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode()
                    + "\n\nSubscription was removed for " +
 request.subscriptionArn());

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

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

JavaScript

SDK for JavaScript (v3)

Note

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

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

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

Kotlin

SDK for Kotlin

Note

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

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

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

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

PHP

SDK for PHP

Note

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

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Deletes a subscription to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $SnSclient->unsubscribe([
        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Unsubscribe in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def delete_subscription(subscription):
        """
        Unsubscribes and deletes a subscription.
        """
        try:
            subscription.delete()
            logger.info("Deleted subscription %s.", subscription.arn)
        except ClientError:
            logger.exception("Couldn't delete subscription %s.",
 subscription.arn)
            raise
```

- For API details, see Unsubscribe in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   TRY.
       lo_sns->unsubscribe( iv_subscriptionarn = iv_subscription_arn ).
       MESSAGE 'Subscription deleted.' TYPE 'I'.
     CATCH /aws1/cx_snsnotfoundexception.
       MESSAGE 'Subscription does not exist.' TYPE 'E'.
     CATCH /aws1/cx_snsinvalidparameterex.
       MESSAGE 'Subscription with "PendingConfirmation" status cannot be
deleted/unsubscribed. Confirm subscription before performing unsubscribe
operation.' TYPE 'E'.
   ENDTRY.
```

- For API details, see Unsubscribe in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(
                subscriptionArn: arn
            )
        )

        print("Unsubscribed.")
```

- For API details, see Unsubscribe in *AWS SDK for Swift API reference*.

Deleting a topic using the AWS SDK

To delete a topic and all of its subscriptions, get the topic ARN by submitting a `ListTopics` request with the Amazon SNS API, and then pass the ARN to the `DeleteTopic` request.

The following code examples show how to use `DeleteTopic`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a topic by its topic ARN.

```
    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteTopic in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::DeleteTopicOutcome outcome =
 snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting topic " << topicARN << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteTopic in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an SNS topic

The following `delete-topic` example deletes the specified SNS topic.

```
aws sns delete-topic \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

This command produces no output.

- For API details, see DeleteTopic in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```

import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

- For API details, see DeleteTopic in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:     <topicArn>

                Where:
                   topicArn - The ARN of the topic to delete.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        System.out.println("Deleting a topic with name: " + topicArn);
        deleteSNSTopic(snsClient, topicArn);
        snsClient.close();
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                    .topicArn(topicArn)
                    .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteTopic in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
```

```
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteTopic in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteSNSTopic(topicArnVal: String) {
    val request =
        DeleteTopicRequest {
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was successfully deleted.")
    }
}
```

- For API details, see DeleteTopic in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see DeleteTopic in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

- For API details, see DeleteTopic in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).
        MESSAGE 'SNS topic deleted.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteTopic in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: arn)
        )
```

- For API details, see DeleteTopic in *AWS SDK for Swift API reference*.

Amazon SNS SMS activity monitoring

By monitoring your SMS activity, you can keep track of destination phone numbers, successful or failed deliveries, reasons for failure, costs, and other information. Amazon SNS helps by summarizing statistics in the console, sending information to Amazon CloudWatch, and sending daily SMS usage reports to an Amazon S3 bucket that you specify.

Viewing Amazon SNS SMS delivery statistics

You can use the Amazon SNS console to view statistics about your recent SMS deliveries.

1. Sign in to the Amazon SNS console.
2. In the console menu, set the region selector to a region that supports SMS messaging.
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Text messaging (SMS)** page, in the **Account stats** section, view the charts for your transactional and promotional SMS message deliveries. Each chart shows the following data for the preceding 15 days:

   - Delivery rate (percentage of successful deliveries)
   - Sent (number of delivery attempts)
   - Failed (number of delivery failures)

On this page, you can also choose the **Usage** button to go to the Amazon S3 bucket where you store your daily usage reports. For more information, see Subscribing to Amazon SNS daily SMS usage reports.

Amazon SNS SMS delivery monitoring with Amazon CloudWatch metrics and logs

You can use Amazon CloudWatch and Amazon CloudWatch Logs to monitor your SMS message deliveries.

Viewing Amazon CloudWatch metrics

Amazon SNS automatically collects metrics about your SMS message deliveries and pushes them to Amazon CloudWatch. You can use CloudWatch to monitor these metrics and create alarms to alert you when a metric crosses a threshold. For example, you can monitor CloudWatch metrics to learn your SMS delivery rate and your month-to-date SMS charges.

For information about monitoring CloudWatch metrics, setting CloudWatch alarms, and the types of metrics available, see Monitoring Amazon SNS topics using CloudWatch.

Viewing CloudWatch Logs

You can collect information about successful and unsuccessful SMS message deliveries by enabling Amazon SNS to write to Amazon CloudWatch Logs. For each SMS message that you send, Amazon SNS writes a log that includes the message price, the success or failure status, the reason for failure (if the message failed), the message dwell time, and other information.

To enable and view CloudWatch Logs for your SMS messages

1. Sign in to the Amazon SNS console.
2. In the console menu, set the region selector to a region that supports SMS messaging.
3. On the navigation panel, choose **Text messaging (SMS)**.
4. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
5. On the next page, expand the **Delivery status logging** section.
6. For **Success sample rate**, specify the percentage of successful SMS deliveries for which Amazon SNS will write logs in CloudWatch Logs. For example:

    - To write logs only for failed deliveries, set this value to 0.
    - To write logs for 10% of your successful deliveries, set it to 10.

    If you don't specify a percentage, Amazon SNS writes logs for all successful deliveries.
7. To provide the required permissions, do one of the following:

    - To create a new service role, choose **Create new service role** and then **Create new roles**. On the next page, choose **Allow** to give Amazon SNS write access to your account's resources.
    - To use an existing service role, choose **Use existing service role** and then paste the ARN name in the **IAM role for successful and failed deliveries** box.

      The service role you specify must allow write access to your account's resources. For more information on creating IAM roles, see Creating a role for an AWS service in the *IAM User Guide*.

8. Choose **Save changes**.
9. Back on the **Mobile text messaging (SMS)** page, go to the **Delivery status logs** section to view any available logs.

> **Note**
>
> Depending on the destination phone number's carrier, it can take up to 72 hours for delivery logs to appear in the Amazon SNS console.

Example log for successful SMS delivery

The delivery status log for a successful SMS delivery will resemble the following example:

```
{
    "notification": {
        "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
        "timestamp": "2016-06-28 00:40:34.558"
    },
    "delivery": {
        "phoneCarrier": "My Phone Carrier",
        "mnc": 270,
        "numberOfMessageParts": 1,
        "destination": "+1XXX5550100",
        "priceInUSD": 0.00645,
        "smsType": "Transactional",
        "mcc": 310,
        "providerResponse": "Message has been accepted by phone carrier",
        "dwellTimeMs": 599,
        "dwellTimeMsUntilDeviceAck": 1344
    },
    "status": "SUCCESS"
}
```

Example log for failed SMS delivery

The delivery status log for a failed SMS delivery will resemble the following example:

```
{
    "notification": {
        "messageId": "1077257a-92f3-5ca3-bc97-6a915b310625",
        "timestamp": "2016-06-28 00:40:34.559"
    },
    "delivery": {
        "mnc": 0,
        "numberOfMessageParts": 1,
        "destination": "+1XXX5550100",
        "priceInUSD": 0.00645,
        "smsType": "Transactional",
        "mcc": 0,
        "providerResponse": "Unknown error attempting to reach phone",
        "dwellTimeMs": 1420,
        "dwellTimeMsUntilDeviceAck": 1692
    },
    "status": "FAILURE"
}
```

SMS delivery failure reasons

The reason for a failure is provided with the `providerResponse` attribute. SMS messages might fail to deliver for the following reasons:

- Blocked as spam by phone carrier
- Destination is on a blocked list
- Invalid phone number
- Message body is invalid
- Phone carrier has blocked this message
- Phone carrier is currently unreachable/unavailable
- Phone has blocked SMS
- Phone is on a blocked list
- Phone is currently unreachable/unavailable
- Phone number is opted out
- This delivery would exceed max price
- Unknown error attempting to reach phone

Subscribing to Amazon SNS daily SMS usage reports

You can monitor your SMS deliveries by subscribing to daily usage reports from Amazon SNS. For each day that you send at least one SMS message, Amazon SNS delivers a usage report as a CSV file to the specified Amazon S3 bucket. It takes 24 hours for the SMS usage report to be available in the Amazon S3 bucket.

Daily usage report information

The usage report includes the following information for each SMS message that you send from your account.

Note that the report does not include messages that are sent to recipients who have opted out.

- Time of publication for message (in UTC)
- Message ID
- Destination phone number
- Message type
- Delivery status
- Message price (in USD)
- Part number (a message is split into multiple parts if it is too long for a single message)
- Total number of parts

> **Note**
>
> If Amazon SNS did not receive the part number, we set its value to zero.

Subscribing to daily usage reports

To subscribe to daily usage reports, you must create an Amazon S3 bucket with the appropriate permissions.

To create an Amazon S3 bucket for your daily usage reports

1. From the AWS account that sends SMS messages, sign in to the Amazon S3 console.
2. Choose **Create Bucket**.
3. For **Bucket Name**, we recommend that you enter a name that is unique for your account and your organization. For example, use the pattern `<my-bucket-prefix>-<account_id>-<org-id>`.

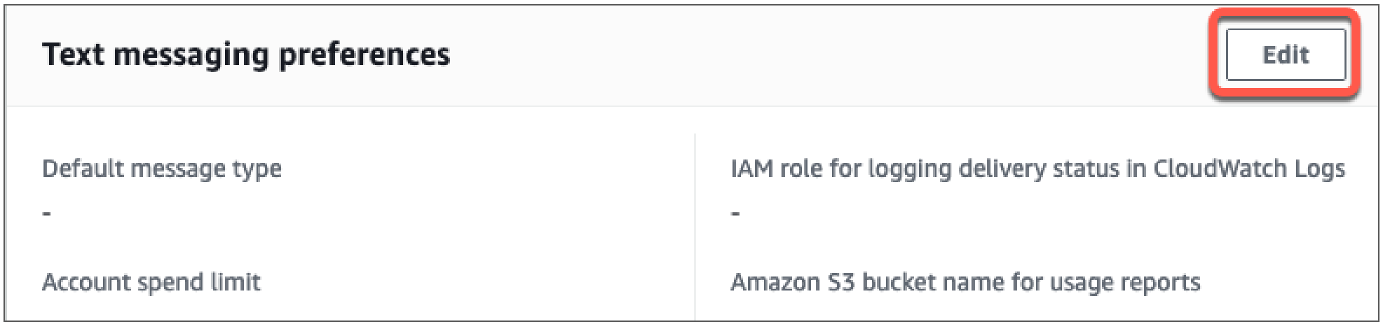
For information about conventions and restrictions for bucket names, see Rules for Bucket Naming in the *Amazon Simple Storage Service User Guide*.

4. Choose **Create**.
5. In the **All Buckets** table, choose the bucket.
6. In the **Permissions** tab, choose **Bucket policy**.
7. In the **Bucket Policy Editor** window, provide a policy that allows the Amazon SNS service principal to write to your bucket. For an example, see Example bucket policy.

   If you use the example policy, remember to replace `my-s3-bucket` with the bucket name that you chose in Step 3.
8. Choose **Save**.

To subscribe to daily usage reports

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Text messaging (SMS)**.
3. On the **Text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.



Text messaging preferences

Default message type

-

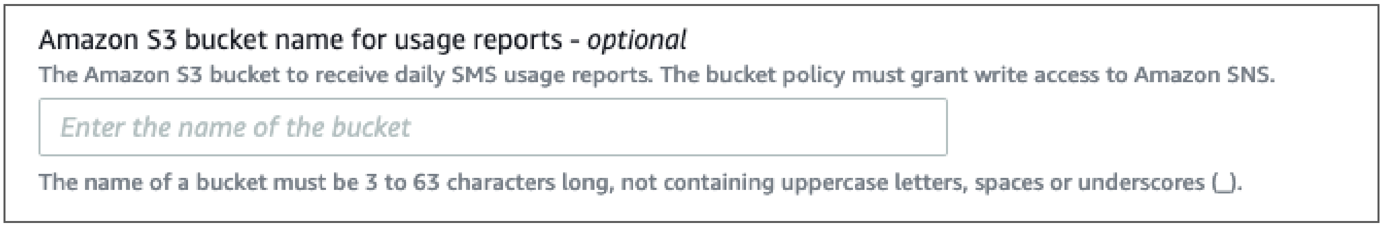
Account spend limit

IAM role for logging delivery status in CloudWatch Logs

-

Amazon S3 bucket name for usage reports

4. On the **Edit text messaging preferences** page, in the **Details** section, specify the **Amazon S3 bucket name for usage reports**.



Amazon S3 bucket name for usage reports - *optional*

The Amazon S3 bucket to receive daily SMS usage reports. The bucket policy must grant write access to Amazon SNS.

Enter the name of the bucket

The name of a bucket must be 3 to 63 characters long, not containing uppercase letters, spaces or underscores (_).

5. Choose **Save changes**.

Example bucket policy

The following policy allows the Amazon SNS service principal to perform the `s3:PutObject`, `s3:GetBucketLocation`, and `s3:ListBucket` actions.

AWS provides tools for all services with service principals that have been given access to resources in your account. When the principal in an Amazon S3 bucket policy statement is an confused deputy problem. To limit which region and account from which the bucket can receive daily usage reports, use `aws:SourceArn` as shown in the example below. If you do not wish to limit which regions can generate these reports, use `aws:SourceAccount` to limit based on which account is generating the reports. If you don't know the ARN of the resource, use `aws:SourceAccount`.

Use the following example that includes confused deputy protection when you create an Amazon S3 bucket to receive daily SMS usage reports from Amazon SNS.

JSON

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "AllowPutObject",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account_id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"
                }
            }
        },
        {
            "Sid": "AllowGetBucketLocation",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "s3:GetBucketLocation",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account_id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"
                }
            }
        },
        {
            "Sid": "AllowListBucket",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account_id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"
                }
            }
        }
    ]
}
```

Note

You can publish usage reports to Amazon S3 buckets that are owned by the AWS account that's specified in the `Condition` element in the Amazon S3 policy. To publish usage reports to an Amazon S3 bucket that another AWS account owns, see How can I copy Amazon S3 objects from another AWS account?.

Example daily usage report

After you subscribe to daily usage reports, each day, Amazon SNS puts a CSV file with usage data in the following location:

```
<my-s3-bucket>/SMSUsageReports/<region>/YYYY/MM/DD/00x.csv.gz
```

Each file can contain up to 50,000 records. If the records for a day exceed this quota, Amazon SNS will add multiple files. The following shows an example report:

```
PublishTimeUTC,MessageId,DestinationPhoneNumber,MessageType,DeliveryStatus,PriceInUSD,PartNumbe
2016-05-10T03:00:29.476Z,96a298ac-1458-4825-
a7eb-7330e0720b72,1XXX5550100,Promotional,Message has been accepted by phone
 carrier,0.90084,0,1
2016-05-10T03:00:29.561Z,1e29d394-
d7f4-4dc9-996e-26412032c344,1XXX5550100,Promotional,Message has been accepted by phone
 carrier,0.34322,0,1
2016-05-10T03:00:30.769Z,98ba941c-afc7-4c51-
ba2c-56c6570a6c08,1XXX5550100,Transactional,Message has been accepted by phone
 carrier,0.27815,0,1
```

Requesting support for Amazon SNS SMS messaging

⚠ **Important**

The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

Certain SMS options with Amazon SNS aren't available for your AWS account until you contact Support. Create a case in the AWS Support Center to request any of the following:

- An increase to your monthly SMS spending threshold

  By default, the monthly spending threshold is $1.00 (USD). Your spending threshold determines the volume of messages that you can send with Amazon SNS. You can request a spending threshold that meets the expected monthly message volume for your SMS use case.

- A move from the SMS sandbox so that you can send SMS messages without restrictions. For more information, see Moving out of the Amazon SNS SMS sandbox.
- A dedicated origination number
- A dedicated sender ID. A sender ID is a custom ID that is shown as the sender on the recipient's device. For example, you can use your business brand to make the message source easier to recognize. Support for sender IDs varies by country or region. For more information, see Supported countries and regions for SMS messaging with AWS End User Messaging SMS in the *AWS End User Messaging SMS User Guide*.

Requesting increases to your monthly Amazon SNS SMS spending quota

Amazon SNS provides spending quotas to help you manage the maximum per-month cost incurred by sending SMS using your account. The spending quota limits your risk in case of malicious attack, and prevents your upstream application from sending more messages than expected. You can configure Amazon SNS to stop publishing SMS messages when it determines that sending an SMS message will incur a cost that exceeds your spending quota for the current month.

To ensure your operations are not impacted, we recommend requesting a spending quota high enough to support your production workloads. For more information, see Step 1: Open an Amazon SNS SMS case. Once you have received the quota, you can manage your risk by applying the full quota, or a smaller value, as described in Step 2: Update your SMS settings. By applying a smaller value, you can control your monthly spending with the option to scale up if necessary.

> ⚠ **Important**
>
> Because Amazon SNS is a distributed system, it stops sending SMS messages within minutes if the spending quota is exceeded. During this period, if you continue to send SMS messages, you might incur costs that exceed your quota.

We set the spending quota for all new accounts at $1.00 (USD) per month. This quota is intended to let you test the message-sending capabilities of Amazon SNS. To request an increase to the SMS spending quota for your account, open a quota increase case in the AWS Support Center.

Topics

- Step 1: Open an Amazon SNS SMS case
- Step 2: Update your SMS settings on the Amazon SNS console

Step 1: Open an Amazon SNS SMS case

You can request an increase to your monthly spending quota by opening a quota increase case in the AWS Support Center.

> **Note**
>
> Some of the fields on the request form are marked as "optional." However, Support requires all of the information that's mentioned in the following steps in order to process your request. If you don't provide all of the required information, you may experience delays in processing your request.

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/.
2. On the **Support** menu, choose **Support Center**.
3. On the **Your support cases** pane, choose **Create case**.
4. Choose the **Looking for service limit increases?** link, then complete the following:
   - For **Limit type**, choose **SNS Text Messaging**.
   - (Optional) For **Provide a link to the site or app which will be sending SMS messages**, provide information about the website, application, or service that will send SMS messages.
   - (Optional) For **What type of messages do you plan to send**, choose the type of message that you plan to send using your long code:
     - **One Time Password** – Messages that provide passwords that your customers use to authenticate with your website or application.
     - **Promotional** – Noncritical messages that promote your business or service, such as special offers or announcements.
     - **Transactional** – Important informational messages that support customer transactions, such as order confirmations or account alerts. Transactional messages must not contain promotional or marketing content.
   - (Optional) For **Which AWS Region will you be sending messages from**, choose the region that you'll be sending messages from.
   - (Optional) For **Which countries do you plan to send messages to**, enter the country or region that you want to purchase short codes in.
   - (Optional) In the **How do your customers opt to receive messages from you**, provide details about your opt-in process.

- (Optional) In the **Please provide the message template that you plan to use to send messages to your customers** field, include the template that you will be using.

5. Under **Requests**, complete the following sections:

   - For the **Region**, choose the Region from which you'll be sending messages.

     > **Note**
     >
     > The Region is required in the **Requests** section. Even if you provided this information in the **Case details** section you must also include it here.

   - For **Resource Type**, choose **General Limits**.
   - For **Limit**, choose **Account Spend Threshold Increase**.

6. For New limit value, enter the maximum amount (in USD) that you can spend on SMS each calendar month.

7. Under **Case description**, for **Use case description**, provide the following details:

   - The website or app of the company or service that's sending SMS messages.
   - The service that's provided by your website or app, and how your SMS messages contribute to that service.
   - How users sign up to voluntarily receive your SMS messages on your website, app, or other location.

   If your requested spending quota (the value you specified for **New quota value**) exceeds $10,000 (USD), provide the following additional details for each country that you're messaging:

   - Whether you're using a sender ID or short code. If you're using a sender ID, provide:
     - The sender ID.
     - Whether the sender ID is registered with wireless carriers in the country.
   - The maximum expected transactions-per-second (TPS) for your messaging.
   - The average message size.
   - The template for the messages that you send to the country.
   - (Optional) Character encoding needs, if any.

8. (Optional) If you want to submit any further requests, choose **Add another request**. If you include multiple requests, provide the required information for each. For the required information, see the other sections within Requesting support for Amazon SNS SMS messaging.
9. Under **Contact options**, for **Preferred contact language**, choose the language in which you want to receive communications for this case.
10. When you finish, choose **Submit**.

The Support team provides an initial response to your request within 24 hours.

To prevent our systems from being used to send unsolicited or malicious content, we consider each request carefully. If we can, we will grant your request within this 24-hour period. However, if we need additional information from you, it might take longer to resolve your request.

If your use case doesn't align with our policies, we might be unable to grant your request.

Step 2: Update your SMS settings on the Amazon SNS console

After we notify you that your monthly spending quota has been increased, you have to adjust the spending quota for your account on the Amazon SNS console.

> ⚠ **Important**
>
> You must complete the following steps or your SMS spend limit will not be increased.

To adjust your spending quota on the console

1. Sign in to the Amazon SNS console.
2. Open the left navigation menu, expand **Mobile**, and then choose **Text messaging (SMS)**.
3. On the **Mobile text messaging (SMS)** page, in the **Text messaging preferences** section, choose **Edit**.
4. On the **Edit text messaging preferences** page, in the **Details** section, enter your new SMS spend limit in the **Account spend limit** field.

> **Note**
>
> You might receive a warning that the entered value is larger than the default spend limit. You can ignore this.

5. Choose **Save** changes.

> **Note**
>
> If you get an "Invalid Parameter" error, check the contact from AWS Support and confirm that you entered the correct new SMS spend limit. If you still experience a problem, open a case in the AWS Support Center.

When you create your case in the Support Center, be sure to include all the required information for the type of request that you're submitting. Otherwise, Support must contact you to obtain this information before proceeding. By submitting a detailed case, you help ensure that your case is fulfilled without delays. For the required details for specific types of SMS requests, see the following topics.

For more information on sender IDs, see the following documentation in the *AWS End User Messaging SMS User Guide*:

AWS End User Messaging SMS Topic	Description
Requesting a spending quota increase	Your spending quota determines how much money you can spend sending SMS messages through AWS End User Messaging SMS each month.
Open a case in support center for a sender ID	If you plan to send messages to recipients a country where sender IDs are required, you can request a sender ID by creating a new case in the Support Center.

Best practices for Amazon SNS SMS messaging

⚠ **Important**

The Amazon SNS SMS Developer Guide has been updated. Amazon SNS has integrated with AWS End User Messaging SMS for the delivery of SMS messages. This guide contains the latest information on how to create, configure, and manage your Amazon SNS SMS messages.

Mobile phone users tend to have a very low tolerance for unsolicited SMS messages. Response rates for unsolicited SMS campaigns will almost always be low, and therefore the return on your investment will be poor.

Additionally, mobile phone carriers continuously audit bulk SMS senders. They throttle or block messages from numbers that they determine to be sending unsolicited messages.

Sending unsolicited content is also a violation of the AWS acceptable use policy. The Amazon SNS team routinely audits SMS campaigns, and might throttle or block your ability to send messages if it appears that you're sending unsolicited messages.

Finally, in many countries, regions, and jurisdictions, there are severe penalties for sending unsolicited SMS messages. For example, in the United States, the Telephone Consumer Protection Act (TCPA) states that consumers are entitled to $500–$1,500 in damages (paid by the sender) for each unsolicited message that they receive.

This section describes several best practices that might help you improve your customer engagement and avoid costly penalties. However, note that this section doesn't contain legal advice. Always consult an attorney to obtain legal advice.

Comply with laws, regulations, and carrier requirements

You can face significant fines and penalties if you violate the laws and regulations of the places where your customers reside. For this reason, it's vital to understand the laws related to SMS messaging in each country or region where you do business.

The following list includes links to key laws that apply to SMS communications in major markets around the world.

- **United States**: The Telephone Consumer Protection Act of 1991, also known as TCPA, applies to certain types of SMS messages. For more information, see the rules and regulations at the Federal Communications Commission website.
- **United Kingdom**: The Privacy and Electronic Communications (EC Directive) Regulations 2003, also known as PECR, applies to certain types of SMS messages. For more information, see What are PECR? at the website of the UK Information Commissioner's Office.
- **European Union**: The Privacy and Electronic Communications Directive 2002, sometimes known as the ePrivacy Directive, applies to some types of SMS messages. For more information, see the full text of the law at the Europa.eu website.
- **Canada**: The Fighting Internet and Wireless Spam Act, more commonly known as Canada's Anti-Spam Law or CASL, applies to certain types of SMS messages. For more information, see the full text of the law at the website of the Parliament of Canada.
- **Japan**: The Act on Regulation of Transmission of Specific Electronic Mail may apply to certain types of SMS messages. For more information, see Japan's countermeasures against spam at the website of the Japanese Ministry of Internal Affairs and Communications.

As a sender, these laws may apply to you even if your company or organization isn't based in one of these countries. Some of the laws in this list were originally created to address unsolicited email or telephone calls, but have been interpreted or expanded to apply to SMS messages as well. Other countries and regions may have their own laws related to the transmission of SMS messages. Consult an attorney in each country or region where your customers are located to obtain legal advice.

In many countries, the local carriers ultimately have the authority to determine what kind of traffic flows over their networks. This means that the carriers might impose restrictions on SMS content that exceed the minimum requirements of local laws.

Obtain permission

Never send messages to recipients who haven't explicitly asked to receive the specific types of messages that you plan to send. Don't share opt-in lists, even among organizations within the same company.

If recipients can sign up to receive your messages by using an online form, add systems that prevent automated scripts from subscribing people without their knowledge. You should also limit the number of times a user can submit a phone number in a single session.

When you receive an SMS opt-in request, send the recipient a message that asks them to confirm that they want to receive messages from you. Don't send that recipient any additional messages until they confirm their subscription. A subscription confirmation message might resemble the following example:

```
Text YES to join ExampleCorp alerts. 2 msgs/month. Msg & data rates may
apply. Reply HELP for help, STOP to cancel.
```

Maintain records that include the date, time, and source of each opt-in request and confirmation. This might be useful if a carrier or regulatory agency requests it, and can also help you perform routine audits of your customer list.

Opt-in workflow

In some cases (like US Toll-Free or Short Code registration) mobile carriers require you to provide mockups or screen shot of your entire opt-in workflow. The mockups or screen shot must closely resemble the opt-in workflow that your recipients will complete.

Your mockups or screen shot should include all of the required disclosures listed below to maintain the highest level of compliance.

Required disclosures

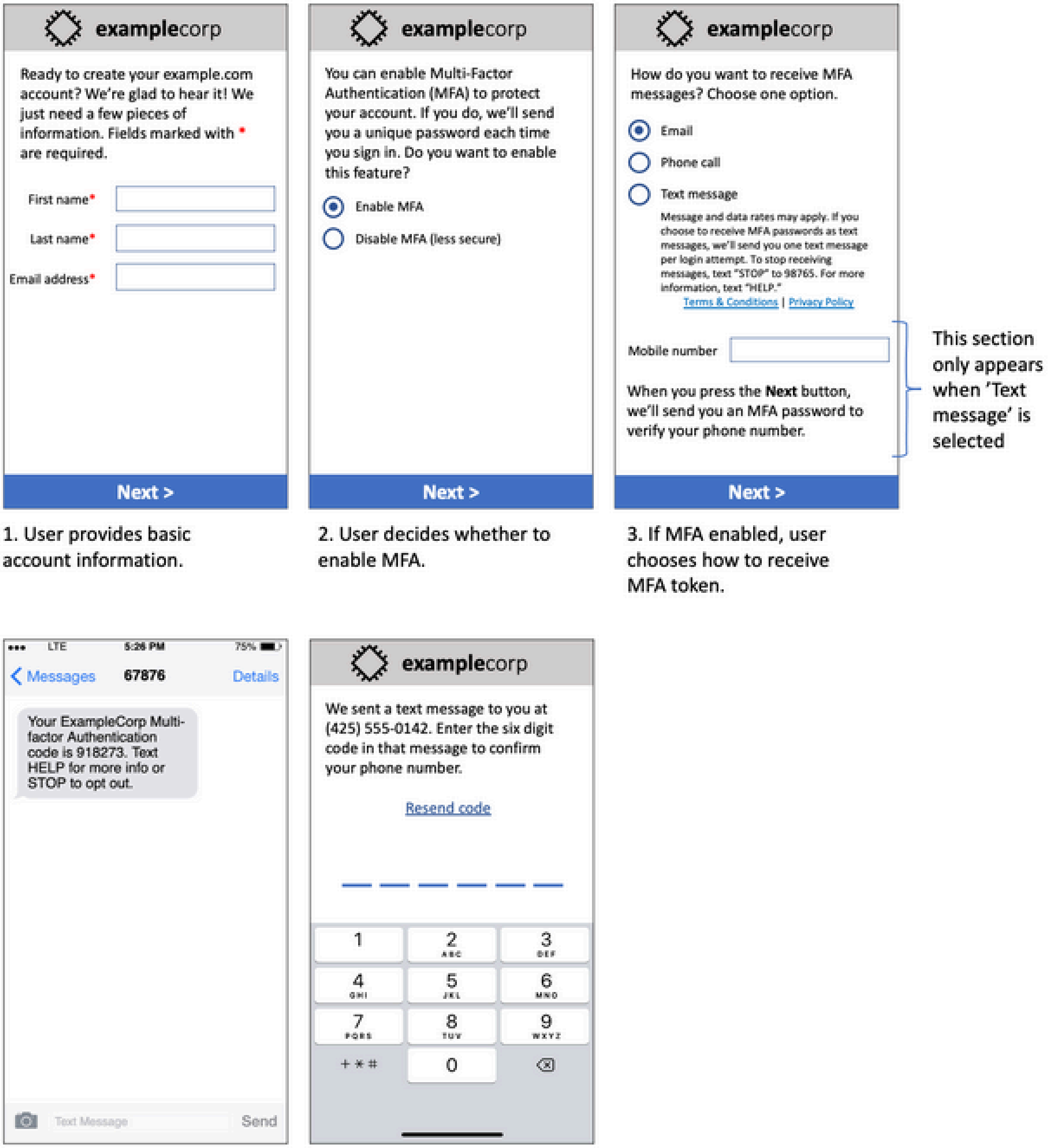
- A description of the messaging use case that you will send through your program.
- The phrase “Message and data rates may apply.”
- An indication of how often recipients will get messages from you. For example, a recurring messaging program might say “one message per week.” A one-time password or multi-factor authentication use case might say “message frequency varies” or “one message per login attempt.”
- Links to your Terms and Conditions and Privacy Policy documents.

Common rejection reasons for non compliant opt-ins

- If the provided company name does not match what is provided in the mockup or screen shot. Any non obvious relations should be explained in the opt-in workflow description.
- If it appears that a message will be sent to the recipient, but no consent is explicitly gathered before doing so. Explicit consent is a requirement of all messaging.

- If it appears that receiving a text message is required to sign up for a service. This is not compliant if the workflow doesn't provide any alternative to receiving an opt-in message in another form like email or a voice call.
- If the opt-in language is presented entirely in the Terms of Service. The disclosures should always be presented to the recipient at time of opt-in rather than housed inside a linked policy document.
- If a customer provided consent to receive one type of message from you and you send them other types of text messages. For example they consent to receive one-time passwords but are also sent polling and survey messages.
- If the required disclosures (listed above) are not presented to the recipients.

The following example complies with the mobile carriers' requirements for a multi-factor authentication use case.



Ready to create your example.com account? We're glad to hear it! We just need a few pieces of information. Fields marked with * are required.

First name*

Last name*

Email address*

1. User provides basic account information.

You can enable Multi-Factor Authentication (MFA) to protect your account. If you do, we'll send you a unique password each time you sign in. Do you want to enable this feature?

● Enable MFA

○ Disable MFA (less secure)

2. User decides whether to enable MFA.

How do you want to receive MFA messages? Choose one option.

● Email

○ Phone call

○ Text message

Message and data rates may apply. If you choose to receive MFA passwords as text messages, we'll send you one text message per login attempt. To stop receiving messages, text "STOP" to 98765. For more information, text "HELP."

Terms & Conditions | Privacy Policy

Mobile number

When you press the **Next** button, we'll send you an MFA password to verify your phone number.

This section only appears when 'Text message' is selected

3. If MFA enabled, user chooses how to receive MFA token.

Your ExampleCorp Multi-factor Authentication code is 918273. Text HELP for more info or STOP to opt out.

4. If user chooses to receive MFA token by text, send a token.

We sent a text message to you at (425) 555-0142. Enter the six digit code in that message to confirm your phone number.

Resend code

5. User enters MFA token to verify phone number.

Mockup of a multi-factor authentication use case

It contains finalized text and images, and it shows the entire opt-in flow, complete with annotations. In the opt-in flow, the customer has to take distinct, intentional actions to provide their consent to receive text messages and contains all of the required disclosures.

Other opt-in workflow types

Mobile carriers will also accept opt-in workflows outside of applications and websites like verbal or written opt-in if it complies with what is outlined above. A compliant opt-in workflow and verbal or written script will gather explicit consent from the recipient to receive a specific message type. Examples of this include the verbal script a support agent uses to gather consent before recording into a service database or a phone number listed on a promotional flyer. To provide a mockup of these opt-in workflow types you can provide a screen shot of your opt-in script, marketing material or database where numbers are collected. Mobile carriers may have additional questions around these use cases if an opt-in is not clear or the use case exceed certain volumes.

Don't send to old lists

People change phone numbers often. A phone number that you gathered consent to contact two years ago might belong to somebody else today. Don't use an old list of phone numbers for a new messaging program; if you do, you're likely to have some messages fail because the number is no longer in service, and some people who opt out because they don't remember giving you their consent in the first place.

Audit your customer lists

If you send recurring SMS campaigns, audit your customer lists on a regular basis. Auditing your customer lists ensures that the only customers who receive your messages are those who are interested in receiving them.

When you audit your list, send each opted-in customer a message that reminds them that they're subscribed, and provides them with information about unsubscribing. A reminder message might resemble the following example:

```
You're subscribed to ExampleCorp alerts. Msg & data rates may apply. Reply
HELP for help, STOP to unsubscribe.
```

Keep records

Keep records that show when each customer requested to receive SMS messages from you, and which messages you sent to each customer. Many countries and regions around the world require SMS senders to maintain these records in a way that can be easily retrieved. Mobile carriers might also request this information from you at any time. The exact information that you have to provide varies by country or region. For more information about record-keeping requirements, review the regulations about commercial SMS messaging in each country or region where your customers are located.

Occasionally, a carrier or regulatory agency asks us to provide proof that a customer opted to receive messages from you. In these situations, Support contacts you with a list of the information that the carrier or agency requires. If you can't provide the necessary information, we may pause your ability to send additional SMS messages.

Make your messages clear, honest, and concise

SMS is a unique medium. The 160-character-per-message limit means that your messages have to be concise. Techniques that you might use in other communication channels, such as email, might not apply to the SMS channel, and might even seem dishonest or deceptive when used with SMS messages. If the content in your messages doesn't align with best practices, recipients might ignore your messages; in the worst case, the mobile carriers might identify your messages as spam and block future messages from your phone number.

This section provides some tips and ideas for creating an effective SMS message body.

Identify yourself as the sender

Your recipients should be able to immediately tell that a message is from you. Senders who follow this best practice include an identifying name ("program name") at the beginning of each message.

Don't do this:

```
Your account has been accessed from a new device. Reply Y to confirm.
```

Try this instead:

```
ExampleCorp Financial Alerts: You have logged in to your account from a
new device. Reply Y to confirm, or STOP to opt-out.
```

Don't try to make your message look like a person-to-person message

Some marketers are tempted to add a personal touch to their SMS messages by making their messages appear to come from an individual. However, this technique might make your message seem like a phishing attempt.

Don't do this:

```
Hi, this is Jane. Did you know that you can save up to 50% at
Example.com? Click here for more info: https://www.example.com.
```

Try this instead:

```
ExampleCorp Offers: Save 25-50% on sale items at Example.com. Click here
to browse the sale: https://www.example.com. Text STOP to opt-out.
```

Be careful when talking about money

Scammers often prey upon people's desire to save and receive money. Don't make offers seem too good to be true. Don't use the lure of money to deceive people. Don't use currency symbols to indicate money.

Don't do this:

```
Save big $$$ on your next car repair by going to https://
www.example.com.
```

Try this instead:

```
ExampleCorp Offers: Your ExampleCorp insurance policy gets you discounts
at 2300+ repair shops nationwide. More info at https://www.example.com.
Text STOP to opt-out.
```

Use only the necessary characters

Brands are often inclined to protect their trademarks by including trademark symbols such as ™ or ® in their messages. However, these symbols are not part of the standard set of characters (known as the GSM alphabet) that can be included in a 160-character SMS message. When you send a message that contains one of these characters, your message is automatically sent using a different character encoding system, which only supports 70 characters per message part. As a result, your message could be broken into several parts. Because you're billed for each message part that you send, it could cost you more than you expect to spend to send the entire message. Additionally, your recipients might receive several sequential messages from you, rather than one single message. For more information about SMS character encoding, see SMS character limits in Amazon SNS.

Don't do this:

```
ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget® at
example.com and use the promo code WIDGET.
```

Try this instead:

```
ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget(R) at
example.com and use the promo code WIDGET.
```

> **Note**
>
> The two preceding examples are almost identical, but the first example contains a Registered Trademark symbol (®), which is not part of the GSM alphabet. As a result, the first example is sent as two message parts, while the second example is sent as one message part.

Use valid, safe links

If your message includes links, double-check the links to make sure that they work. Test your links on a device outside your corporate network to ensure that links resolve properly. Because of the 160-character limit of SMS messages, very long URLs could be split across multiple messages. You should use redirect domains to provide shortened URLs. However, you shouldn't use free link-shortening services such as tinyurl.com or bitly.com, because carriers tend to filter messages that include links on these domains. However, you can use paid link-shortening services as long as your links point to a domain that is dedicated to the exclusive use of your company or organization.

Don't do this:

```
Go to https://tinyurl.com/4585y8mr today for a special offer!
```

Try this instead:

```
ExampleCorp Offers: Today only, get an exclusive deal on an ExampleCorp
Widget. See https://a.co/cFKmaRG for more info. Text STOP to opt-out.
```

Limit the number of abbreviations that you use

The 160-character limitation of the SMS channel leads some senders to believe that they need to use abbreviations extensively in their messages. However, the overuse of abbreviations can seem unprofessional to many readers, and could cause some users to report your message as spam. It's completely possible to write a coherent message without using an excessive number of abbreviations.

Don't do this:

```
Get a gr8 deal on ExampleCorp widgets when u buy a 4-pack 2day.
```

Try this instead:

```
ExampleCorp Alerts: Today only—an exclusive deal on ExampleCorp Widgets
at example.com. Text STOP to opt-out.
```

Respond appropriately

When a recipient replies to your messages, make sure that you respond with useful information. For example, when a customer responds to one of your messages with the keyword "HELP", send them information about the program that they're subscribed to, the number of messages you'll send each month, and the ways that they can contact you for more information. A HELP response might resemble the following example:

```
HELP: ExampleCorp alerts: email help@example.com or call 425-555-0199. 2
msgs/month. Msg & data rates may apply. Reply STOP to cancel.
```

When a customer replies with the keyword "STOP", let them know that they won't receive any further messages. A STOP response might resemble the following example:

```
You're unsubscribed from ExampleCorp alerts. No more messages will be sent.
Reply HELP, email help@example.com, or call 425-555-0199 for more info.
```

Adjust your sending based on engagement

Your customers' priorities can change over time. If customers no longer find your messages to be useful, they might opt out of your messages entirely, or even report your messages as unsolicited. For these reasons, it's important that you adjust your sending practices based on customer engagement.

For customers who rarely engage with your messages, you should adjust the frequency of your messages. For example, if you send weekly messages to engaged customers, you could create a separate monthly digest for customers who are less engaged.

Finally, remove customers who are completely unengaged from your customer lists. This step prevents customers from becoming frustrated with your messages. It also saves you money and helps protect your reputation as a sender.

Send at appropriate times

Only send messages during normal daytime business hours. If you send messages at dinner time or in the middle of the night, there's a good chance that your customers will unsubscribe from your lists in order to avoid being disturbed. Furthermore, it doesn't make sense to send SMS messages when your customers can't respond to them immediately.

If you send campaigns or journeys to very large audiences, double-check the throughput rates for your origination numbers. Divide the number of recipients by your throughput rate to determine how long it will take to send messages to all of your recipients.

Avoid cross-channel fatigue

In your campaigns, if you use multiple communication channels (such as email, SMS, and push messages), don't send the same message in every channel. When you send the same message at the same time in more than one channel, your customers will probably perceive your sending behavior to be annoying rather than helpful.

Use dedicated short codes

If you use short codes, maintain a separate short code for each brand and each type of message. For example, if your company has two brands, use a separate short code for each one. Similarly, if you send both transactional and promotional messages, use a separate short code for each type of message. To learn more about requesting short codes, see Requesting short codes for SMS messaging with AWS End User Messaging SMS in the *AWS End User Messaging SMS User Guide*.

Verify your destination phone numbers

When you send SMS messages through Amazon SNS, you're billed for each message part you send. The price you pay per message part varies on the recipient's country or region. For more information about SMS pricing, see AWS Worldwide SMS Pricing.

When Amazon SNS accepts a request to send an SMS message (as the result of a call to the SendMessages API, or as the result of a campaign or journey being launched), you're charged for sending that message. This statement is true even if the intended recipient doesn't actually receive the message. For example, if the recipient's phone number is no longer in service, or if the number that you sent the message to wasn't a valid mobile phone number, you're still billed for sending the message.

Amazon SNS accepts valid requests to send SMS messages and attempts to deliver them. For this reason, you should validate that the phone numbers that you send messages to are valid mobile numbers. You can use AWS End User Messaging SMS to send a test message to determine if a phone number is valid and what type of number it is (such as mobile, landline, or VoIP). For more information, see Send a test message with the SMS simulator in the *AWS End User Messaging SMS User Guide*.

Design with redundancy in mind

For mission-critical messaging programs, we recommend that you configure Amazon SNS in more than one AWS Region. Amazon SNS is available in several AWS Regions. For a complete list of Regions where Amazon SNS is available, see the AWS General Reference.

The phone numbers that you use for SMS messages—including short codes, long codes, toll-free numbers, and 10DLC numbers—can't be replicated across AWS Regions. As a result, in order to use Amazon SNS in multiple Regions, you must request separate phone numbers in each Region where you want to use Amazon SNS. For example, if you use a short code to send text messages to recipients in the United States, you need to request separate short codes in each AWS Region that you plan to use.

In some countries, you can also use multiple types of phone numbers for added redundancy. For example, in the United States, you can request short codes, 10DLC numbers, and toll-free numbers. Each of these phone number types takes a different route to the recipient. Having multiple phone number types available—either in the same AWS Region or spread across multiple AWS Regions—provides an additional layer of redundancy, which can help improve resiliency.

SMS limits and restrictions

For SMS limits and restrictions, see SMS and MMS limits and restrictions in the *AWS End User Messaging SMS User Guide*.

Managing opt out keywords

SMS recipients can use their devices to opt out of messages by replying with a keyword. For more information, see Opting out of receiving SMS messages.

CreatePool

Use the `CreatePool` API action to create a new pool and associate a specified origination identity to the pool. For more information, see CreatePool in *AWS End User Messaging SMS API Reference*.

PutKeyword

Use the `PutKeyword` API action to create or update a keyword configuration on an origination phone number or pool. For more information, see PutKeyword in *AWS End User Messaging SMS API Reference*.

Managing number settings

To manage settings for the dedicated short codes and long codes that you requested from AWS Support and assigned to your account, see Change a phone number's capabilities with the AWS CLI in *AWS End User Messaging SMS*.

SMS character limits in Amazon SNS

A single SMS message can contain up to 140 bytes of information. The number of characters you can include in a single SMS message depends on the type of characters the message contains.

If your message only uses characters in the GSM 03.38 character set, also known as the GSM 7-bit alphabet, it can contain up to 160 characters. If your message contains any characters that are outside the GSM 03.38 character set, it can have up to 70 characters. When you send an SMS message, Amazon SNS automatically determines the most efficient encoding to use.

When a message contains more than the maximum number of characters, the message is split into multiple parts. When messages are split into multiple parts, each part contains additional information about the message part that precedes it. When the recipient's device receives message parts that are separated in this way, it uses this additional information to ensure that all of the message parts are displayed in the correct order. Depending on the recipient's mobile carrier and device, multiple messages might be displayed as a single message, or as a sequence of separate messages. As a result the number of characters in each message part is reduced to 153 (for messages that only contain GSM 03.38 characters) or 67 (for messages that contain other characters). You can estimate how many message parts your message contains before you send it by using SMS length calculator tools, several of which are available online. The maximum supported size of any message is 1600 GSM characters or 630 non-GSM characters. For more information about throughput and message size, see SMS character limits in Amazon Pinpoint in the *Amazon Pinpoint User Guide*.

To view the number of message parts for each message that you send, you should first enable Event stream settings. When you do, Amazon SNS produces an `_SMS.SUCCESS` event when the message is delivered to the recipient's mobile provider. The `_SMS.SUCCESS` event record contains an attribute called `attributes.number_of_message_parts`. This attribute specifies the number of message parts that the message contained.

> ⚠ **Important**
>
> When you send a message that contains more than one message parts, you're charged for the number of message parts contained in the message.

GSM 03.38 character set

The following table lists all of the characters that are present in the GSM 03.38 character set. If you send a message that only includes the characters shown in the following table, then the message can contain up to 160 characters.

GSM 03.38 standard characters												
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
à	Å	å	Ä	ä	Ç	É	é	è	ì	Ñ	ñ	ò
Ø	ø	Ö	ö	ù	Ü	ü	Æ	æ	ß	0	1	2
3	4	5	6	7	8	9	&	*	@	:	,	¤

GSM 03.38 standard characters												
$	=	!	>	#	-	¡	¿	(	<	%	.	+
£	?	"	)	§	;	'	/	_	¥	Δ	Φ	Γ
Λ	Ω	Π	Ψ	Σ	Θ	Ξ						

The GSM 03.38 character set includes several symbols in addition to those shown in the preceding table. However, each of these characters is counted as two characters because it also includes an invisible escape character:

- ^
- {
- }
- \
- [
-]
- ~
- |
- €

Finally, the GSM 03.38 character set also includes the following non-printed characters:

- A space character.
- A line feed control, which signifies the end of one line of text and the beginning of another.
- A carriage return control, which moves to the beginning of a line of text (usually following a line feed character).
- An escape control, which is automatically added to the characters in the preceding list.

Example messages

This section contains several example SMS messages. For each example, this section shows the total number of characters, as well as the number of message parts for the message.

Example 1: A long message that only contains characters in the GSM 03.38 alphabet

The following message only contains characters that are in the GSM 03.38 alphabet.

```
Hello Carlos. Your Example Corp. bill of $100 is now available. Autopay is
scheduled for next Thursday, April 9. To view the details of your bill, go
to https://example.com/bill1.
```

The preceding message contains 180 characters, so it has to be split into multiple message parts. When a message is split into multiple message parts, each part can contain 153 GSM 03.38 characters. As a result, this message is sent as 2 message parts.

Example 2: A message that contains multi-byte characters

The following message contains several Chinese characters, all of which are outside of the GSM 03.38 alphabet.

```
###################################################·####1994#7#########
```

The preceding message contains 71 characters. However, because almost all of the characters in the message are outside of the GSM 03.38 alphabet, it's sent as two message parts. Each of these message parts can contain a maximum of 67 characters.

Example 3: A message that contains a single non-GSM character

The following message contains a single character that isn't part of the GSM 03.38 alphabet. In this example, the character is a closing single quote (’), which is a different character from a regular apostrophe ('). Word processing applications such as Microsoft Word often automatically replace apostrophes with closing single quotes. If you draft your SMS messages in Microsoft Word and paste them into Amazon SNS, you should remove these special characters and replace them with apostrophes.

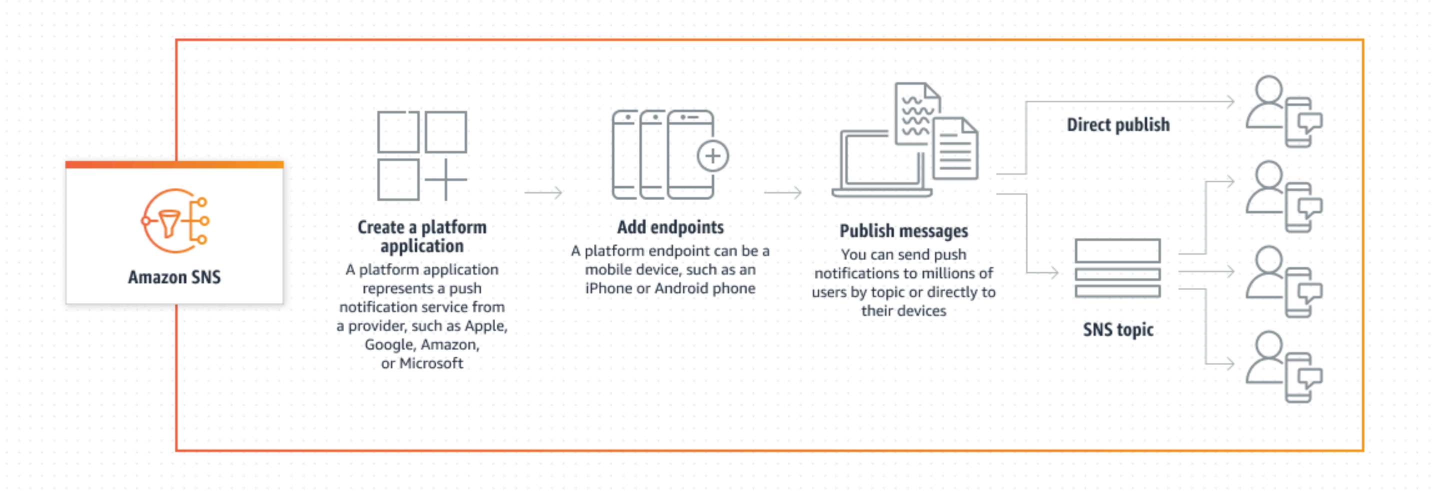
```
John: Your appointment with Dr. Salazar’s office is scheduled for next
Thursday at 4:30pm. Reply YES to confirm, NO to reschedule.
```

The preceding message contains 130 characters. However, because it contains the closing single quote character, which isn't part of the GSM 03.38 alphabet, it's sent as two message parts.

If you replace the closing single quote character in this message with an apostrophe (which is part of the GSM 03.38 alphabet), then the message is sent as a single message part.

Sending mobile push notifications with Amazon SNS

You can use Amazon SNS to send push notification messages directly to apps on mobile devices. Push notification messages sent to a mobile endpoint can appear in the mobile app as message alerts, badge updates, or sound alerts.


Amazon SNS
Create a platform application
A platform application represents a push notification service from a provider, such as Apple, Google, Amazon, or Microsoft
Add endpoints
A platform endpoint can be a mobile device, such as an iPhone or Android phone
Publish messages
You can send push notifications to millions of users by topic or directly to their devices
Direct publish
SNS topic

Topics

- How Amazon SNS user notifications work
- Setting up push notifications with Amazon SNS
- Setting up a mobile app in Amazon SNS
- Using Amazon SNS for mobile push notifications
- Amazon SNS mobile app attributes
- Amazon SNS application event notifications for mobile applications
- Mobile push API actions
- Common Amazon SNS mobile push API errors
- Using the Amazon SNS time to live message attribute for mobile push notifications
- Amazon SNS mobile application supported Regions
- Best practices for managing Amazon SNS mobile push notifications

How Amazon SNS user notifications work

You send push notification messages to both mobile devices and desktops using one of the following supported push notification services:

- Amazon Device Messaging (ADM)
- Apple Push Notification Service (APNs) for both iOS and Mac OS X
- Baidu Cloud Push (Baidu)
- Firebase Cloud Messaging (FCM)
- Microsoft Push Notification Service for Windows Phone (MPNS)
- Windows Push Notification Services (WNS)

Push notification services, such as APNs and FCM, maintain a connection with each app and associated mobile device registered to use their service. When an app and mobile device register, the push notification service returns a device token. Amazon SNS uses the device token to create a mobile endpoint, to which it can send direct push notification messages. In order for Amazon SNS to communicate with the different push notification services, you submit your push notification service credentials to Amazon SNS to be used on your behalf. For more information, see Setting up push notifications with Amazon SNS.

In addition to sending direct push notification messages, you can also use Amazon SNS to send messages to mobile endpoints subscribed to a topic. The concept is the same as subscribing other endpoint types, such as Amazon SQS, HTTP/S, email, and SMS, to a topic, as described in What is Amazon SNS?. The difference is that Amazon SNS communicates using the push notification services in order for the subscribed mobile endpoints to receive push notification messages sent to the topic.

Setting up push notifications with Amazon SNS

1. Obtain the credentials and device token for the mobile platforms that you want to support.
2. Use the credentials to create a platform application object (`PlatformApplicationArn`) using Amazon SNS. For more information, see Creating an Amazon SNS platform application.
3. Use the returned credentials to request a device token for your mobile app and device from the push notification service. The token you receive represents your mobile app and device.
4. Use the device token and the `PlatformApplicationArn` to create a platform endpoint object (`EndpointArn`) using Amazon SNS. For more information, see Setting up an Amazon SNS platform endpoint for mobile notifications.
5. Use the `EndpointArn` to publish a message to an app on a mobile device. For more information, see Direct Amazon SNS mobile device messaging and the Publish API in the Amazon Simple Notification Service API Reference.

Setting up a mobile app in Amazon SNS

This topic describes how to set up mobile applications in the AWS Management Console using the information described in Prerequisites for Amazon SNS user notifications.

Prerequisites for Amazon SNS user notifications

To begin using Amazon SNS mobile push notifications, you'll need the following:

- A set of credentials for connecting to one of the supported push notification services: ADM, APNs, Baidu, FCM, MPNS, or WNS.
- A device token or registration ID for the mobile app and device.
- Amazon SNS configured to send push notification messages to the mobile endpoints.
- A mobile app that is registered and configured to use one of the supported push notification services.

Registering your application with a push notification service requires several steps. Amazon SNS needs some of the information you provide to the push notification service in order to send direct push notification messages to the mobile endpoint. Generally speaking, you need the required credentials for connecting to the push notification service, a device token or registration ID (representing your mobile device and mobile app) received from the push notification service, and the mobile app registered with the push notification service.

The exact form the credentials take differs between mobile platforms, but in every case, these credentials must be submitted while making a connection to the platform. One set of credentials is issued for each mobile app, and it must be used to send a message to any instance of that app.

The specific names will vary depending on which push notification service is being used. For example, when using APNs as the push notification service, you need a *device token*. Alternatively, when using FCM, the device token equivalent is called a *registration ID*. The *device token* or *registration ID* is a string that is sent to the application by the operating system of the mobile device. It uniquely identifies an instance of a mobile app running on a particular mobile device and can be thought of as unique identifiers of this app-device pair.

Amazon SNS stores the credentials (plus a few other settings) as a platform application resource. The device tokens (again with some extra settings) are represented as objects called *platform endpoints*. Each platform endpoint belongs to one specific platform application, and every platform endpoint can be communicated with using the credentials that are stored in its corresponding platform application.

The following sections include the prerequisites for each of the supported push notification services. Once you've obtained the prerequisite information, you can send a push notification message using the AWS Management Console or the Amazon SNS mobile push APIs. For more information, see Setting up push notifications with Amazon SNS.

Creating an Amazon SNS platform application

To send notifications from Amazon SNS to mobile endpoints—whether directly or through subscriptions to a topic—you must first create a platform application. After registering the app with AWS, you need to create an endpoint for both the app and the mobile device. This endpoint allows Amazon SNS to send messages to the device.

To create a platform application

1. Sign in to the Amazon SNS console.
2. In the navigation pane, select **Push notifications**.
3. In the **Platform applications** section, choose **Create platform application**.
4. Choose your **AWS Region**. For a list of AWS Regions where you can create mobile applications, see Amazon SNS mobile application supported Regions.
5. Enter the following **application details**:
   - **Application name** – Provide a **name** for your platform application. The name must be between 1 and 256 characters and can contain uppercase and lowercase letters, numbers, underscores, hyphens, and periods.
   - **Push notification platform** – Select the appropriate **notification service** that the app is registered with (For example, Apple Push Notification Service (APNs), Firebase Cloud Messaging (FCM)).
6. Depending on the platform you selected, you'll need to provide specific credentials:
   - For **APNs** (Apple Push Notification Service) – Choose between **token-based** or **certificate-based** authentication.
     - For token-based authentication, upload a **.p8 file** (generated via Keychain Access).
     - For certificate-based authentication, upload a **.p12 file** (also exported from Keychain Access).

- For **FCM** (Firebase Cloud Messaging) – Enter the **Server key** from Firebase Console.
- For **other platforms** (such as ADM or GCM) – Enter the respective **API keys** or **credentials**.

7. After entering the necessary details, choose **Create platform application**. This action registers the app with Amazon SNS and creates the corresponding platform application object.
8. Upon creation, Amazon SNS generates and returns a `PlatformApplicationArn` (Amazon Resource Name). This ARN uniquely identifies your platform application and is used when creating endpoints for mobile devices.

Setting up an Amazon SNS platform endpoint for mobile notifications

When an app and mobile device register with a push notification service (such as APNs or Firebase Cloud Messaging), the push notification service returns a device token. Amazon SNS uses this device token to create a platform endpoint, which acts as a target for sending direct push notification messages to the app on the device. The platform endpoint serves as a bridge, routing messages sent by Amazon SNS to the push notification service for delivery to the corresponding mobile device. For more information, see Prerequisites for Amazon SNS user notifications and Setting up push notifications with Amazon SNS.

Understanding device tokens and platform endpoints

A device token uniquely identifies a mobile device registered with a push notification service (for example, APNs, Firebase Cloud Messaging). When an app registers with the push notification service, it generates a device token specific to that app and device. Amazon SNS uses this device token to create a platform endpoint within the corresponding platform application.

The platform endpoint allows Amazon SNS to send push notification messages to the device through the push notification service, maintaining the connection between your app and the user's device.

Create a platform endpoint

To push notifications to an app with Amazon SNS, that app's device token must first be registered with Amazon SNS by calling the create platform endpoint action. This action takes the Amazon Resource Name (ARN) of the platform application and the device token as parameters and returns the ARN of the created platform endpoint.

The `CreatePlatformEndpoint` action does the following:

- If the platform endpoint already exists, do not create it again. Return to the caller the ARN of the existing platform endpoint.
- If the platform endpoint with the same device token but different settings already exists, do not create it again. Throw an exception to the caller.
- If the platform endpoint does not exist, create it. Return to the caller the ARN of the newly-created platform endpoint.

You should not call the create platform endpoint action immediately every time an app starts, because this approach does not always provide a working endpoint. This can happen, for example, when an app is uninstalled and reinstalled on the same device and the endpoint for it already exists but is disabled. A successful registration process should accomplish the following:

1. Ensure a platform endpoint exists for this app-device combination.
2. Ensure the device token in the platform endpoint is the latest valid device token.
3. Ensure the platform endpoint is enabled and ready to use.

Pseudo code

The following pseudo code describes a recommended practice for creating a working, current, enabled platform endpoint in a wide variety of starting conditions. This approach works whether this is a first time the app is being registered or not, whether the platform endpoint for this app already exists, and whether the platform endpoint is enabled, has the correct device token, and so on. It is safe to call it multiple times in a row, as it will not create duplicate platform endpoints or change an existing platform endpoint if it is already up to date and enabled.

```
retrieve the latest device token from the mobile operating system
if (the platform endpoint ARN is not stored)
  # this is a first-time registration
  call create platform endpoint
  store the returned platform endpoint ARN
endif

call get endpoint attributes on the platform endpoint ARN

if (while getting the attributes a not-found exception is thrown)
  # the platform endpoint was deleted
  call create platform endpoint with the latest device token
  store the returned platform endpoint ARN
```

```
else
  if (the device token in the endpoint does not match the latest one) or
      (GetEndpointAttributes shows the endpoint as disabled)
    call set endpoint attributes to set the latest device token and then enable the
 platform endpoint
  endif
endif
```

This approach can be used any time the app wants to register or re-register itself. It can also be used when notifying Amazon SNS of a device token change. In this case, you can just call the action with the latest device token value. Some points to note about this approach are:

- There are two cases where it may call the create platform endpoint action. It may be called at the very beginning, where the app does not know its own platform endpoint ARN, as happens during a first-time registration. It is also called if the initial `GetEndpointAttributes` action call fails with a not-found exception, as would happen if the application knows its endpoint ARN but it was deleted.
- The `GetEndpointAttributes` action is called to verify the platform endpoint's state even if the platform endpoint was just created. This happens when the platform endpoint already exists but is disabled. In this case, the create platform endpoint action succeeds but does not enable the platform endpoint, so you must double-check the state of the platform endpoint before returning success.

AWS SDK example

The following code shows how to implement the previous pseudo code using the Amazon SNS clients that are provided by the AWS SDKs.

To use an AWS SDK, you must configure it with your credentials. For more information, see The shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

CLI

AWS CLI

To create a platform application endpoint

The following `create-platform-endpoint` example creates an endpoint for the specified platform application using the specified token.

```
aws sns create-platform-endpoint \
    --platform-application-arn arn:aws:sns:us-west-2:123456789012:app/GCM/
MyApplication \
    --token EXAMPLE12345...
```

Output:

```
{
      "EndpointArn": "arn:aws:sns:us-west-2:1234567890:endpoint/GCM/
MyApplication/12345678-abcd-9012-efgh-345678901234"
}
```

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointRequest;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * In addition, create a platform application using the AWS Management Console.
 * See this doc topic:
 *
 * https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-register.html
 *
 * Without the values created by following the previous link, this code examples
 * does not work.
 */

public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = """

            Usage:     <token> <platformApplicationArn>

            Where:
               token - The device token or registration ID of the mobile device.
 This is a unique
               identifier provided by the device platform (e.g., Apple Push
 Notification Service (APNS) for iOS devices, Firebase Cloud Messaging (FCM)
               for Android devices) when the mobile app is registered to receive
 push notifications.

               platformApplicationArn - The ARN value of platform application.
 You can get this value from the AWS Management Console.\s

            """;

        if (args.length != 2) {
            System.out.println(usage);
            return;
        }

        String token = args[0];
        String platformApplicationArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        createEndpoint(snsClient, token, platformApplicationArn);
    }
    public static void createEndpoint(SnsClient snsClient, String token, String
 platformApplicationArn) {
        System.out.println("Creating platform endpoint with token " + token);
        try {
            CreatePlatformEndpointRequest endpointRequest =
 CreatePlatformEndpointRequest.builder()
```

```
                .token(token)
                .platformApplicationArn(platformApplicationArn)
                .build();

            CreatePlatformEndpointResponse response =
 snsClient.createPlatformEndpoint(endpointRequest);
            System.out.println("The ARN of the endpoint is " +
 response.endpointArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
}
```

For more information, see Mobile push API actions.

Troubleshooting

Repeatedly calling create platform endpoint with an outdated device token

Especially for FCM endpoints, you may think it is best to store the first device token the application is issued and then call the create platform endpoint with that device token every time on application start-up. This may seem correct since it frees the app from having to manage the state of the device token and Amazon SNS will automatically update the device token to its latest value. However, this solution has a number of serious issues:

- Amazon SNS relies on feedback from FCM to update expired device tokens to new device tokens. FCM retains information about old device tokens for some time, but not indefinitely. Once FCM forgets about the connection between the old device token and the new device token, Amazon SNS will no longer be able to update the device token stored in the platform endpoint to its correct value; it will just disable the platform endpoint instead.
- The platform application will contain multiple platform endpoints corresponding to the same device token.
- Amazon SNS imposes a quota on the number of platform endpoints that can be created starting with the same device token. Eventually, the creation of new endpoints will fail with an invalid parameter exception and the following error message: "This endpoint is already registered with a different token."

For more information on managing FCM endpoints, see Amazon SNS management of Firebase Cloud Messaging endpoints.

Re-enabling a platform endpoint associated with an invalid device token

When a mobile platform (such as APNs or FCM) informs Amazon SNS that the device token used in the publish request was invalid, Amazon SNS disables the platform endpoint associated with that device token. Amazon SNS will then reject subsequent publishes to that device token. While you may think it is best to simply re-enable the platform endpoint and keep publishing, in most situations doing this will not work: the messages that are published do not get delivered and the platform endpoint becomes disabled again soon afterward.

This is because the device token associated with the platform endpoint is genuinely invalid. Deliveries to it cannot succeed because it no longer corresponds to any installed app. The next time it is published to, the mobile platform will again inform Amazon SNS that the device token is invalid, and Amazon SNS will again disable the platform endpoint.

To re-enable a disabled platform endpoint, it needs to be associated with a valid device token (with a set endpoint attributes action call) and then enabled. Only then will deliveries to that platform endpoint become successful. The only time re-enabling a platform endpoint without updating its device token will work is when a device token associated with that endpoint used to be invalid but then became valid again. This can happen, for example, when an app was uninstalled and then re-installed on the same mobile device and receives the same device token. The approach presented above does this, making sure to only re-enable a platform endpoint after verifying that the device token associated with it is the most current one available.

Integrating device tokens with Amazon SNS for mobile notifications

When you first register an app and mobile device with a notification service, such as Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM), device tokens or registration IDs are returned by the service. These tokens/IDs are added to Amazon SNS to create an endpoint for the app and device, using the `PlatformApplicationArn` API. Once the endpoint is created, an `EndpointArn` is returned, which Amazon SNS uses to direct notifications to the correct app/device.

You can add device tokens or registration IDs to Amazon SNS in the following ways:

- Manually add a single token via the AWS Management Console
- Upload several tokens using the `CreatePlatformEndpoint` API

- Register tokens for future devices

To manually add a device token or registration ID

1. Sign in to the Amazon SNS console.
2. In the navigation pane, select **Push Notifications**.
3. In the **Platform applications** section, select your application, and then choose **Edit**. If you haven't already created a platform application, follow the Creating an Amazon SNS platform application guide to do so now.
4. Choose **Create Endpoint**.
5. In the **Endpoint Token** box, enter the **token** or **registration ID**, depending on the notification service you're using (for example, FCM registration ID).
6. (Optional) Enter additional data in the **User Data** field. This data must be UTF-8 encoded and less than 2 KB.
7. Choose **Create Endpoint**.

Once the endpoint is created, you can send messages directly to the mobile device or to mobile devices subscribed to an Amazon SNS topic.

To upload several tokens using the `CreatePlatformEndpoint` API

The following steps show how to use the sample Java app (`bulkupload` package) provided by AWS to upload several tokens (device tokens or registration IDs) to Amazon SNS. You can use this sample app to help you get started with uploading your existing tokens.

> **Note**
>
> The following steps use the Eclipse Java IDE. The steps assume you have installed the AWS SDK for Java and you have the AWS security credentials for your AWS account. For more information, see AWS SDK for Java. For more information about credentials, see AWS security credentials in the *IAM User Guide*.

1. Download and unzip the snsmobilepush.zip file.
2. Create a new **Java project** in Eclipse and import the `SNSSamples` folder to the project.

3. Download the OpenCSV library and add it to the build path.

4. In the `BulkUpload.properties` file, specify the following:

   - Your `ApplicationArn` (platform application ARN).
   - The absolute path to your CSV file containing the tokens.
   - Logging filenames for successful and failed tokens. For example, `goodTokens.csv` and `badTokens.csv`.
   - (Optional) A configuration for delimiter, quote character, and number of threads to use.

   Your completed `BulkUpload.properties` should look similar to the following:

   ```
   applicationarn: arn:aws:sns:us-west-2:111122223333:app/FCM/fcmpushapp
   csvfilename: C:\\mytokendirectory\\mytokens.csv
   goodfilename: C:\\mylogfiles\\goodtokens.csv
   badfilename: C:\\mylogfiles\\badtokens.csv
   delimiterchar: ','
   quotechar: '"'
   numofthreads: 5
   ```

5. Run the **BatchCreatePlatformEndpointSample.java** application to upload the tokens to Amazon SNS. Tokens uploaded successfully will be logged in `goodTokens.csv`, while malformed tokens will be logged in `badTokens.csv`.

To register tokens from devices for future app installations

You have two options for this process:

Use the Amazon Cognito service

Your mobile app can use temporary security credentials to create endpoints. Amazon Cognito is recommended to generate temporary credentials. For more information, see the *Amazon Cognito Developer Guide*

To track app registrations, use Amazon SNS events to receive notifications when new endpoint ARNs are created.

Alternatively, you can use the `ListEndpointByPlatformApplication` API to retrieve the list of registered endpoints.

Use a proxy server

If your app infrastructure already supports device registration on installation, you can use your server as a proxy. It will forward device tokens to Amazon SNS via the `CreatePlatformEndpoint` API.

The endpoint ARN created by Amazon SNS will be returned and can be stored by your server for future message publishing.

Amazon SNS Apple push notification authentication methods

You can authorize Amazon SNS to send push notifications to your iOS or macOS app by providing information that identifies you as the developer of the app. To authenticate, provide either a *key* or a *certificate* when creating a platform application, both of which you can get from your Apple Developer account.

Token signing key

A private signing key that Amazon SNS uses to sign Apple Push Notification Service (APNs) authentication tokens.

If you provide a signing key, Amazon SNS uses a token to authenticate with APNs for every push notification that you send. With your signing key, you can send push notifications to APNs production and sandbox environments.

Your signing key doesn't expire, and you can use the same signing key for multiple apps. For more information, see Communicate with APNs using authentication tokens in the **Developer Account Help** section of the Apple website.

Certificate

A TLS certificate that Amazon SNS uses to authenticate with APNs when you send push notifications. You obtain the certificate from your Apple Developer account.

Certificates expire after one year. When this happens, you must create a new certificate and provide it to Amazon SNS. For more information, see Establishing a Certificate-Based Connection to APNs on the Apple Developer website.

To manage APNs settings using the AWS Management Console

1. Sign in to the Amazon SNS console.

2. In the navigation pane, select **Push notifications**.
3. In the **Platform applications** section, select the **application** whose APNs settings you want to edit, and then choose **Edit**. If you haven't already created a platform application, follow the Creating an Amazon SNS platform application guide to do so now.
4. Choose **Edit** to modify the settings for your platform application.
5. In the**Authentication type** section, choose one of the following options:
   - **Token-based authentication** (recommended for modern APNs integrations)
   - **Certificate-based authentication** (older method)
6. Configure your **credentials** based on the authentication type:
   - **For token-based authentication:**
     - Upload the **.p8 file**, which is the authentication token signing key you downloaded from your Apple Developer account.
     - Enter the **Signing Key ID** that you find in your Apple Developer account. Navigate to **Certificates**, **IDs & Profiles**, **Keys**, and select the **key** you want to use.
     - Provide the **Team Identifier** from your Apple Developer account. You can find this on the Membership page.
     - Enter the **Bundle Identifier** assigned to your app. You can find this under Certificates, IDs and Profiles, App IDs.
   - **For certificate-based authentication:**
     - Upload the **.p12 file** for your TLS certificate. This file can be exported from Keychain Access on macOS after downloading the certificate from your Apple Developer account.
     - If you assigned a **password** to your .p12 certificate, enter it here.
7. After entering the necessary credentials, choose **Save changes** to update the settings.

Amazon SNS integration with Firebase Cloud Messaging authentication setup

This topic describes how to obtain the required FCM API (HTTP v1) credentials from Google to use with the AWS API, AWS CLI and the AWS Management Console.

⚠ **Important**

March 26, 2024 – Amazon SNS supports FCM HTTP v1 API for Apple devices and Webpush destinations. We recommend that you migrate your existing mobile push applications to the latest FCM HTTP v1 API on or before June 1, 2024 to avoid application disruption.
January 18, 2024 – Amazon SNS introduced support for FCM HTTP v1 API for mobile push notification delivery to Android devices.
June 20, 2023 – Google deprecated their Firebase Cloud Messaging (FCM) legacy HTTP API. Amazon SNS now supports delivery to all device types using FCM HTTP v1 API. We recommend that you migrate your existing mobile push applications to the latest FCM HTTP v1 API on or before June 1, 2024 to avoid disruption.

You can authorize Amazon SNS to send push notifications to your applications by providing information that identifies you as the developer of the app. To authenticate, provide either an **API key** or a **token** when creating a platform application. You can get the following information from your Firebase application console:

API Key

The API key is a credential used when calling Firebase's Legacy API. The FCM Legacy APIs will be removed by Google June 20, 2024. If you are currently using an API key as your platform credential, you can update the platform credential by selecting **Token** as the option, and uploading the associated JSON file for your Firebase application.

Token

A short lived access token is used when calling the HTTP v1 API. This is Firebase's suggested API for sending push notifications. In order to generate access tokens, Firebase provides developers a set of credentials in the form of a private key file (also referred to as a service.json file).

Prerequisite

You must obtain your FCM service.json credentials before you can begin managing FCM settings in Amazon SNS. To obtain your service.json credentials, see Migrate from legacy FCM APIs to HTTP v1 in the Google Firebase documentation.

Managing FCM settings using the CLI

You can create FCM push notifications using the AWS API. The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas in the *AWS General Reference Guide*.

To create an FCM push notification together with an Amazon SNS topic (AWS API)

When using **key** credentials, the `PlatformCredential` is `API key`. When using **token** credentials, the `PlatformCredential` is a JSON formatted private key file:

- `CreatePlatformApplication`

To retrieve an FCM credential type for an existing Amazon SNS topic (AWS API)

Retrieves the credential type `"AuthenticationMethod": "Token"`, or `"AuthenticationMethod": "Key"`:

- GetPlatformApplicationAttributes

To set an FCM attribute for an existing Amazon SNS topic (AWS API)

Sets the FCM attribute:

- SetPlatformApplicationAttributes

Managing FCM settings using the console

You can create FCM push notifications using the AWS Command Line Interface (CLI). The number and size of Amazon SNS resources in an AWS account are limited. For more information, see Amazon Simple Notification Service endpoints and quotas.

To create an FCM push notification together with an Amazon SNS topic (AWS CLI)

When using **key** credentials, the `PlatformCredential` is `API key`. When using **token** credentials, the `PlatformCredential` is a JSON formatted private key file. When using the AWS CLI, the file must be in string format and special characters must be ignored. To format the file correctly,Amazon SNS recommends using the following command: `` SERVICE_JSON=`jq @json <<< cat service.json` ``:

- create-platform-application

To retrieve an FCM credential type for an existing Amazon SNS topic (AWS CLI)

Retrieves the credential type `"AuthenticationMethod": "Token"`, or `"AuthenticationMethod": "Key"`:

- get-platform-application-attributes

To set an FCM attribute for an existing Amazon SNS topic (AWS CLI)

Sets the FCM attribute:

- set-platform-application-attributes

Managing FCM settings (console)

Use the following steps to enter and manage your Firebase Cloud Messaging (FCM) credentials in Amazon SNS.

1. Sign in to the Amazon SNS console.
2. In the navigation pane, select **Push Notifications**.
3. In the **Platform applications** section, select the **FCM platform application** whose credentials you want to edit, and then choose **Edit**.
4. In the **Firebase Cloud Messaging Credentials** section, choose one of the following options:

   - **Token-based authentication** (recommended method) – Upload the **private key file** (JSON) that you downloaded from the Firebase Console. This file contains the credentials needed to generate short-lived access tokens for FCM notifications. To get this file:

     1. Go to your Firebase application console.
     2. In the **Project Settings**, select **Cloud Messaging**.
     3. Download the **Private key** JSON file (for use in the token-based authentication method).

   - **API key authentication** – If you prefer to use the older API key authentication method, enter the **Google API key** in the provided field. To get this file:

     1. Go to your Firebase application console.
     2. In **Project Settings**, select **Cloud Messaging**.

3. Copy the **Server key** (API key) to use for sending notifications.

5. When you finish, choose **Save changes**.

Related topics

- Using Google Firebase Cloud Messaging v1 payloads in Amazon SNS

Amazon SNS management of Firebase Cloud Messaging endpoints

Managing and maintaining device tokens

You can ensure deliverability of your mobile application's push notifications by following these steps:

1. Store all device tokens, corresponding Amazon SNS endpoint ARNs, and timestamps on your application server.
2. Remove all stale tokens and delete the corresponding Amazon SNS endpoint ARNs.

Upon your app's initial start-up, you'll receive a device token (also referred to as registration token) for the device. This device token is minted by the device’s operating system, and is tied to your FCM application. Once you receive this device token, you can register it with Amazon SNS as a platform endpoint. We recommend that you store the device token, the Amazon SNS platform endpoint ARN, and the timestamp by saving the them to your application server, or another persistent store. To set-up your FCM application to retrieve and store device tokens, see Retrieve and store registration tokens in Google's *Firebase* documentation.

It's important that you maintain up-to-date tokens. Your user’s device tokens can change under the following conditions:

1. The mobile application is restored on a new device.
2. The user uninstalls or updates the application.
3. The user clears application data.

When your device token changes, we recommended that you update the corresponding Amazon SNS endpoint with the new token. This allows Amazon SNS to continue communication to the registered device. You can do this by implementing the following pseudo code within your mobile application. It describes a recommended practice for creating and maintaining enabled platform endpoints. This approach can be executed each time the mobile applications starts, or as a scheduled job in the background.

Pseudo code

Use the following FCM pseudo code to manage and maintain device tokens.

```
retrieve the latest token from the mobile OS
if (endpoint arn not stored)
    # first time registration
    call CreatePlatformEndpoint
    store returned endpoint arn
endif

call GetEndpointAttributes on the endpoint arn

if (getting attributes encountered NotFound exception)
    #endpoint was deleted
    call CreatePlatformEndpoint
    store returned endpoint arn
else
    if (token in endpoint does not match latest) or
        (GetEndpointAttributes shows endpoint as disabled)
        call SetEndpointAttributes to set the
                     latest token and enable the endpoint
    endif
endif
```

To learn more about token update requirements, see Update Tokens on a Regular Basis in Google's *Firebase* documentation.

Detecting invalid tokens

When a message is dispatched to an FCM v1 endpoint with an invalid device token, Amazon SNS will receive one of the following exceptions:

- `UNREGISTERED` (HTTP 404) – When Amazon SNS receives this exception, you will receive a delivery failure event with a `FailureType` of `InvalidPlatformToken`, and a `FailureMessage` of *Platform token associated with the endpoint is not valid*. Amazon SNS will disable your platform endpoint when a delivery fails with this exception.

- `INVALID_ARGUMENT` (HTTP 400) – When Amazon SNS receives this exception, it means that the device token or the message payload is invalid. For more information, see ErrorCode in Google's *Firebase* documentation.

Since `INVALID_ARGUMENT` can be returned in either of these cases, Amazon SNS will return a `FailureType` of `InvalidNotification`, and a `FailureMessage` of *Notification body is invalid*. When you receive this error, verify that your payload is correct. If it is correct, verify that the device token is up-to-date. Amazon SNS will not disable your platform endpoint when a delivery fails with this exception.

Another case where you will experience an `InvalidPlatformToken` delivery failure event is when the registered device token doesn't belong to the application attempting to send that message. In this case, Google will return a *SENDER_ID_MISMATCH* error. Amazon SNS will disable your platform endpoint when a delivery fails with this exception.

All observed error codes received from the FCM v1 API are available to you in CloudWatch when you set up delivery status logging for your application.

To receive delivery events for your application, see Available application events.

Removing stale tokens

Tokens are considered stale once message deliveries to the endpoint device start failing. Amazon SNS sets these stale tokens as disabled endpoints for your platform application. When you publish to a disabled endpoint, Amazon SNS will return a `EventDeliveryFailure` event with the `FailureType` of `EndpointDisabled`, and a `FailureMessage` of *Endpoint is disabled*. To receive delivery events for your application, see Available application events.

When you receive this error from Amazon SNS, you need to remove or update the stale token in your platform application.

Using Amazon SNS for mobile push notifications

This section describes how to send mobile push notifications.

Publishing to a topic

You can also use Amazon SNS to send messages to mobile endpoints subscribed to a topic. The concept is the same as subscribing other endpoint types, such as Amazon SQS, HTTP/S, email, and SMS, to a topic, as described in What is Amazon SNS?. The difference is that Amazon SNS communicates through notification services like Apple Push Notification Service (APNS) and Google Firebase Cloud Messaging (FCM). Through the notifications service, the subscribed mobile endpoints receive notifications sent to the topic.

Direct Amazon SNS mobile device messaging

You can send Amazon SNS push notification messages directly to an endpoint which represents an application on a mobile device.

To send a direct message

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Push notifications**.
3. On the **Mobile push notifications** page, in the **Platform applications** section, choose the name of the application, for example ***MyApp***.
4. On the ***MyApp*** page, in the **Endpoints** section, choose an endpoint and then choose **Publish message**.
5. On the **Publish message to endpoint** page, enter the message that will appear in the application on the mobile device and then choose **Publish message**.

   Amazon SNS sends the notification message to the platform notification service which, in turn, sends the message to the application.

Publishing Amazon SNS notifications with platform-specific payloads

You can use the AWS Management Console or Amazon SNS APIs to send custom messages with platform-specific payloads to mobile devices. For information about using the Amazon SNS APIs, see Mobile push API actions and the `SNSMobilePush.java` file in `snsmobilepush.zip`.

Sending JSON-formatted messages

When you send platform-specific payloads, the data must be formatted as JSON key-value pair strings, with the quotation marks escaped.

The following examples show a custom message for the FCM platform.

```
{
```

```
"GCM": "{\"fcmV1Message\": {\"message\": {\"notification\": {\"title\": \"Hello\",
 \"body\": \"This is a test.\"}, \"data\": {\"dataKey\": \"example\"}}}}"
}
```

Sending platform-specific messages

In addition to sending custom data as key-value pairs, you can send platform-specific key-value pairs.

The following example shows the inclusion of the FCM parameters `time_to_live` and `collapse_key` after the custom data key-value pairs in the FCM `data` parameter.

```
{
"GCM": "{\"fcmV1Message\": {\"message\": {\"notification\": {\"title\": \"TitleTest\",
 \"body\": \"Sample message for Android or iOS endpoints.\"}, \"data\":{\"time_to_live
\": 3600,\"collapse_key\":\"deals\"}}}}"
}
```

For a list of the key-value pairs supported by each of the push notification services supported in Amazon SNS, see the following:

> **Important**
>
> Amazon SNS now supports Firebase Cloud Messaging (FCM) HTTP v1 API for sending mobile push notifications to Android devices.
> March 26, 2024 – Amazon SNS supports FCM HTTP v1 API for Apple devices and Webpush destinations. We recommend that you migrate your existing mobile push applications to the latest FCM HTTP v1 API on or before June 1, 2024 to avoid application disruption.

- Payload Key Reference in the APNs documentation
- Firebase Cloud Messaging HTTP Protocol in the FCM documentation
- Send a Message in the ADM documentation

Sending messages to an application on multiple platforms

To send a message to an application installed on devices for multiple platforms, such as FCM and APNs, you must first subscribe the mobile endpoints to a topic in Amazon SNS and then publish the message to the topic.

The following example shows a message to send to subscribed mobile endpoints on APNs, FCM, and ADM:

```
{
  "default": "This is the default message which must be present when publishing a
 message to a topic. The default message will only be used if a message is not present
 for
one of the notification platforms.",
  "APNS": "{\"aps\":{\"alert\": \"Check out these awesome deals!\",\"url\":
\"www.amazon.com\"} }",
  "GCM": "{\"data\":{\"message\":\"Check out these awesome deals!\",\"url\":
\"www.amazon.com\"}}",
  "ADM": "{\"data\":{\"message\":\"Check out these awesome deals!\",\"url\":
\"www.amazon.com\"}}"
}
```

Sending messages to APNs as alert or background notifications

Amazon SNS can send messages to APNs as `alert` or `background` notifications (for more information, see Pushing Background Updates to Your App in the APNs documentation).

- An `alert` APNs notification informs the user by displaying an alert message, playing a sound, or adding a badge to your application’s icon.
- A `background` APNs notification wakes up or instructs your application to act upon the content of the notification, without informing the user.

Specifying custom APNs header values

We recommend specifying custom values for the `AWS.SNS.MOBILE.APNS.PUSH_TYPE` reserved message attribute using the Amazon SNS `Publish` API action, AWS SDKs, or the AWS CLI. The following CLI example sets `content-available` to `1` and `apns-push-type` to `background` for the specified topic.

```
aws sns publish \
--endpoint-url https://sns.us-east-1.amazonaws.com \
--target-arn arn:aws:sns:us-east-1:123456789012:endpoint/APNS_PLATFORM/MYAPP/1234a567-
bc89-012d-3e45-6fg7h890123i \
--message '{"APNS_PLATFORM":"{\"aps\":{\"content-available\":1}}"}' \
--message-attributes '{ \
  "AWS.SNS.MOBILE.APNS.TOPIC":
{"DataType":"String","StringValue":"com.amazon.mobile.messaging.myapp"}, \
  "AWS.SNS.MOBILE.APNS.PUSH_TYPE":{"DataType":"String","StringValue":"background"}, \
  "AWS.SNS.MOBILE.APNS.PRIORITY":{"DataType":"String","StringValue":"5"}}' \
--message-structure json
```

> **Note**
>
> Ensure that the JSON structure is valid. Add a comma after each key-value pair, except the last one.

Inferring the APNs push type header from the payload

If you don't set the `apns-push-type` APNs header, Amazon SNS sets header to `alert` or `background` depending on the `content-available` key in the `aps` dictionary of your JSON-formatted APNs payload configuration.

> **Note**
>
> Amazon SNS is able to infer only `alert` or `background` headers, although the `apns-push-type` header can be set to other values.

- `apns-push-type` is set to `alert`
  - If the `aps` dictionary contains `content-available` set to `1` and *one or more keys* that trigger user interactions.
  - If the `aps` dictionary contains `content-available` set to `0` *or* if the `content-available` key is absent.
  - If the value of the `content-available` key isn't an integer or a Boolean.
- `apns-push-type` is set to `background`
  - If the `aps` dictionary *only* contains `content-available` set to `1` and *no other keys* that trigger user interactions.

    > **Important**
    >
    > If Amazon SNS sends a raw configuration object for APNs as a background-only notification, you must include `content-available` set to `1` in the `aps` dictionary.

Although you can include custom keys, the aps dictionary must not contain any keys that trigger user interactions (for example, alerts, badges, or sounds).

The following is an example raw configuration object.

```
{
  "APNS": "{\"aps\":{\"content-available\":1},\"Foo1\":\"Bar\",\"Foo2\":123}"
}
```

In this example, Amazon SNS sets the `apns-push-type` APNs header for the message to `background`. When Amazon SNS detects that the apn dictionary contains the `content-available` key set to `1`—and doesn't contain any other keys that can trigger user interactions—it sets the header to `background`.

Using Google Firebase Cloud Messaging v1 payloads in Amazon SNS

Amazon SNS supports using FCM HTTP v1 API to send notifications to Android, iOS, and Webpush destinations. This topic provides examples of the payload structure when publishing mobile push notifications using the CLI, or the Amazon SNS API.

You can include the following message types in your payload when sending an FCM notification:

- **Data message** – A data message is handled by your client app and contains custom key-value pairs. When constructing a data message, you must include the `data` key with a JSON object as the value, and then enter your custom key-value pairs.
- **Notification message** or **display message** – A notification message contains a predefined set of keys handled by the FCM SDK. These keys vary depending on the device type to which you are delivering. For more information on platform-specific notification keys, see the following:
  - Android notification keys
  - APNS notification keys
  - Webpush notification keys

For more information about FCM message types, see Message types in the in Google's *Firebase* documentation.

Using the FCM v1 payload structure to send messages

If you are creating an FCM application for the first time, or wish to take advantage of FCM v1 features, you can opt-in to send an FCM v1 formatted payload. To do this, you must include the top-level key `fcmV1Message`. For more information about constructing FCM v1 payloads, see Migrate from legacy FCM APIs to HTTP v1 and Customizing a message across platforms in Google's *Firebase* documentation.

FCM v1 example payload sent to Amazon SNS:

> **Note**
>
> The `GCM` key value used in the following example must be encoded as a String when publishing a notification using Amazon SNS.

```
{
  "GCM": "{
    \"fcmV1Message\": {
      \"validate_only\": false,
      \"message\": {
        \"notification\": {
          \"title\": \"string\",
          \"body\": \"string\"
        },
        \"data\": {
          \"dataGen\": \"priority message\"
        },
        \"android\": {
          \"priority\": \"high\",
          \"notification\": {
            \"body_loc_args\": [\"string\"],
            \"title_loc_args\": [\"string\"],
            \"sound\": \"string\",
            \"title_loc_key\": \"string\",
            \"title\": \"string\",
            \"body\": \"string\",
            \"click_action\": \"clicky_clacky\",
            \"body_loc_key\": \"string\"
          },
          \"data\": {
            \"dataAndroid\": \"priority message\"
```

```
     },
     \"ttl\": \"10023.32s\"
    },
    \"apns\": {
     \"payload\": {
      \"aps\": {
       \"alert\": {
        \"subtitle\": \"string\",
        \"title-loc-args\": [\"string\"],
        \"title-loc-key\": \"string\",
        \"loc-args\": [\"string\"],
        \"loc-key\": \"string\",
        \"title\": \"string\",
        \"body\": \"string\"
       },
       \"category\": \"Click\",
       \"content-available\": 0,
       \"sound\": \"string\",
       \"badge\": 5
      }
     }
    },
    \"webpush\": {
     \"notification\": {
      \"badge\": \"5\",
      \"title\": \"string\",
      \"body\": \"string\"
     },
     \"data\": {
      \"dataWeb\": \"priority message\"
     }
    }
   }
  }
 }"
}
```

When sending a JSON payload, be sure to include the `message-structure` attribute in your request, and set it to `json`.

CLI example:

```
aws sns publish --topic $TOPIC_ARN --message '{"GCM": "{\"fcmV1Message\": {\"message\":
{\"notification\":{\"title\":\"string\",\"body\":\"string\"},\"android\":{\"priority
\":\"high\",\"notification\":{\"title\":\"string\",\"body\":\"string\"},\"data\":
{\"customAndroidDataKey\":\"custom key value\"},\"ttl\":\"0s\"},\"apns\":{\"payload
\":{\"aps\":{\"alert\":{\"title\":\"string\", \"body\":\"string\"},\"content-
available\":1,\"badge\":5}}},\"webpush\":{\"notification\":{\"badge\":\"URL\",\"body
\":\"Test\"},\"data\":{\"customWebpushDataKey\":\"priority message\"}},\"data\":
{\"customGeneralDataKey\":\"priority message\"}}}}", "default": "{\"notification\":
 {\"title\": \"test\"}"}' --region $REGION --message-structure json
```

For more information on sending FCM v1 formatted payloads, see the following in Google's *Firebase* documentation:

- Migrate from legacy FCM APIs to HTTP v1
- About FCM messages
- REST Resource: projects.messages

Using the legacy payload structure to send messages to the FCM v1 API

When migrating to FCM v1, you don't have to change the payload structure that you were using for your legacy credentials. Amazon SNS transforms your payload into the new FCM v1 payload structure, and sends to Google.

Input message payload format:

```
{
  "GCM": "{\"notification\": {\"title\": \"string\", \"body\": \"string\",
 \"android_channel_id\": \"string\", \"body_loc_args\": [\"string\"], \"body_loc_key\":
 \"string\", \"click_action\": \"string\", \"color\": \"string\", \"icon\": \"string
\", \"sound\": \"string\", \"tag\": \"string\", \"title_loc_args\": [\"string\"],
 \"title_loc_key\": \"string\"}, \"data\": {\"message\": \"priority message\"}}"
}
```

Message sent to Google:

```
{
  "message": {
    "token": "***",
    "notification": {
      "title": "string",
      "body": "string"
    },
    "android": {
      "priority": "high",
      "notification": {
        "body_loc_args": [
          "string"
        ],
        "title_loc_args": [
          "string"
        ],
        "color": "string",
        "sound": "string",
        "icon": "string",
        "tag": "string",
        "title_loc_key": "string",
        "title": "string",
        "body": "string",
        "click_action": "string",
        "channel_id": "string",
        "body_loc_key": "string"
      },
      "data": {
        "message": "priority message"
      }
    },
    "apns": {
      "payload": {
        "aps": {
          "alert": {
            "title-loc-args": [
              "string"
            ],
            "title-loc-key": "string",
            "loc-args": [
              "string"
            ],
            "loc-key": "string",
            "title": "string",
            "body": "string"
          },
          "category": "string",
          "sound": "string"
        }
      }
    },
    "webpush": {
      "notification": {
        "icon": "string",
        "tag": "string",
        "body": "string",
        "title": "string"
      },
      "data": {
        "message": "priority message"
      }
    },
    "data": {
      "message": "priority message"
    }
  }
}
```

Potential risks

- Legacy to v1 mapping doesn't support the Apple Push Notification Service (APNS) `headers` or the `fcm_options` keys. If you'd like to use these fields, send an FCM v1 payload.
- In some cases, message headers are required by FCM v1 to send silent notifications to your APNs devices. If you are currently sending silent notifications to your APNs devices, they will not work with the legacy approach. Instead, we recommend using the FCM v1 payload to avoid unexpected issues. To find a list of APNs headers and what they are used for, see Communicating with APNs in the *Apple Developer Guide*.
- If you are using the `TTL` Amazon SNS attribute when sending your notification, it will only be updated in the `android` field. If you'd like to set the `TTL` APNS attribute, use the FCM v1 payload.
- The `android`, `apns`, and `webpush` keys will be mapped and populated with all relevant keys provided. For example, if you provide `title`, which is a key shared among all three platforms, the FCM v1 mapping will populate all three platforms with the title you provided.
- Some shared keys among platforms expect different value types. For example, the `badge` key passed to `apns` expects an integer value, while the `badge` key passed to `webpush` expects a String value. In cases where you provide the `badge` key, the FCM v1 mapping will only populate the key for which you provided a valid value.

FCM delivery failure events

The following table provides the Amazon SNS failure type that corresponds to the error/status codes received from Google for FCM v1 notification requests. All observed error codes received from the FCM v1 API are available to you in CloudWatch when you set-up delivery status logging for your application.

FCM error/status code	Amazon SNS failure type	Failure message	Cause and mitigation
`UNREGISTERED`	`InvalidPlatformToken`	Platform token associated with the endpoint is not valid.	The device token attached to your endpoint is stale or invalid. Amazon SNS disabled your endpoint. Update the Amazon SNS endpoint to the newest device token.
`INVALID_ARGUMENT`	`InvalidNotification`	Notification body is invalid.	The device token or message payload may be invalid. Verify that your message payload is valid. If the message payload is valid, update the Amazon SNS endpoint to the newest device token.
`SENDER_ID_MISMATCH`	`InvalidPlatformToken`	Platform token associated with the endpoint is not valid.	The platform application associated with the device token doesn't have permission to send to the device token. Verify that you are

FCM error/status code	Amazon SNS failure type	Failure message	Cause and mitigation
			using the correct FCM credentials in your Amazon SNS platform application.
`UNAVAILABLE`	`DependencyUnavailable`	Dependency is not available.	FCM couldn't process the request in time. All the retries executed by Amazon SNS have failed. You can store these messages in a dead-letter queue (DLQ) and redrive them later.
`INTERNAL`	`UnexpectedFailure`	Unexpected failure; please contact Amazon. Failure phrase [Internal Error].	The FCM server encountered an error while trying to process your request. All the retries executed by Amazon SNS have failed. You can store these messages in a dead-letter queue (DLQ) and redrive them later.

FCM error/status code	Amazon SNS failure type	Failure message	Cause and mitigation
`THIRD_PARTY_AUTH_ERROR`	`InvalidCredentials`	Platform application credentials are not valid.	A message targeted to an iOS device or a Webpush device could not be sent. Verify that your development and production credentials are valid.
`QUOTA_EXCEEDED`	`Throttled`	Request throttled by [gcm].	A message rate quota, device message rate quota, or topic message rate quota has been exceeded. For information on how to resolve this issue, see ErrorCode in the in Google's *Firebase* documentation.

FCM error/status code	Amazon SNS failure type	Failure message	Cause and mitigation
`PERMISSION_DENIED`	`InvalidNotification`	Notification body is invalid.	In the case of a `PERMISSION_DENIED` exception, the caller (your FCM application) doesn't have permission to execute the specified operation in the payload. Navigate to your FCM console, and verify your credentials have the required API actions enabled.

Amazon SNS mobile app attributes

Amazon Simple Notification Service (Amazon SNS) provides support to log the delivery status of push notification messages. After you configure application attributes, log entries will be sent to CloudWatch Logs for messages sent from Amazon SNS to mobile endpoints. Logging message delivery status helps provide better operational insight, such as the following:

- Know whether a push notification message was delivered from Amazon SNS to the push notification service.
- Identify the response sent from the push notification service to Amazon SNS.
- Determine the message dwell time (the time between the publish timestamp and just before handing off to a push notification service).

To configure application attributes for message delivery status, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

Configuring message delivery status attributes using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. On the navigation panel, point to **Mobile**, and then choose **Push notifications**.
3. From the **Platform applications** section, choose the application that contains the endpoints for which you want receive CloudWatch Logs.
4. Choose **Application Actions** and then choose **Delivery Status**.
5. On the **Delivery Status** dialog box, choose **Create IAM Roles**.

   You will then be redirected to the IAM console.
6. Choose **Allow** to give Amazon SNS write access to use CloudWatch Logs on your behalf.
7. Now, back on the **Delivery Status** dialog box, enter a number in the **Percentage of Success to Sample (0-100)** field for the percentage of successful messages sent for which you want to receive CloudWatch Logs.

   > **Note**
   >
   > After you configure application attributes for message delivery status, all failed message deliveries generate CloudWatch Logs.
8. Finally, choose **Save Configuration**. You will now be able to view and parse the CloudWatch Logs containing the message delivery status. For more information about using CloudWatch, see the CloudWatch Documentation.

Amazon SNS message delivery status CloudWatch log examples

After you configure message delivery status attributes for an application endpoint, CloudWatch Logs will be generated. Example logs, in JSON format, are shown as follows:

SUCCESS

```
{
  "status": "SUCCESS",
  "notification": {
    "timestamp": "2015-01-26 23:07:39.54",
    "messageId": "9655abe4-6ed6-5734-89f7-e6a6a42de02a"
  },
  "delivery": {
    "statusCode": 200,
    "dwellTimeMs": 65,
    "token": "Examplei7fFachkJ1xjlqT64RaBkcGHochmf1VQAr9k-
IBJtKjp7fedYPzEwT_Pq3Tu0lroqro1cwWJUvgkcPPYcaXCpPWmG3Bqn-
wiqIEzp5zZ7y_jsM0PKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw",
    "attempts": 1,
    "providerResponse": "{\"multicast_id\":5138139752481671853,\"success
\":1,\"failure\":0,\"canonical_ids\":0,\"results\":[{\"message_id\":
\"0:1422313659698010%d6ba8edff9fd7ecd\"}]}",
    "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/FCM/FCMPushApp/
c23e42de-3699-3639-84dd-65f84474629d"
  }
}
```

FAILURE

```
{
  "status": "FAILURE",
  "notification": {
    "timestamp": "2015-01-26 23:29:35.678",
    "messageId": "c3ad79b0-8996-550a-8bfa-24f05989898f"
  },
  "delivery": {
    "statusCode": 8,
    "dwellTimeMs": 1451,
    "token": "examp1e29z6j5c4df46f80189c4c83fjcgf7f6257e98542d2jt3395kj73",
    "attempts": 1,
    "providerResponse": "NotificationErrorResponse(command=8, status=InvalidToken,
 id=1, cause=null)",
    "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/APNS_SANDBOX/
APNSPushApp/986cb8a1-4f6b-34b1-9a1b-d9e9cb553944"
  }
}
```

For a list of push notification service response codes, see Platform response codes.

Configuring message delivery status attributes with the AWS SDKs

The AWS SDKs provide APIs in several languages for using message delivery status attributes with Amazon SNS.

The following Java example shows how to use the `SetPlatformApplicationAttributes` API to configure application attributes for message delivery status of push notification messages. You can use the following attributes for message delivery status: `SuccessFeedbackRoleArn`, `FailureFeedbackRoleArn`, and `SuccessFeedbackSampleRate`. The `SuccessFeedbackRoleArn` and `FailureFeedbackRoleArn` attributes are used to give Amazon SNS write access to use CloudWatch Logs on your behalf. The `SuccessFeedbackSampleRate` attribute is for specifying the sample rate percentage (0-100) of successfully delivered messages. After you configure the `FailureFeedbackRoleArn` attribute, then all failed message deliveries generate CloudWatch Logs.

```
SetPlatformApplicationAttributesRequest setPlatformApplicationAttributesRequest = new
 SetPlatformApplicationAttributesRequest();
Map<String, String> attributes = new HashMap<>();
attributes.put("SuccessFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("FailureFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("SuccessFeedbackSampleRate", "5");
setPlatformApplicationAttributesRequest.withAttributes(attributes);
setPlatformApplicationAttributesRequest.setPlatformApplicationArn("arn:aws:sns:us-
west-2:111122223333:app/FCM/FCMPushApp");
sns.setPlatformApplicationAttributes(setPlatformApplicationAttributesRequest);
```

For more information about the SDK for Java, see Getting Started with the AWS SDK for Java.

Platform response codes

The following is a list of links for the push notification service response codes:

Push notification service	Response codes
Amazon Device Messaging (ADM)	See Response Format in the ADM documentation.
Apple Push Notification Service (APNs)	See *HTTP/2 Response from APNs* in Communicating with APNs in the *Local and Remote Notification Programming Guide*.
Firebase Cloud Messaging (FCM)	See Downstream Message Error Response Codes in the Firebase Cloud Messaging documentation.

Push notification service	Response codes
Microsoft Push Notification Service for Windows Phone (MPNS)	See Push Notification Service Response Codes for Windows Phone 8 in the Windows 8 Development documentation.
Windows Push Notification Services (WNS)	See "Response codes" in Push Notification Service Request and Response Headers (Windows Runtime Apps) in the Windows 8 Development documentation.

Amazon SNS application event notifications for mobile applications

Amazon SNS provides support to trigger notifications when certain application events occur. You can then take some programmatic action on that event. Your application must include support for a push notification service such as Apple Push Notification Service (APNs), Firebase Cloud Messaging (FCM), and Windows Push Notification Services (WNS). You set application event notifications using the Amazon SNS console, AWS CLI, or the AWS SDKs.

Available application events

Application event notifications track when individual platform endpoints are created, deleted, and updated, as well as delivery failures. The following are the attribute names for the application events.

Attribute name	Notification trigger
`EventEndpointCreated`	A new platform endpoint is added to your application.
`EventEndpointDeleted`	Any platform endpoint associated with your application is deleted.
`EventEndpointUpdated`	Any of the attributes of the platform endpoints associated with your application are changed.
`EventDeliveryFailure`	A delivery to any of the platform endpoints associated with your application encounters a permanent failure.

Attribute name	Notification trigger
	Note To track delivery failures on the platform application side, subscribe to message delivery status events for the applicati on. For more information, see Using Amazon SNS Application Attributes for Message Delivery Status.

You can associate any attribute with an application which can then receive these event notifications.

Sending mobile push notifications

To send application event notifications, you specify a topic to receive the notifications for each type of event. As Amazon SNS sends the notifications, the topic can route them to endpoints that will take programmatic action.

Important

High-volume applications will create a large number of application event notifications (for example, tens of thousands), which will overwhelm endpoints meant for human use, such as email addresses, phone numbers, and mobile applications. Consider the following guidelines when you send application event notifications to a topic:

- Each topic that receives notifications should contain only subscriptions for programmatic endpoints, such as HTTP or HTTPS endpoints, Amazon SQS queues, or AWS Lambda functions.
- To reduce the amount of processing that is triggered by the notifications, limit each topic's subscriptions to a small number (for example, five or fewer).

You can send application event notifications using the Amazon SNS console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

AWS Management Console

1. Sign in to the Amazon SNS console.

2. On the navigation panel, choose **Mobile**, **Push notifications**.
3. On the **Mobile push notifications** page, in the **Platform applications** section, choose an application and then choose **Edit**.
4. Expand the **Event notifications** section.
5. Choose **Actions**, **Configure events**.
6. Enter the ARNs for topics to be used for the following events:
   - Endpoint Created
   - Endpoint Deleted
   - Endpoint Updated
   - Delivery Failure
7. Choose **Save changes**.

AWS CLI

Run the set-platform-application-attributes command.

The following example sets the same Amazon SNS topic for all four application events:

```
aws sns set-platform-application-attributes
--platform-application-arn arn:aws:sns:us-east-1:12345EXAMPLE:app/FCM/
MyFCMPlatformApplication
--attributes EventEndpointCreated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointDeleted="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointUpdated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventDeliveryFailure="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents"
```

AWS SDKs

Set application event notifications by submitting a `SetPlatformApplicationAttributes` request with the Amazon SNS API using an AWS SDK.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Amazon SNS with an AWS SDK.

Mobile push API actions

To use the Amazon SNS mobile push APIs, you must first meet the prerequisites for the push notification service, such as Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM). For more information about the prerequisites, see Prerequisites for Amazon SNS user notifications.

To send a push notification message to a mobile app and device using the APIs, you must first use the `CreatePlatformApplication` action, which returns a `PlatformApplicationArn` attribute. The `PlatformApplicationArn` attribute is then used by `CreatePlatformEndpoint`, which returns an `EndpointArn` attribute. You can then use the `EndpointArn` attribute with the `Publish` action to send a notification message to a mobile app and device, or you could use the `EndpointArn` attribute with the `Subscribe` action for subscription to a topic. For more information, see Setting up push notifications with Amazon SNS.

The Amazon SNS mobile push APIs are as follows:

`CreatePlatformApplication`

Creates a platform application object for one of the supported push notification services, such as APNs and FCM, to which devices and mobile apps may register. Returns a `PlatformApplicationArn` attribute, which is used by the `CreatePlatformEndpoint` action.

`CreatePlatformEndpoint`

Creates an endpoint for a device and mobile app on one of the supported push notification services. `CreatePlatformEndpoint` uses the `PlatformApplicationArn` attribute returned from the `CreatePlatformApplication` action. The `EndpointArn` attribute, which is returned when using `CreatePlatformEndpoint`, is then used with the `Publish` action to send a notification message to a mobile app and device.

`CreateTopic`

Creates a topic to which messages can be published.

`DeleteEndpoint`

Deletes the endpoint for a device and mobile app on one of the supported push notification services.

`DeletePlatformApplication`

Deletes a platform application object.

`DeleteTopic`

Deletes a topic and all its subscriptions.

`GetEndpointAttributes`

Retrieves the endpoint attributes for a device and mobile app.

`GetPlatformApplicationAttributes`

Retrieves the attributes of the platform application object.

`ListEndpointsByPlatformApplication`

Lists the endpoints and endpoint attributes for devices and mobile apps in a supported push notification service.

`ListPlatformApplications`

Lists the platform application objects for the supported push notification services.

`Publish`

Sends a notification message to all of a topic's subscribed endpoints.

`SetEndpointAttributes`

Sets the attributes for an endpoint for a device and mobile app.

`SetPlatformApplicationAttributes`

Sets the attributes of the platform application object.

`Subscribe`

Prepares to subscribe an endpoint by sending the endpoint a confirmation message. To actually create a subscription, the endpoint owner must call the ConfirmSubscription action with the token from the confirmation message.

`Unsubscribe`

Deletes a subscription.

Common Amazon SNS mobile push API errors

Errors that are returned by the Amazon SNS APIs for mobile push are listed in the following table. For more information about the Amazon SNS APIs for mobile push, see Mobile push API actions.

Error	Description	HTTPS status code	API Action
Application Name is null string	The required application name is set to null.	400	`CreatePlatformApplication`
Platform Name is null string	The required platform name is set to null.	400	`CreatePlatformApplication`
Platform Name is invalid	An invalid or out-of-range value was supplied for the platform name.	400	`CreatePlatformApplication`
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	`CreatePlatformApplication`
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	`CreatePlatformApplication`

Error	Description	HTTPS status code	API Action
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	`CreatePlatformApplication`
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	`CreatePlatformApplication`
APNs — Principal is not provided	The APNs principal , which is the SSL certificate, was not provided.	400	`CreatePlatformApplication`
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see [CreatePlatformApplication](#) in the Amazon Simple Notification Service API Reference .	400	`CreatePlatformApplication`
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	`CreatePlatformApplication`

Error	Description	HTTPS status code	API Action
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	`CreatePlatformApplication`
FCM — serverAPIKey is empty	The FCM credential, which is the API key, is empty.	400	`CreatePlatformApplication`
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	`CreatePlatformApplication`
FCM — serverAPIKey is invalid	The FCM credential, which is the API key, is invalid.	400	`CreatePlatformApplication`
ADM — clientsecret is not provided	The required client secret is not provided.	400	`CreatePlatformApplication`
ADM — clientsecret is a null string	The required string for the client secret is null.	400	`CreatePlatformApplication`
ADM — client_secret is empty string	The required string for the client secret is empty.	400	`CreatePlatformApplication`

Error	Description	HTTPS status code	API Action
ADM — client_secret is not valid	The required string for the client secret is not valid.	400	CreatePlatformApplication
ADM — client_id is empty string	The required string for the client ID is empty.	400	CreatePlatformApplication
ADM — clientId is not provided	The required string for the client ID is not provided.	400	CreatePlatformApplication
ADM — clientid is a null string	The required string for the client ID is null.	400	CreatePlatformApplication
ADM — client_id is not valid	The required string for the client ID is not valid.	400	CreatePlatformApplication
EventEndpointCreated has invalid ARN format	EventEndpointCreated has invalid ARN format.	400	CreatePlatformApplication
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	CreatePlatformApplication
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	CreatePlatformApplication
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure has invalid ARN format.	400	CreatePlatformApplication

Error	Description	HTTPS status code	API Action
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	`CreatePlatformApplication`
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	`CreatePlatformApplication`
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	`CreatePlatformApplication`
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	`CreatePlatformApplication`
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure is not an existing topic.	400	`CreatePlatformApplication`
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	`CreatePlatformApplication`
Platform ARN is invalid	Platform ARN is invalid.	400	`SetPlatformAttributes`
Platform ARN is valid but does not belong to the user	Platform ARN is valid but does not belong to the user.	400	`SetPlatformAttributes`

Error	Description	HTTPS status code	API Action
APNs — Principal is not a valid certificate	An invalid certificate was supplied for the APNs principal, which is the SSL certificate. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	`SetPlatformAttributes`
APNs — Principal is a valid cert but not in a .pem format	A valid certificate that is not in the .pem format was supplied for the APNs principal, which is the SSL certificate.	400	`SetPlatformAttributes`
APNs — Principal is an expired certificate	An expired certificate was supplied for the APNs principal, which is the SSL certificate.	400	`SetPlatformAttributes`
APNs — Principal is not an Apple issued certificate	A non-Apple issued certificate was supplied for the APNs principal, which is the SSL certificate.	400	`SetPlatformAttributes`
APNs — Principal is not provided	The APNs principal, which is the SSL certificate, was not provided.	400	`SetPlatformAttributes`

Error	Description	HTTPS status code	API Action
APNs — Credential is not provided	The APNs credential, which is the private key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference .	400	`SetPlatformAttributes`
APNs — Credential are not in a valid .pem format	The APNs credential, which is the private key, is not in a valid .pem format.	400	`SetPlatformAttributes`
FCM — serverAPIKey is not provided	The FCM credential, which is the API key, was not provided. For more information, see CreatePlatformApplication in the Amazon Simple Notification Service API Reference.	400	`SetPlatformAttributes`
FCM — serverAPIKey is a null string	The FCM credential, which is the API key, is null.	400	`SetPlatformAttributes`
ADM — clientId is not provided	The required string for the client ID is not provided.	400	`SetPlatformAttributes`

Error	Description	HTTPS status code	API Action
ADM — clientid is a null string	The required string for the client ID is null.	400	`SetPlatformAttributes`
ADM — clientsecret is not provided	The required client secret is not provided.	400	`SetPlatformAttributes`
ADM — clientsecret is a null string	The required string for the client secret is null.	400	`SetPlatformAttributes`
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	`SetPlatformAttributes`
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted has invalid ARN format.	400	`SetPlatformAttributes`
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated has invalid ARN format.	400	`SetPlatformAttributes`
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure has invalid ARN format.	400	`SetPlatformAttributes`
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure has invalid ARN format.	400	`SetPlatformAttributes`
EventEndpointCreated is not an existing Topic	EventEndpointCreated is not an existing topic.	400	`SetPlatformAttributes`

Error	Description	HTTPS status code	API Action
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted is not an existing topic.	400	`SetPlatformAttributes`
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated is not an existing topic.	400	`SetPlatformAttributes`
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure is not an existing topic.	400	`SetPlatformAttributes`
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure is not an existing topic.	400	`SetPlatformAttributes`
Platform ARN is invalid	The platform ARN is invalid.	400	`GetPlatformApplicationAttributes`
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	`GetPlatformApplicationAttributes`
Token specified is invalid	The specified token is invalid.	400	`ListPlatformApplications`
Platform ARN is invalid	The platform ARN is invalid.	400	`ListEndpointsByPlatformApplication`

Error	Description	HTTPS status code	API Action
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	`ListEndpointsByPlatformApplication`
Token specified is invalid	The specified token is invalid.	400	`ListEndpointsByPlatformApplication`
Platform ARN is invalid	The platform ARN is invalid.	400	`DeletePlatformApplication`
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	403	`DeletePlatformApplication`
Platform ARN is invalid	The platform ARN is invalid.	400	`CreatePlatformEndpoint`
Platform ARN is valid but does not belong to the user	The platform ARN is valid, but does not belong to the user.	404	`CreatePlatformEndpoint`
Token is not specified	The token is not specified.	400	`CreatePlatformEndpoint`
Token is not of correct length	The token is not the correct length.	400	`CreatePlatformEndpoint`
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	`CreatePlatformEndpoint`

Error	Description	HTTPS status code	API Action
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	`DeleteEndpoint`
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	`DeleteEndpoint`
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	`SetEndpointAttributes`
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	`SetEndpointAttributes`
Token is not specified	The token is not specified.	400	`SetEndpointAttributes`
Token is not of correct length	The token is not the correct length.	400	`SetEndpointAttributes`
Customer User data is too large	The customer user data cannot be more than 2048 bytes long in UTF-8 encoding.	400	`SetEndpointAttributes`
Endpoint ARN is invalid	The endpoint ARN is invalid.	400	`GetEndpointAttributes`
Endpoint ARN is valid but does not belong to the user	The endpoint ARN is valid, but does not belong to the user.	403	`GetEndpointAttributes`
Target ARN is invalid	The target ARN is invalid.	400	`Publish`

Error	Description	HTTPS status code	API Action
Target ARN is valid but does not belong to the user	The target ARN is valid, but does not belong to the user.	403	`Publish`
Message format is invalid	The message format is invalid.	400	`Publish`
Message size is larger than supported by protocol/end-service	The message size is larger than supported by the protocol/end-service.	400	`Publish`

Using the Amazon SNS time to live message attribute for mobile push notifications

Amazon Simple Notification Service (Amazon SNS) provides support for setting a *Time To Live (TTL)* message attribute for mobile push notifications messages. This is in addition to the existing capability of setting TTL within the Amazon SNS message body for the mobile push notification services that support this, such as Amazon Device Messaging (ADM) and Firebase Cloud Messaging (FCM) when sending to Android.

The TTL message attribute is used to specify expiration metadata about a message. This allows you to specify the amount of time that the push notification service, such as Apple Push Notification Service (APNs) or FCM, has to deliver the message to the endpoint. If for some reason (such as the mobile device has been turned off) the message is not deliverable within the specified TTL, then the message will be dropped and no further attempts to deliver it will be made. To specify TTL within message attributes, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

TTL message attributes for push notification services

The following is a list of the TTL message attributes for push notification services that you can use to set when using the AWS SDKs or query API:

Push notification service	TTL message attribute
Amazon Device Messaging (ADM)	`AWS.SNS.MOBILE.ADM.TTL`
Apple Push Notification Service (APNs)	`AWS.SNS.MOBILE.APNS.TTL`
Apple Push Notification Service Sandbox (APNs_SANDBOX)	`AWS.SNS.MOBILE.APNS_SANDBOX.TTL`
Baidu Cloud Push (Baidu)	`AWS.SNS.MOBILE.BAIDU.TTL`
Firebase Cloud Messaging (FCM when sending to Android)	`AWS.SNS.MOBILE.FCM.TTL`
Windows Push Notification Services (WNS)	`AWS.SNS.MOBILE.WNS.TTL`

Each of the push notification services handle TTL differently. Amazon SNS provides an abstract view of TTL over all the push notification services, which makes it easier to specify TTL. When you use the AWS Management Console to specify TTL (in seconds), you only have to enter the TTL value once and Amazon SNS will then calculate the TTL for each of the selected push notification services when publishing the message.

TTL is relative to the publish time. Before handing off a push notification message to a specific push notification service, Amazon SNS computes the dwell time (the time between the publish timestamp and just before handing off to a push notification service) for the push notification and passes the remaining TTL to the specific push notification service. If TTL is shorter than the dwell time, Amazon SNS won't attempt to publish.

If you specify a TTL for a push notification message, then the TTL value must be a positive integer, unless the value of `0` has a specific meaning for the push notification service—such as with APNs and FCM (when sending to Android). If the TTL value is set to `0` and the push notification service does not have a specific meaning for `0`, then Amazon SNS will drop the message. For more information about the TTL parameter set to `0` when using APNs, see *Table A-3 Item identifiers for remote notifications* in the Binary Provider API documentation.

Precedence order for determining TTL

The precedence that Amazon SNS uses to determine the TTL for a push notification message is based on the following order, where the lowest number has the highest priority:

1. Message attribute TTL
2. Message body TTL
3. Push notification service default TTL (varies per service)
4. Amazon SNS default TTL (4 weeks)

If you set different TTL values (one in message attributes and another in the message body) for the same message, then Amazon SNS will modify the TTL in the message body to match the TTL specified in the message attribute.

Specifying TTL using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Mobile**, **Push notifications**.
3. On the **Mobile push notifications** page, in the **Platform applications** section, choose an application.
4. On the ***`MyApplication`*** page, in the **Endpoints** section, choose an application endpoint and then choose **Publish message**.
5. In the **Message details** section, enter the TTL (the number of seconds that the push notification service has to deliver the message to the endpoint).
6. Choose **Publish message**.

Amazon SNS mobile application supported Regions

Currently, you can create mobile applications in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)

- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)
- AWS GovCloud (US-West)

Best practices for managing Amazon SNS mobile push notifications

This section describes best practices that might help you improve your customer engagement.

Endpoint management

Delivery issues might occur in situations were device tokens change due to a user's action on the device (for example, an app is re-installed on the device), or certificate updates affecting devices running on a particular iOS version. It is a recommended best practice by Apple to register with APNs each time your app launches.

Since the device token won't change each time an app is opened by a user, the idempotent `CreatePlatformEndpoint` API can be used. However, this can introduce duplicates for the same device in cases where the token itself is invalid, or if the endpoint is valid but disabled (for example, a mismatch of production and sandbox environments).

A device token management mechanism such as the one in the pseudo code can be used.

For information on managing and maintaining FCM v1 device tokens, see Amazon SNS management of Firebase Cloud Messaging endpoints.

Delivery status logging

To monitor push notification delivery status, we recommended you enable delivery status logging for your Amazon SNS platform application. This helps you troubleshoot delivery failures because the logs contain provider response codes returned from the push platform service. For details on enabling delivery status logging, see How do I access Amazon SNS topic delivery logs for push notifications?.

Event notifications

For managing endpoints in an event driven fashion, you can make use of the event notifications functionality. This allows the configured Amazon SNS topic to fanout events to the subscribers such as a Lambda function, for platform application events of endpoint creation, deletion, updates, and delivery failures.

Amazon SNS email subscription setup and management

You can subscribe an email address to an Amazon SNS topic using the AWS Management Console, AWS SDK for Java, or AWS SDK for .NET.

Notes

- Customization of the email message body is not supported. The email delivery feature is intended to provide internal system alerts, not marketing messages.
- Directly subscribing email endpoints is supported for standard topics only.
- Email delivery throughput is throttled. For more information, see Amazon SNS quotas.

Important

To prevent mailing list recipients from unsubscribing all recipients from Amazon SNS topic emails, see Set up an email subscription that requires authentication to unsubscribe from AWS Support.

Subscribing an email address to an Amazon SNS topic using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:

   a. For **Topic ARN**, choose the Amazon Resource Name (ARN) of a topic.

   b. For **Protocol**, choose **Email**.

   c. For **Endpoint**, enter the email address.

   d. (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see Amazon SNS subscription filter policies.

   e. (Optional) To enable payload-based filtering, configure `Filter Policy Scope` to `MessageBody`. For more information, see Amazon SNS subscription filter policy scope.

   f. (Optional) To configure a dead-letter queue for the subscription, expand the **Redrive policy (dead-letter queue)** section. For more information, see Amazon SNS dead-letter queues.

   g. Choose **Create subscription**.

   The console creates the subscription and opens the subscription's **Details** page.

You must confirm the subscription before the email address can start to receive messages.

To confirm a subscription

1. Check your email inbox and choose **Confirm subscription** in the email from Amazon SNS.
2. Amazon SNS opens your web browser and displays a subscription confirmation with your subscription ID.

Subscribing an email address to an Amazon SNS topic using an AWS SDK

To use an AWS SDK, you must configure it with your credentials. For more information, see The shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

The following code examples show how to use `Subscribe`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

Subscribe an email address to a topic.

```
        /// <summary>
        /// Creates a new subscription to a topic.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object, used
        /// to create an Amazon SNS subscription.</param>
        /// <param name="topicArn">The ARN of the topic to subscribe to.</param>
        /// <returns>A SubscribeResponse object which includes the subscription
        /// ARN for the new subscription.</returns>
        public static async Task<SubscribeResponse> TopicSubscribeAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            SubscribeRequest request = new SubscribeRequest()
            {
                TopicArn = topicArn,
                ReturnSubscriptionArn = true,
                Protocol = "email",
                Endpoint = "recipient@example.com",
            };

            var response = await client.SubscribeAsync(request);

            return response;
        }
```

Subscribe a queue to a topic with optional filters.

```
   /// <summary>
   /// Subscribe a queue to a topic with optional filters.
   /// </summary>
   /// <param name="topicArn">The ARN of the topic.</param>
   /// <param name="useFifoTopic">The optional filtering policy for the 
subscription.</param>
   /// <param name="queueArn">The ARN of the queue.</param>
   /// <returns>The ARN of the new subscription.</returns>
   public async Task<string> SubscribeTopicWithFilter(string topicArn, string? 
filterPolicy, string queueArn)
   {
       var subscribeRequest = new SubscribeRequest()
       {
           TopicArn = topicArn,
           Protocol = "sqs",
           Endpoint = queueArn
       };

       if (!string.IsNullOrEmpty(filterPolicy))
       {
           subscribeRequest.Attributes = new Dictionary<string, string> 
{ { "FilterPolicy", filterPolicy } };
       }

       var subscribeResponse = await 
_amazonSNSClient.SubscribeAsync(subscribeRequest);
       return subscribeResponse.SubscriptionArn;
   }
```

- For API details, see Subscribe in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to an email address.
/*!
  \param topicARN: An SNS topic Amazon Resource Name (ARN).
  \param emailAddress: An email address.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::subscribeEmail(const Aws::String &topicARN,
                                 const Aws::String &emailAddress,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("email");
    request.SetEndpoint(emailAddress);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                  << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
```

```
                 << std::endl;
    }

    return outcome.IsSuccess();
}
```

Subscribe a mobile application to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to a mobile app.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param endpointARN: The ARN for a mobile app or device endpoint.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SNS::subscribeApp(const Aws::String &topicARN,
                          const Aws::String &endpointARN,
                          const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("application");
    request.SetEndpoint(endpointARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                 << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
                 << std::endl;
    }
    return outcome.IsSuccess();
}
```

Subscribe a Lambda function to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to an AWS Lambda function.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param lambdaFunctionARN: The ARN for an AWS Lambda function.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::subscribeLambda(const Aws::String &topicARN,
                                  const Aws::String &lambdaFunctionARN,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("lambda");
    request.SetEndpoint(lambdaFunctionARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                  << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

Subscribe an SQS queue to a topic.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SNS::SNSClient snsClient(clientConfiguration);

            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);

            Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
 outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
 "."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
```

Subscribe with a filter to a topic.

```
        static const Aws::String TONE_ATTRIBUTE("tone");
        static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
 "serious",
                                                      "sincere"};

        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SNS::SNSClient snsClient(clientConfiguration);

            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
                              << "\"'s subscription to the topic \""
                              << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
```

```
               if (askYesNoQuestion(ostringstream.str())) {
                   Aws::String jsonPolicy = getFilterPolicyFromUser();
                   if (!jsonPolicy.empty()) {
                       filteringMessages = true;

                       std::cout << "This is the filter policy for this
subscription."
                                 << std::endl;
                       std::cout << jsonPolicy << std::endl;

                       request.AddAttributes("FilterPolicy", jsonPolicy);
                   }
                   else {
                       std::cout
                               << "Because you did not select any attributes, no
filter "
                               << "will be added to this subscription." <<
std::endl;
                   }
               }
           }  // if (isFifoTopic)
           Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

           if (outcome.IsSuccess()) {
               Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
               std::cout << "The queue '" << queueName
                         << "' has been subscribed to the topic '"
                         << "'" << topicName << "'" << std::endl;
               std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                         << std::endl;
               subscriptionARNS.push_back(subscriptionARN);
           }
           else {
               std::cerr << "Error with TopicsAndQueues::Subscribe. "
                         << outcome.GetError().GetMessage()
                         << std::endl;

               cleanUp(topicARN,
                       queueURLS,
                       subscriptionARNS,
                       snsClient,
                        sqsClient);

                return false;
            }

//! Routine that lets the user select attributes for a subscription filter
 policy.
/*!
 \sa getFilterPolicyFromUser()
 \return Aws::String: The filter policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::getFilterPolicyFromUser() {
    std::cout
            << "You can filter messages by one or more of the following \""
            << TONE_ATTRIBUTE << "\" attributes." << std::endl;

    std::vector<Aws::String> filterSelections;
    int selection;
    do {
        for (size_t j = 0; j < TONES.size(); ++j) {
            std::cout << "  " << (j + 1) << ". " << TONES[j]
                      << std::endl;
        }
        selection = askQuestionForIntRange(
                "Enter a number (or enter zero to stop adding more). ",
                0, static_cast<int>(TONES.size()));

        if (selection != 0) {
            const Aws::String &selectedTone(TONES[selection - 1]);
            // Add the tone to the selection if it is not already added.
            if (std::find(filterSelections.begin(),
                          filterSelections.end(),
                          selectedTone)
                == filterSelections.end()) {
                filterSelections.push_back(selectedTone);
            }
        }
    } while (selection != 0);

    Aws::String result;
    if (!filterSelections.empty()) {
        std::ostringstream jsonPolicyStream;
        jsonPolicyStream << "{ \"" << TONE_ATTRIBUTE << "\": [";
        for (size_t j = 0; j < filterSelections.size(); ++j) {
            jsonPolicyStream << "\"" << filterSelections[j] << "\"";
            if (j < filterSelections.size() - 1) {
                jsonPolicyStream << ",";
            }
        }
        jsonPolicyStream << "] }";

        result = jsonPolicyStream.str();
    }

    return result;
}
```

- For API details, see Subscribe in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To subscribe to a topic

The following `subscribe` command subscribes an email address to the specified topic.

```
aws sns subscribe \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
    --protocol email \
    --notification-endpoint my-email@example.com
```

Output:

```
{
    "SubscriptionArn": "pending confirmation"
}
```

- For API details, see Subscribe in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/gov2/workflows/topics_and_queues#code-examples).

Subscribe a queue to a topic with optional filters.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

- For API details, see Subscribe in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeEmail {
    public static void main(String[] args) {
        final String usage = """
                Usage:     <topicArn> <email>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   email - The email address to use.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String email = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subEmail(snsClient, topicArn, email);
        snsClient.close();
    }

    public static void subEmail(SnsClient snsClient, String topicArn, String
 email) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("email")
                    .endpoint(email)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN: " + result.subscriptionArn() +
 "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Subscribe an HTTP endpoint to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeHTTPS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn> <url>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   url - The HTTPS endpoint that you want to receive
 notifications.
                """;

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String url = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subHTTPS(snsClient, topicArn, url);
        snsClient.close();
    }

    public static void subHTTPS(SnsClient snsClient, String topicArn, String url)
 {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("https")
                    .endpoint(url)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN is " + result.subscriptionArn()
 + "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Subscribe a Lambda function to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeLambda {

    public static void main(String[] args) {

        final String usage = """

                Usage:    <topicArn> <lambdaArn>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   lambdaArn - The ARN of an AWS Lambda function.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String lambdaArn = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnValue = subLambda(snsClient, topicArn, lambdaArn);
        System.out.println("Subscription ARN: " + arnValue);
        snsClient.close();
    }
    public static String subLambda(SnsClient snsClient, String topicArn, String
 lambdaArn) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("lambda")
                    .endpoint(lambdaArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            return result.subscriptionArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see Subscribe in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 a subscription.
 * @param {string} emailAddress - The email address that is subscribed to the
 topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Subscribe a mobile application to a topic.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint
 is created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Subscribe a Lambda function to a topic.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
```

```
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Subscribe an SQS queue to a topic.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Subscribe with a filter to a topic.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute
 set to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see Subscribe in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
suspend fun subEmail(
    topicArnVal: String,
    email: String,
): String {
    val request =
        SubscribeRequest {
            protocol = "email"
            endpoint = email
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        return result.subscriptionArn.toString()
    }
}
```

Subscribe a Lambda function to a topic.

```
suspend fun subLambda(
    topicArnVal: String?,
    lambdaArn: String?,
) {
    val request =
        SubscribeRequest {
            protocol = "lambda"
            endpoint = lambdaArn
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        println(" The subscription Arn is ${result.subscriptionArn}")
    }
}
```

- For API details, see Subscribe in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Subscribe an HTTP endpoint to a topic.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;



/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'https';
$endpoint = 'https://';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see Subscribe in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def subscribe(topic, protocol, endpoint):
        """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
 subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.

        :param topic: The topic to subscribe to.
        :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
        :param endpoint: The endpoint that receives messages, such as a phone
 number
                         (in E.164 format) for SMS messages, or an email address
 for
                         email messages.
        :return: The newly added subscription.
        """
        try:
            subscription = topic.subscribe(
                Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True
            )
            logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
 topic.arn)
        except ClientError:
            logger.exception(
                "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
 topic.arn
            )
            raise
        else:
            return subscription
```

- For API details, see Subscribe in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
require 'aws-sdk-sns'
require 'logger'
```

```
# Represents a service for creating subscriptions in Amazon Simple Notification
 Service (SNS)
class SubscriptionService
  # Initializes the SubscriptionService with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Attempts to create a subscription to a topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param protocol [String] The subscription protocol (e.g., email)
  # @param endpoint [String] The endpoint that receives the notifications (email
 address)
  # @return [Boolean] true if subscription was successfully created, false
 otherwise
  def create_subscription(topic_arn, protocol, endpoint)
    @sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
 endpoint)
    @logger.info('Subscription created successfully.')
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while creating the subscription: #{e.message}")
    false
  end
end

# Main execution if the script is run directly
if $PROGRAM_NAME == __FILE__
  protocol = 'email'
  endpoint = 'EMAIL_ADDRESS' # Should be replaced with a real email address
  topic_arn = 'TOPIC_ARN'    # Should be replaced with a real topic ARN

  sns_client = Aws::SNS::Client.new
  subscription_service = SubscriptionService.new(sns_client)

  @logger.info('Creating the subscription.')
  unless subscription_service.create_subscription(topic_arn, protocol, endpoint)
    @logger.error('Subscription creation failed. Stopping program.')
    exit 1
  end
```

```
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see Subscribe in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;
    println!("Published message: {:?}", rsp);

    Ok(())
}
```

- For API details, see Subscribe in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
    TRY.
        oo_result = lo_sns->subscribe(                      "oo_result is
 returned for testing purposes."
                iv_topicarn = iv_topic_arn
                iv_protocol = 'email'
                iv_endpoint = iv_email_address
                iv_returnsubscriptionarn = abap_true ).
        MESSAGE 'Email address subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.
```

- For API details, see Subscribe in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.subscribe(
            input: SubscribeInput(
                endpoint: email,
                protocol: "email",
                returnSubscriptionArn: true,
                topicArn: arn
            )
        )

        guard let subscriptionArn = output.subscriptionArn else {
            print("No subscription ARN received from Amazon SNS.")
            return
        }

        print("Subscription \(subscriptionArn) created.")
```

Subscribe a phone number to a topic to receive notifications by SMS.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.subscribe(
            input: SubscribeInput(
                endpoint: phone,
                protocol: "sms",
                returnSubscriptionArn: true,
                topicArn: arn
            )
        )

        guard let subscriptionArn = output.subscriptionArn else {
            print("No subscription ARN received from Amazon SNS.")
            return
        }

        print("Subscription \(subscriptionArn) created.")
```

- For API details, see Subscribe in *AWS SDK for Swift API reference*.

Code examples for Amazon SNS using AWS SDKs

The following code examples show how to use Amazon SNS with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello Amazon SNS

The following code examples show how to get started using Amazon SNS.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your
 topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"\tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sns)

# Set this project's name.
```

```
project("hello_sns")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_sns.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_sns.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/sns/SNSClient.h>
#include <aws/sns/model/ListTopicsRequest.h>
```

```
#include <iostream>

/*
 *  A "Hello SNS" starter application which initializes an Amazon Simple
 Notification
 *  Service (Amazon SNS) client and lists the SNS topics in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sns'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::SNS::SNSClient snsClient(clientConfig);

        Aws::Vector<Aws::SNS::Model::Topic> allTopics;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SNS::Model::ListTopicsRequest request;

            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
            }

            const Aws::SNS::Model::ListTopicsOutcome outcome =
 snsClient.ListTopics(
                    request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SNS::Model::Topic> &paginatedTopics =
                        outcome.GetResult().GetTopics();
                if (!paginatedTopics.empty()) {
                    allTopics.insert(allTopics.cend(), paginatedTopics.cbegin(),
                                     paginatedTopics.cend());
                }
            }
            else {
                std::cerr << "Error listing topics " <<
 outcome.GetError().GetMessage()
                          << std::endl;
                return 1;
            }

            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());

        std::cout << "Hello Amazon SNS! You have " << allTopics.size() << "
 topic"
                  << (allTopics.size() == 1 ? "" : "s") << " in your account."
                  << std::endl;

        if (!allTopics.empty()) {
            std::cout << "Here are your topic ARNs." << std::endl;
            for (const Aws::SNS::Model::Topic &topic: allTopics) {
                std::cout << "  * " << topic.GetTopicArn() << std::endl;
            }
        }
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- For API details, see ListTopics in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
 Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- For API details, see ListTopics in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package com.example.sns;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.paginators.ListTopicsIterable;

public class HelloSNS {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listSNSTopics(snsClient);
        snsClient.close();
    }

    public static void listSNSTopics(SnsClient snsClient) {
        try {
            ListTopicsIterable listTopics = snsClient.listTopicsPaginator();
            listTopics.stream()
                    .flatMap(r -> r.topics().stream())
                    .forEach(content -> System.out.println(" Topic ARN: " +
 content.topicArn()));

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Initialize an SNS client and and list topics in your account.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your
 account.`,
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- For API details, see ListTopics in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.ListTopicsRequest
import aws.sdk.kotlin.services.sns.paginators.listTopicsPaginated
import kotlinx.coroutines.flow.transform

/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient
            .listTopicsPaginated(ListTopicsRequest { })
            .transform { it.topics?.forEach { topic -> emit(topic) } }
            .collect { topic ->
                println("The topic ARN is ${topic.topicArn}")
            }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Kotlin API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The Package.swift file.

```
import PackageDescription

let package = Package(
    name: "sns-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sns-basics",
            dependencies: [
                .product(name: "AWSSNS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

The main Swift program.

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sns-basics",
        abstract: """
        This example shows how to list all of your available Amazon SNS topics.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        var topics: [String] = []
        let outputPages = snsClient.listTopicsPaginated(
            input: ListTopicsInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let topicList = output.topics else {
                print("Unable to get a page of Amazon SNS topics.")
                return
            }

            // Iterate over the topics listed on this page, adding their ARNs
            // to the `topics` array.

            for topic in topicList {
                guard let arn = topic.topicArn else {
                    print("Topic has no ARN.")
                    return
                }
                topics.append(arn)
            }
        }

        print("You have \(topics.count) topics:")
        for topic in topics {
            print("   \(topic)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Swift API reference*.

Code examples

- Basic examples for Amazon SNS using AWS SDKs
  - Hello Amazon SNS
  - Actions for Amazon SNS using AWS SDKs
    - Use CheckIfPhoneNumberIsOptedOut with an AWS SDK or CLI
    - Use ConfirmSubscription with an AWS SDK or CLI
    - Use CreateTopic with an AWS SDK or CLI
    - Use DeleteTopic with an AWS SDK or CLI
    - Use GetSMSAttributes with an AWS SDK or CLI
    - Use GetTopicAttributes with an AWS SDK or CLI
    - Use ListPhoneNumbersOptedOut with an AWS SDK or CLI
    - Use ListSubscriptions with an AWS SDK or CLI
    - Use ListTopics with an AWS SDK or CLI
    - Use Publish with an AWS SDK or CLI
    - Use SetSMSAttributes with an AWS SDK or CLI
    - Use SetSubscriptionAttributes with an AWS SDK or CLI
    - Use SetSubscriptionAttributesRedrivePolicy with an AWS SDK
    - Use SetTopicAttributes with an AWS SDK or CLI
    - Use Subscribe with an AWS SDK or CLI
    - Use TagResource with an AWS SDK or CLI
    - Use Unsubscribe with an AWS SDK or CLI
- Scenarios for Amazon SNS using AWS SDKs
  - Build an application to submit data to a DynamoDB table
  - Build a publish and subscription application that translates messages

Basic examples for Amazon SNS using AWS SDKs

The following code examples show how to use the basics of Amazon Simple Notification Service with AWS SDKs.

Examples

- Use Publish with an AWS SDK or CLI
- Use SetSMSAttributes with an AWS SDK or CLI
- Use SetSubscriptionAttributes with an AWS SDK or CLI
- Use SetSubscriptionAttributesRedrivePolicy with an AWS SDK
- Use SetTopicAttributes with an AWS SDK or CLI
- Use Subscribe with an AWS SDK or CLI
- Use TagResource with an AWS SDK or CLI
- Use Unsubscribe with an AWS SDK or CLI

Hello Amazon SNS

The following code examples show how to get started using Amazon SNS.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your
 topics:");
        Console.WriteLine();
        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"\tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sns)

# Set this project's name.
project("hello_sns")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)
```

```
# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_sns.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_sns.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/sns/SNSClient.h>
#include <aws/sns/model/ListTopicsRequest.h>
#include <iostream>

/*
 *  A "Hello SNS" starter application which initializes an Amazon Simple
 Notification
 *  Service (Amazon SNS) client and lists the SNS topics in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sns'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::SNS::SNSClient snsClient(clientConfig);

        Aws::Vector<Aws::SNS::Model::Topic> allTopics;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SNS::Model::ListTopicsRequest request;

            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
            }

            const Aws::SNS::Model::ListTopicsOutcome outcome =
 snsClient.ListTopics(
                    request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SNS::Model::Topic> &paginatedTopics =
                        outcome.GetResult().GetTopics();
                if (!paginatedTopics.empty()) {
                    allTopics.insert(allTopics.cend(), paginatedTopics.cbegin(),
                                     paginatedTopics.cend());
                }
            }
            else {
                std::cerr << "Error listing topics " <<
 outcome.GetError().GetMessage()
```

```
                            << std::endl;
                  return 1;
              }

              nextToken = outcome.GetResult().GetNextToken();
          } while (!nextToken.empty());

          std::cout << "Hello Amazon SNS! You have " << allTopics.size() << "
 topic"
                    << (allTopics.size() == 1 ? "" : "s") << " in your account."
                    << std::endl;

          if (!allTopics.empty()) {
              std::cout << "Here are your topic ARNs." << std::endl;
              for (const Aws::SNS::Model::Topic &topic: allTopics) {
                  std::cout << "  * " << topic.GetTopicArn() << std::endl;
              }
          }
      }


      Aws::ShutdownAPI(options); // Should only be called once.
      return 0;
}
```

- For API details, see ListTopics in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
 Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- For API details, see ListTopics in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package com.example.sns;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.paginators.ListTopicsIterable;

public class HelloSNS {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listSNSTopics(snsClient);
        snsClient.close();
    }

    public static void listSNSTopics(SnsClient snsClient) {
        try {
            ListTopicsIterable listTopics = snsClient.listTopicsPaginator();
            listTopics.stream()
                    .flatMap(r -> r.topics().stream())
                    .forEach(content -> System.out.println(" Topic ARN: " +
 content.topicArn()));

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
                System.exit(1);
            }
        }
    }
```

- For API details, see ListTopics in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Initialize an SNS client and and list topics in your account.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";
  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your
 account.`,
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- For API details, see ListTopics in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.ListTopicsRequest
import aws.sdk.kotlin.services.sns.paginators.listTopicsPaginated
import kotlinx.coroutines.flow.transform

/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient
            .listTopicsPaginated(ListTopicsRequest { })
            .transform { it.topics?.forEach { topic -> emit(topic) } }
            .collect { topic ->
```

```
                println("The topic ARN is ${topic.topicArn}")
            }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Kotlin API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The Package.swift file.

```
import PackageDescription

let package = Package(
    name: "sns-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
```

```
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sns-basics",
            dependencies: [
                .product(name: "AWSSNS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

The main Swift program.

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sns-basics",
        abstract: """
        This example shows how to list all of your available Amazon SNS topics.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        var topics: [String] = []
        let outputPages = snsClient.listTopicsPaginated(
            input: ListTopicsInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let topicList = output.topics else {
                print("Unable to get a page of Amazon SNS topics.")
                return
            }

            // Iterate over the topics listed on this page, adding their ARNs
            // to the `topics` array.

            for topic in topicList {
                guard let arn = topic.topicArn else {
                    print("Topic has no ARN.")
                    return
                }
                topics.append(arn)
            }
        }

        print("You have \(topics.count) topics:")
        for topic in topics {
            print("   \(topic)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for Amazon SNS using AWS SDKs

The following code examples demonstrate how to perform individual Amazon SNS actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon SNS API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon SNS using AWS SDKs .

The following examples include only the most commonly used actions. For a complete list, see the Amazon Simple Notification Service API Reference.

Examples

- Use Subscribe with an AWS SDK or CLI
- Use TagResource with an AWS SDK or CLI
- Use Unsubscribe with an AWS SDK or CLI

Use CheckIfPhoneNumberIsOptedOut with an AWS SDK or CLI

The following code examples show how to use CheckIfPhoneNumberIsOptedOut.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// This example shows how to use the Amazon Simple Notification Service
    /// (Amazon SNS) to check whether a phone number has been opted out.
    /// </summary>
    public class IsPhoneNumOptedOut
    {
        public static async Task Main()
        {
            string phoneNumber = "+15551112222";

            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            await CheckIfOptedOutAsync(client, phoneNumber);
        }

        /// <summary>
        /// Checks to see if the supplied phone number has been opted out.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS Client object used
        /// to check if the phone number has been opted out.</param>
        /// <param name="phoneNumber">A string representing the phone number
        /// to check.</param>
        public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string
phoneNumber)
        {
            var request = new CheckIfPhoneNumberIsOptedOutRequest
            {
                PhoneNumber = phoneNumber,
            };

            try
            {
                var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    string optOutStatus = response.IsOptedOut ? "opted out" :
"not opted out.";
                    Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
                }
            }
            catch (AuthorizationErrorException ex)
            {
                Console.WriteLine($"{ex.Message}");
            }
        }
    }
```

- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To check SMS message opt-out for a phone number

The following `check-if-phone-number-is-opted-out` example checks whether the specified phone number is opted out of receiving SMS messages from the current AWS account.

```
aws sns check-if-phone-number-is-opted-out \
    --phone-number +1555550100
```

Output:

```
{
    "isOptedOut": false
}
```

- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import
 software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutRequest;
import
 software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CheckOptOut {
    public static void main(String[] args) {

        final String usage = """

                Usage:    <phoneNumber>

                Where:
                   phoneNumber - The mobile phone number to look up (for example,
 +1XXX5550100).

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String phoneNumber = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        checkPhone(snsClient, phoneNumber);
        snsClient.close();
    }

    public static void checkPhone(SnsClient snsClient, String phoneNumber) {
        try {
            CheckIfPhoneNumberIsOptedOutRequest request =
 CheckIfPhoneNumberIsOptedOutRequest.builder()
                    .phoneNumber(phoneNumber)
                    .build();

            CheckIfPhoneNumberIsOptedOutResponse result =
 snsClient.checkIfPhoneNumberIsOptedOut(request);
            System.out.println(
                    result.isOptedOut() + "Phone Number " + phoneNumber + " has
 Opted Out of receiving sns messages." +
                            "\n\nStatus was " +
 result.sdkHttpResponse().statusCode());
```

```
        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

```
/**
 * Indicates whether the phone number owner has opted out of receiving SMS
 messages from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$phone = '+1XXX5550100';

try {
    $result = $SnSclient->checkIfPhoneNumberIsOptedOut([
        'phoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ConfirmSubscription` with an AWS SDK or CLI

The following code examples show how to use `ConfirmSubscription`.

CLI

AWS CLI

To confirm a subscription

The following `confirm-subscription` command completes the confirmation process started when you subscribed to an SNS topic named `my-topic`. The --token parameter comes from the confirmation message sent to the notification endpoint specified in the subscribe call.

```
aws sns confirm-subscription \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
    --
token 2336412f37fb687f5d51e6e241d7700ae02f7124d8268910b858cb4db727ceeb2474bb937929d3bdd7c
```

Output:

```
{
    "SubscriptionArn": "arn:aws:sns:us-west-2:123456789012:my-
topic:8a21d249-4329-4871-acc6-7be709c6ea7f"
}
```

- For API details, see ConfirmSubscription in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ConfirmSubscriptionRequest;
import software.amazon.awssdk.services.sns.model.ConfirmSubscriptionResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ConfirmSubscription {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <subscriptionToken> <topicArn>

                Where:
                   subscriptionToken - A short-lived token sent to an endpoint
 during the Subscribe action.
                   topicArn - The ARN of the topic.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionToken = args[0];
        String topicArn = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        confirmSub(snsClient, subscriptionToken, topicArn);
        snsClient.close();
    }

    public static void confirmSub(SnsClient snsClient, String subscriptionToken,
 String topicArn) {
        try {
            ConfirmSubscriptionRequest request =
 ConfirmSubscriptionRequest.builder()
                    .token(subscriptionToken)
                    .topicArn(topicArn)
                    .build();

            ConfirmSubscriptionResponse result =
 snsClient.confirmSubscription(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode() + "\n\nSubscription Arn: \n\n"
                    + result.subscriptionArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ConfirmSubscription in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 a subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while
 unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ConfirmSubscription in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Verifies an endpoint owner's intent to receive messages by
 * validating the token sent to the endpoint by an earlier Subscribe action.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription_token = 'arn:aws:sns:us-east-1:111122223333:MyTopic:123456-
abcd-12ab-1234-12ba3dc1234a';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->confirmSubscription([
        'Token' => $subscription_token,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see ConfirmSubscription in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateTopic` with an AWS SDK or CLI

The following code examples show how to use `CreateTopic`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and publish to a FIFO topic
- Publish messages to queues

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a topic with a specific name.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// This example shows how to use Amazon Simple Notification Service
    /// (Amazon SNS) to add a new Amazon SNS topic.
    /// </summary>
    public class CreateSNSTopic
    {
        public static async Task Main()
        {
            string topicName = "ExampleSNSTopic";

            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            var topicArn = await CreateSNSTopicAsync(client, topicName);
            Console.WriteLine($"New topic ARN: {topicArn}");
        }

        /// <summary>
        /// Creates a new SNS topic using the supplied topic name.
        /// </summary>
        /// <param name="client">The initialized SNS client object used to
        /// create the new topic.</param>
        /// <param name="topicName">A string representing the topic name.</param>
        /// <returns>The Amazon Resource Name (ARN) of the created topic.</
returns>
        public static async Task<string>
 CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
        {
            var request = new CreateTopicRequest
            {
                Name = topicName,
            };

            var response = await client.CreateTopicAsync(request);

            return response.TopicArn;
        }
    }
```

Create a new topic with a name and specific FIFO and de-duplication attributes.

```
    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
 "true");
            }
        }

        var createResponse = await
 _amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }
```

- For API details, see CreateTopic in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Create an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param topicName: An Amazon SNS topic name.
  \param topicARNResult: String to return the Amazon Resource Name (ARN) for the
 topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::createTopic(const Aws::String &topicName,
                              Aws::String &topicARNResult,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::CreateTopicRequest request;
    request.SetName(topicName);

    const Aws::SNS::Model::CreateTopicOutcome outcome =
 snsClient.CreateTopic(request);

    if (outcome.IsSuccess()) {
        topicARNResult = outcome.GetResult().GetTopicArn();
        std::cout << "Successfully created an Amazon SNS topic " << topicName
                  << " with topic ARN '" << topicARNResult
                  << "'." << std::endl;

    }
    else {
        std::cerr << "Error creating topic " << topicName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
        topicARNResult.clear();
    }
    return outcome.IsSuccess();
}
```

- For API details, see CreateTopic in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create an SNS topic

The following `create-topic` example creates an SNS topic named `my-topic`.

```
aws sns create-topic \
    --name my-topic
```

Output:

```
{
    "ResponseMetadata": {
        "RequestId": "1469e8d7-1642-564e-b85d-a19b4b341f83"
    },
    "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
}
```

For more information, see Using the AWS Command Line Interface with Amazon SQS and Amazon SNS in the *AWS Command Line Interface User Guide*.

- For API details, see CreateTopic in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}
```

- For API details, see CreateTopic in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicName>

                Where:
                    topicName - The name of the topic to create (for example,
 mytopic).

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicName = args[0];
        System.out.println("Creating a topic with name: " + topicName);
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnVal = createSNSTopic(snsClient, topicName);
        System.out.println("The topic ARN is" + arnVal);
        snsClient.close();
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                    .name(topicName)
                    .build();

            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreateTopic in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateTopic in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createSNSTopic(topicName: String): String {
    val request =
        CreateTopicRequest {
            name = topicName
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}
```

- For API details, see CreateTopic in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Create a Simple Notification Service topics in your AWS account at the
 requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topicname = 'myTopic';

try {
    $result = $SnSclient->createTopic([
        'Name' => $topicname,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see CreateTopic in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    def create_topic(self, name):
        """
        Creates a notification topic.

        :param name: The name of the topic to create.
        :return: The newly created topic.
        """
        try:
            topic = self.sns_resource.create_topic(Name=name)
            logger.info("Created topic %s with ARN %s.", name, topic.arn)
        except ClientError:
            logger.exception("Couldn't create topic %s.", name)
            raise
        else:
            return topic
```

- For API details, see CreateTopic in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# This class demonstrates how to create an Amazon Simple Notification Service
 (SNS) topic.
class SNSTopicCreator
  # Initializes an SNS client.
  #
  # Utilizes the default AWS configuration for region and credentials.
  def initialize
    @sns_client = Aws::SNS::Client.new
  end

  # Attempts to create an SNS topic with the specified name.
  #
  # @param topic_name [String] The name of the SNS topic to create.
  # @return [Boolean] true if the topic was successfully created, false
 otherwise.
  def create_topic(topic_name)
    @sns_client.create_topic(name: topic_name)
    puts "The topic '#{topic_name}' was successfully created."
    true
  rescue Aws::SNS::Errors::ServiceError => e
    # Handles SNS service errors gracefully.
    puts "Error while creating the topic named '#{topic_name}': #{e.message}"
    false
  end
end

# Example usage:
```

```
if $PROGRAM_NAME == __FILE__
  topic_name = 'YourTopicName' # Replace with your topic name
  sns_topic_creator = SNSTopicCreator.new

  puts "Creating the topic '#{topic_name}'..."
  unless sns_topic_creator.create_topic(topic_name)
    puts 'The topic was not created. Stopping program.'
    exit 1
  end
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see CreateTopic in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn().unwrap_or_default()
    );

    Ok(())
}
```

- For API details, see CreateTopic in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   TRY.
       oo_result = lo_sns->createtopic( iv_name = iv_topic_name ). " oo_result
 is returned for testing purposes. "
       MESSAGE 'SNS topic created' TYPE 'I'.
     CATCH /aws1/cx_snstopiclimitexcdex.
       MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
   ENDTRY.
```

- For API details, see CreateTopic in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.createTopic(
            input: CreateTopicInput(name: name)
        )
        guard let arn = output.topicArn else {
            print("No topic ARN returned by Amazon SNS.")
            return
        }
```

- For API details, see CreateTopic in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteTopic` with an AWS SDK or CLI

The following code examples show how to use `DeleteTopic`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Publish messages to queues

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a topic by its topic ARN.

```
    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteTopic in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::DeleteTopicOutcome outcome =
 snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting topic " << topicARN << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteTopic in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an SNS topic

The following `delete-topic` example deletes the specified SNS topic.

```
aws sns delete-topic \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

This command produces no output.

- For API details, see DeleteTopic in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

- For API details, see DeleteTopic in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:     <topicArn>

                Where:
                   topicArn - The ARN of the topic to delete.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        System.out.println("Deleting a topic with name: " + topicArn);
        deleteSNSTopic(snsClient, topicArn);
        snsClient.close();
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                    .topicArn(topicArn)
                    .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteTopic in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
```

```
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteTopic in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteSNSTopic(topicArnVal: String) {
    val request =
        DeleteTopicRequest {
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was successfully deleted.")
    }
}
```

- For API details, see DeleteTopic in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see DeleteTopic in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

- For API details, see DeleteTopic in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).
        MESSAGE 'SNS topic deleted.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteTopic in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)
        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: arn)
        )
```

- For API details, see DeleteTopic in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetSMSAttributes` with an AWS SDK or CLI

The following code examples show how to use `GetSMSAttributes`.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Retrieve the default settings for sending SMS messages from your AWS account
 by using
//! Amazon Simple Notification Service (Amazon SNS).
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SNS::getSMSType(const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::GetSMSAttributesRequest request;
    //Set the request to only retrieve the DefaultSMSType setting.
    //Without the following line, GetSMSAttributes would retrieve all settings.
    request.AddAttributes("DefaultSMSType");
    const Aws::SNS::Model::GetSMSAttributesOutcome outcome =
 snsClient.GetSMSAttributes(
            request);

    if (outcome.IsSuccess()) {
        const Aws::Map<Aws::String, Aws::String> attributes =
                outcome.GetResult().GetAttributes();
        if (!attributes.empty()) {
            for (auto const &att: attributes) {
                std::cout << att.first << ":  " << att.second << std::endl;
            }
        }
        else {
            std::cout
                    << "AwsDoc::SNS::getSMSType - an empty map of attributes was
 retrieved."
                    << std::endl;
        }
    }
    else {
        std::cerr << "Error while getting SMS Type: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see GetSMSAttributes in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list the default SMS message attributes

The following `get-sms-attributes` example lists the default attributes for sending SMS messages.

```
aws sns get-sms-attributes
```

Output:

```
{
    "attributes": {
        "DefaultSenderID": "MyName"
    }
}
```

- For API details, see GetSMSAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import
 software.amazon.awssdk.services.sns.model.GetSubscriptionAttributesRequest;
import
 software.amazon.awssdk.services.sns.model.GetSubscriptionAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.Iterator;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetSMSAtrributes {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn>

                Where:
                   topicArn - The ARN of the topic from which to retrieve 
 attributes.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        getSNSAttrutes(snsClient, topicArn);
        snsClient.close();
    }

    public static void getSNSAttrutes(SnsClient snsClient, String topicArn) {
        try {
            GetSubscriptionAttributesRequest request = 
 GetSubscriptionAttributesRequest.builder()
                    .subscriptionArn(topicArn)
                    .build();

            // Get the Subscription attributes
            GetSubscriptionAttributesResponse res = 
 snsClient.getSubscriptionAttributes(request);
            Map<String, String> map = res.attributes();

            // Iterate through the map
            Iterator iter = map.entrySet().iterator();
            while (iter.hasNext()) {
                Map.Entry entry = (Map.Entry) iter.next();
                System.out.println("[Key] : " + entry.getKey() + " [Value] : " + 
 entry.getValue());
            }

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        System.out.println("\n\nStatus was good");
    }
}
```

- For API details, see GetSMSAttributes in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetSMSAttributes in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Get the type of SMS Message sent by default from the AWS SNS service.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->getSMSAttributes([
        'attributes' => ['DefaultSMSType'],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see GetSMSAttributes in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetTopicAttributes` with an AWS SDK or CLI

The following code examples show how to use `GetTopicAttributes`.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;

    /// <summary>
    /// This example shows how to retrieve the attributes of an Amazon Simple
    /// Notification Service (Amazon SNS) topic.
    /// </summary>
    public class GetTopicAttributes
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-
west-2:000000000000:ExampleSNSTopic";
            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            var attributes = await GetTopicAttributesAsync(client, topicArn);
            DisplayTopicAttributes(attributes);
        }

        /// <summary>
        /// Given the ARN of the Amazon SNS topic, this method retrieves the
 topic
        /// attributes.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the attributes for the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic for which to retrieve
        /// the attributes.</param>
        /// <returns>A Dictionary of topic attributes.</returns>
        public static async Task<Dictionary<string, string>>
 GetTopicAttributesAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            var response = await client.GetTopicAttributesAsync(topicArn);

            return response.Attributes;
        }

        /// <summary>
        /// This method displays the attributes for an Amazon SNS topic.
        /// </summary>
        /// <param name="topicAttributes">A Dictionary containing the
        /// attributes for an Amazon SNS topic.</param>
        public static void DisplayTopicAttributes(Dictionary<string, string>
 topicAttributes)
        {
            foreach (KeyValuePair<string, string> entry in topicAttributes)
            {
                Console.WriteLine($"{entry.Key}: {entry.Value}\n");
            }
        }
    }
```

- For API details, see GetTopicAttributes in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Retrieve the properties of an Amazon Simple Notification Service (Amazon SNS)
 topic.
/*!
```

```
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::getTopicAttributes(const Aws::String &topicARN,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);
    Aws::SNS::Model::GetTopicAttributesRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::GetTopicAttributesOutcome outcome =
 snsClient.GetTopicAttributes(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Topic Attributes:" << std::endl;
        for (auto const &attribute: outcome.GetResult().GetAttributes()) {
            std::cout << "  * " << attribute.first << " : " << attribute.second
                      << std::endl;
        }
    }
    else {
        std::cerr << "Error while getting Topic attributes "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see GetTopicAttributes in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To retrieve the attributes of a topic

The following `get-topic-attributes` example displays the attributes for the specified topic.

```
aws sns get-topic-attributes \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

Output:

```
{
    "Attributes": {
        "SubscriptionsConfirmed": "1",
        "DisplayName": "my-topic",
        "SubscriptionsDeleted": "0",
        "EffectiveDeliveryPolicy": "{\"http\":{\"defaultHealthyRetryPolicy
\":{\"minDelayTarget\":20,\"maxDelayTarget\":20,\"numRetries\":3,
\"numMaxDelayRetries\":0,\"numNoDelayRetries\":0,\"numMinDelayRetries\":0,
\"backoffFunction\":\"linear\"},\"disableSubscriptionOverrides\":false}}",
        "Owner": "123456789012",
        "Policy": "{\"Version\":\"2008-10-17\",\"Id\":\"__default_policy_ID
\",\"Statement\":[{\"Sid\":\"__default_statement_ID\",\"Effect\":
\"Allow\",\"Principal\":{\"AWS\":\"*\"},\"Action\":[\"SNS:Subscribe\",
\"SNS:ListSubscriptionsByTopic\",\"SNS:DeleteTopic\",\"SNS:GetTopicAttributes
\",\"SNS:Publish\",\"SNS:RemovePermission\",\"SNS:AddPermission\",
\"SNS:SetTopicAttributes\"],\"Resource\":\"arn:aws:sns:us-west-2:123456789012:my-
topic\",\"Condition\":{\"StringEquals\":{\"AWS:SourceOwner\":
\"0123456789012\"}}}]}",
        "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic",
        "SubscriptionsPending": "0"
    }
}
```

- For API details, see GetTopicAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.GetTopicAttributesRequest;
import software.amazon.awssdk.services.sns.model.GetTopicAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetTopicAttributes {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn>

                Where:
                   topicArn - The ARN of the topic to look up.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        System.out.println("Getting attributes for a topic with name: " +
 topicArn);
        getSNSTopicAttributes(snsClient, topicArn);
        snsClient.close();
    }

    public static void getSNSTopicAttributes(SnsClient snsClient, String
 topicArn) {
        try {
            GetTopicAttributesRequest request =
 GetTopicAttributesRequest.builder()
                    .topicArn(topicArn)
                    .build();

            GetTopicAttributesResponse result =
 snsClient.getTopicAttributes(request);
            System.out.println("\n\nStatus is " +
 result.sdkHttpResponse().statusCode() + "\n\nAttributes: \n\n"
                    + result.attributes());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see GetTopicAttributes in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDela
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/).

- For API details, see GetTopicAttributes in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetTopicAttributes in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getSNSTopicAttributes(topicArnVal: String) {
    val request =
        GetTopicAttributesRequest {
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.getTopicAttributes(request)
        println("${result.attributes}")
    }
}
```

- For API details, see GetTopicAttributes in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
```

```
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->getTopicAttributes([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see GetTopicAttributes in *AWS SDK for PHP API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_sns->gettopicattributes( iv_topicarn = iv_topic_arn ). "
 oo_result is returned for testing purposes. "
        DATA(lt_attributes) = oo_result->get_attributes( ).
        MESSAGE 'Retrieved attributes/properties of a topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see GetTopicAttributes in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListPhoneNumbersOptedOut` with an AWS SDK or CLI

The following code examples show how to use `ListPhoneNumbersOptedOut`.

CLI

AWS CLI

To list SMS message opt-outs

The following `list-phone-numbers-opted-out` example lists the phone numbers opted out of receiving SMS messages.

```
aws sns list-phone-numbers-opted-out
```

Output:

```
{
    "phoneNumbers": [
        "+15555550100"
    ]
}
```

- For API details, see ListPhoneNumbersOptedOut in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutRequest;
import
 software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListOptOut {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listOpts(snsClient);
        snsClient.close();
    }

    public static void listOpts(SnsClient snsClient) {
        try {
            ListPhoneNumbersOptedOutRequest request =
 ListPhoneNumbersOptedOutRequest.builder().build();
            ListPhoneNumbersOptedOutResponse result =
 snsClient.listPhoneNumbersOptedOut(request);
            System.out.println("Status is " +
 result.sdkHttpResponse().statusCode() + "\n\nPhone Numbers: \n\n"
                    + result.phoneNumbers());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListPhoneNumbersOptedOut in *AWS SDK for Java 2.x API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;



/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages
 from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->listPhoneNumbersOptedOut();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.

- For API details, see ListPhoneNumbersOptedOut in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListSubscriptions` with an AWS SDK or CLI

The following code examples show how to use `ListSubscriptions`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   using Amazon.SimpleNotificationService;
   using Amazon.SimpleNotificationService.Model;

   /// <summary>
   /// This example will retrieve a list of the existing Amazon Simple
   /// Notification Service (Amazon SNS) subscriptions.
   /// </summary>
   public class ListSubscriptions
   {
       public static async Task Main()
       {
           IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

           Console.WriteLine("Enter a topic ARN to list subscriptions for a
specific topic, " +
                             "or press Enter to list subscriptions for all
topics.");
```

```
            var topicArn = Console.ReadLine();
            Console.WriteLine();

            var subscriptions = await GetSubscriptionsListAsync(client,
 topicArn);

            DisplaySubscriptionList(subscriptions);
        }

        /// <summary>
        /// Gets a list of the existing Amazon SNS subscriptions, optionally by
 specifying a topic ARN.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to obtain the list of subscriptions.</param>
        /// <param name="topicArn">The optional ARN of a specific topic. Defaults
 to null.</param>
        /// <returns>A list containing information about each subscription.</
returns>
        public static async Task<List<Subscription>>
 GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string
 topicArn = null)
        {
            var results = new List<Subscription>();

            if (!string.IsNullOrEmpty(topicArn))
            {
                var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
                    new ListSubscriptionsByTopicRequest()
                    {
                        TopicArn = topicArn,
                    });

                // Get the entire list using the paginator.
                await foreach (var subscription in paginateByTopic.Subscriptions)
                {
                    results.Add(subscription);
                }
            }
            else
            {
                var paginateAllSubscriptions =
 client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());
```

```
               // Get the entire list using the paginator.
               await foreach (var subscription in
paginateAllSubscriptions.Subscriptions)
               {
                   results.Add(subscription);
               }
           }

           return results;
       }

       /// <summary>
       /// Display a list of Amazon SNS subscription information.
       /// </summary>
       /// <param name="subscriptionList">A list containing details for existing
       /// Amazon SNS subscriptions.</param>
       public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
       {
           foreach (var subscription in subscriptionList)
           {
               Console.WriteLine($"Owner: {subscription.Owner}");
               Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
               Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
               Console.WriteLine($"Endpoint: {subscription.Endpoint}");
               Console.WriteLine($"Protocol: {subscription.Protocol}");
               Console.WriteLine();
           }
       }
   }
```

- For API details, see ListSubscriptions in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Retrieve a list of Amazon Simple Notification Service (Amazon SNS)
 subscriptions.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::listSubscriptions(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::String nextToken; // Next token is used to handle a paginated response.
    bool result = true;
    Aws::Vector<Aws::SNS::Model::Subscription> subscriptions;
    do {
        Aws::SNS::Model::ListSubscriptionsRequest request;

        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        const Aws::SNS::Model::ListSubscriptionsOutcome outcome =
 snsClient.ListSubscriptions(
                request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::SNS::Model::Subscription> &newSubscriptions =
                    outcome.GetResult().GetSubscriptions();
            subscriptions.insert(subscriptions.cend(), newSubscriptions.begin(),
                                 newSubscriptions.end());
        }
        else {
            std::cerr << "Error listing subscriptions "
```

```
                      << outcome.GetError().GetMessage()
                      <<
                      std::endl;
            result = false;
            break;
        }

        nextToken = outcome.GetResult().GetNextToken();
    } while (!nextToken.empty());

    if (result) {
        if (subscriptions.empty()) {
            std::cout << "No subscriptions found" << std::endl;
        }
        else {
            std::cout << "Subscriptions list:" << std::endl;
            for (auto const &subscription: subscriptions) {
                std::cout << "  * " << subscription.GetSubscriptionArn() <<
 std::endl;
            }
        }
    }
    return result;
}
```

- For API details, see ListSubscriptions in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list your SNS subscriptions

The following `list-subscriptions` example displays a list of the SNS subscriptions in your AWS account.

```
aws sns list-subscriptions
```

Output:

```
{
    "Subscriptions": [
        {
            "Owner": "123456789012",
            "Endpoint": "my-email@example.com",
            "Protocol": "email",
            "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic",
            "SubscriptionArn": "arn:aws:sns:us-west-2:123456789012:my-
topic:8a21d249-4329-4871-acc6-7be709c6ea7f"
        }
    ]
}
```

- For API details, see ListSubscriptions in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListSubscriptionsRequest;
import software.amazon.awssdk.services.sns.model.ListSubscriptionsResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListSubscriptions {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
```

```
                .region(Region.US_EAST_1)
                .build();

        listSNSSubscriptions(snsClient);
        snsClient.close();
    }

    public static void listSNSSubscriptions(SnsClient snsClient) {
        try {
            ListSubscriptionsRequest request = ListSubscriptionsRequest.builder()
                    .build();

            ListSubscriptionsResponse result =
 snsClient.listSubscriptions(request);
            System.out.println(result.subscriptions());

        } catch (SnsException e) {

            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListSubscriptions in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListSubscriptions in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listSNSSubscriptions() {
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.listSubscriptions(ListSubscriptionsRequest {})
        response.subscriptions?.forEach { sub ->
            println("Sub ARN is ${sub.subscriptionArn}")
            println("Sub protocol is ${sub.protocol}")
        }
    }
}
```

- For API details, see ListSubscriptions in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';
```

```
use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Returns a list of Amazon SNS subscriptions in the requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->listSubscriptions();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see ListSubscriptions in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    def list_subscriptions(self, topic=None):
        """
        Lists subscriptions for the current account, optionally limited to a
        specific topic.

        :param topic: When specified, only subscriptions to this topic are
 returned.
        :return: An iterator that yields the subscriptions.
        """
        try:
            if topic is None:
                subs_iter = self.sns_resource.subscriptions.all()
            else:
                subs_iter = topic.subscriptions.all()
            logger.info("Got subscriptions.")
        except ClientError:
            logger.exception("Couldn't get subscriptions.")
            raise
        else:
            return subs_iter
```

- For API details, see ListSubscriptions in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# This class demonstrates how to list subscriptions to an Amazon Simple
 Notification Service (SNS) topic
class SnsSubscriptionLister
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Lists subscriptions for a given SNS topic
  # @param topic_arn [String] The ARN of the SNS topic
  # @return [Types::ListSubscriptionsResponse] subscriptions: The response object
  def list_subscriptions(topic_arn)
    @logger.info("Listing subscriptions for topic: #{topic_arn}")
    subscriptions = @sns_client.list_subscriptions_by_topic(topic_arn: topic_arn)
    subscriptions.subscriptions.each do |subscription|
      @logger.info("Subscription endpoint: #{subscription.endpoint}")
    end
    subscriptions
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error listing subscriptions: #{e.message}")
    raise
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  sns_client = Aws::SNS::Client.new
  topic_arn = 'SNS_TOPIC_ARN' # Replace with your SNS topic ARN
  lister = SnsSubscriptionLister.new(sns_client)

  begin
    lister.list_subscriptions(topic_arn)
  rescue StandardError => e
    puts "Failed to list subscriptions: #{e.message}"
    exit 1
  end
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see ListSubscriptions in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   TRY.
       oo_result = lo_sns->listsubscriptions( ).                " oo_result is
returned for testing purposes. "
       DATA(lt_subscriptions) = oo_result->get_subscriptions( ).
       MESSAGE 'Retrieved list of subscribers.' TYPE 'I'.
     CATCH /aws1/cx_rt_generic.
       MESSAGE 'Unable to list subscribers.' TYPE 'E'.
   ENDTRY.
```

- For API details, see ListSubscriptions in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListTopics` with an AWS SDK or CLI

The following code examples show how to use `ListTopics`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// Lists the Amazon Simple Notification Service (Amazon SNS)
    /// topics for the current account.
    /// </summary>
    public class ListSNSTopics
    {
        public static async Task Main()
        {
            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            await GetTopicListAsync(client);
        }

        /// <summary>
        /// Retrieves the list of Amazon SNS topics in groups of up to 100
        /// topics.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the list of topics.</param>
        public static async Task
 GetTopicListAsync(IAmazonSimpleNotificationService client)
        {
            // If there are more than 100 Amazon SNS topics, the call to
            // ListTopicsAsync will return a value to pass to the
            // method to retrieve the next 100 (or less) topics.
            string nextToken = string.Empty;

            do
            {
                var response = await client.ListTopicsAsync(nextToken);
                DisplayTopicsList(response.Topics);
                nextToken = response.NextToken;
            }
            while (!string.IsNullOrEmpty(nextToken));
        }
        /// <summary>
        /// Displays the list of Amazon SNS Topic ARNs.
        /// </summary>
        /// <param name="topicList">The list of Topic ARNs.</param>
        public static void DisplayTopicsList(List<Topic> topicList)
        {
            foreach (var topic in topicList)
            {
                Console.WriteLine($"{topic.TopicArn}");
            }
        }
    }
```

- For API details, see ListTopics in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Retrieve a list of Amazon Simple Notification Service (Amazon SNS) topics.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SNS::listTopics(const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::String nextToken; // Next token is used to handle a paginated response.
    bool result = true;
    do {
        Aws::SNS::Model::ListTopicsRequest request;
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        const Aws::SNS::Model::ListTopicsOutcome outcome = snsClient.ListTopics(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "Topics list:" << std::endl;
            for (auto const &topic: outcome.GetResult().GetTopics()) {
                std::cout << "  * " << topic.GetTopicArn() << std::endl;
            }
        }
        else {
            std::cerr << "Error listing topics " <<
 outcome.GetError().GetMessage() <<
                      std::endl;
            result = false;
            break;
        }

        nextToken = outcome.GetResult().GetNextToken();
    } while (!nextToken.empty());

    return result;
}
```

- For API details, see ListTopics in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list your SNS topics

The following `list-topics` example lists all of SNS topics in your AWS account.

```
aws sns list-topics
```

Output:

```
{
    "Topics": [
        {
            "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
        }
    ]
}
```

- For API details, see ListTopics in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
 Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- For API details, see ListTopics in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListTopicsRequest;
import software.amazon.awssdk.services.sns.model.ListTopicsResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTopics {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listSNSTopics(snsClient);
        snsClient.close();
    }

    public static void listSNSTopics(SnsClient snsClient) {
        try {
            ListTopicsRequest request = ListTopicsRequest.builder()
                    .build();

            ListTopicsResponse result = snsClient.listTopics(request);
            System.out.println(
                    "Status was " + result.sdkHttpResponse().statusCode() + "\n
\nTopics\n\n" + result.topics());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListTopics in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listSNSTopics() {
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.listTopics(ListTopicsRequest { })
        response.topics?.forEach { topic ->
            println("The topic ARN is ${topic.topicArn}")
        }
    }
}
```

- For API details, see ListTopics in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

```
/**
 * Returns a list of the requester's topics from your AWS SNS account in the
 region specified.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->listTopics();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see ListTopics in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    def list_topics(self):
        """
        Lists topics for the current account.

        :return: An iterator that yields the topics.
        """
        try:
            topics_iter = self.sns_resource.topics.all()
            logger.info("Got topics.")
        except ClientError:
            logger.exception("Couldn't get topics.")
            raise
        else:
            return topics_iter
```

- For API details, see ListTopics in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def list_topics?(sns_client)
  sns_client.topics.each do |topic|
    puts topic.arn
  rescue StandardError => e
    puts "Error while listing the topics: #{e.message}"
  end
end

def run_me
  region = 'REGION'
  sns_client = Aws::SNS::Resource.new(region: region)

  puts 'Listing the topics.'

  return if list_topics?(sns_client)

  puts 'The bucket was not created. Stopping program.'
  exit 1
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see ListTopics in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_topics(client: &Client) -> Result<(), Error> {
    let resp = client.list_topics().send().await?;

    println!("Topic ARNs:");

    for topic in resp.topics() {
        println!("{}", topic.topic_arn().unwrap_or_default());
    }

    Ok(())
}
```

- For API details, see ListTopics in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_sns->listtopics( ).            " oo_result is returned for
 testing purposes. "
        DATA(lt_topics) = oo_result->get_topics( ).
        MESSAGE 'Retrieved list of topics.' TYPE 'I'.
      CATCH /aws1/cx_rt_generic.
        MESSAGE 'Unable to list topics.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListTopics in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        var topics: [String] = []
        let outputPages = snsClient.listTopicsPaginated(
            input: ListTopicsInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let topicList = output.topics else {
                print("Unable to get a page of Amazon SNS topics.")
                return
            }

            // Iterate over the topics listed on this page, adding their ARNs
            // to the `topics` array.

            for topic in topicList {
                guard let arn = topic.topicArn else {
                    print("Topic has no ARN.")
                    return
                }
                topics.append(arn)
            }
        }
```

- For API details, see ListTopics in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use Publish with an AWS SDK or CLI

The following code examples show how to use `Publish`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and publish to a FIFO topic
- Publish an SMS text message
- Publish messages to queues

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Publish a message to a topic.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// This example publishes a message to an Amazon Simple Notification
    /// Service (Amazon SNS) topic.
    /// </summary>
    public class PublishToSNSTopic
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-
east-2:000000000000:ExampleSNSTopic";
            string messageText = "This is an example message to publish to the
 ExampleSNSTopic.";

            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            await PublishToTopicAsync(client, topicArn, messageText);
        }

        /// <summary>
        /// Publishes a message to an Amazon SNS topic.
        /// </summary>
        /// <param name="client">The initialized client object used to publish
        /// to the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic.</param>
        /// <param name="messageText">The text of the message.</param>
        public static async Task PublishToTopicAsync(
            IAmazonSimpleNotificationService client,
            string topicArn,
            string messageText)
        {
            var request = new PublishRequest
            {
                TopicArn = topicArn,
                Message = messageText,
            };

            var response = await client.PublishAsync(request);

            Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
        }
    }
```

Publish a message to a topic with group, duplication, and attribute options.

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
```

```
       {
           Console.WriteLine();
           var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

           if (_useFifoTopic)
           {
               Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                 "\r\nAll messages within the same group will be
received in the order " +
                                 "they were published.");

               Console.WriteLine();
               var messageGroupId = GetUserResponse("Enter a message group ID
for this message:", "1");

               if (!_useContentBasedDeduplication)
               {
                   Console.WriteLine("Because you are not using content-based
deduplication, " +
                                     "you must enter a deduplication ID.");

                   Console.WriteLine("Enter a deduplication ID for this
message.");
                   deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
               }

               if (GetYesNoResponse("Add an attribute to this message?"))
               {
                   Console.WriteLine("Enter a number for an attribute.");
                   for (int i = 0; i < _tones.Length; i++)
                   {
                       Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                   }

                   var selection = GetUserResponse("", "1");
                   int.TryParse(selection, out var selectionNumber);

                   if (selectionNumber > 0 && selectionNumber < _tones.Length)
                   {
                       toneAttribute = _tones[selectionNumber - 1];
                   }
```

```
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
 messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?",
 false);
        }
    }
```

Apply the user's selections to the publish action.

```
    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
 message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
 attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await
 _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
```

- For API details, see Publish in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Send a message to an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
  \param message: The message to publish.
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::publishToTopic(const Aws::String &message,
                                 const Aws::String &topicARN,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with id '"
                  << outcome.GetResult().GetMessageId() << "'." << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

Publish a message with an attribute.

```
        static const Aws::String TONE_ATTRIBUTE("tone");
        static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
 "serious",
                                                       "sincere"};

        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SNS::SNSClient snsClient(clientConfiguration);

        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);

        if (filteringMessages && askYesNoQuestion(
                "Add an attribute to this message? (y/n) ")) {
            for (size_t i = 0; i < TONES.size(); ++i) {
                std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
            }
            int selection = askQuestionForIntRange(
                    "Enter a number for an attribute. ",
                    1, static_cast<int>(TONES.size()));
            Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
            messageAttributeValue.SetDataType("String");
            messageAttributeValue.SetStringValue(TONES[selection - 1]);
            request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
        }

        Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

        if (outcome.IsSuccess()) {
            std::cout << "Your message was successfully published." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Publish. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }
```

- For API details, see Publish in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To publish a message to a topic

The following `publish` example publishes the specified message to the specified SNS topic. The message comes from a text file, which enables you to include line breaks.

```
aws sns publish \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic" \
    --message file://message.txt
```

Contents of `message.txt`:

```
Hello World
Second Line
```

Output:

```
{
    "MessageId": "123a45b6-7890-12c3-45d6-111122223333"
}
```

Example 2: To publish an SMS message to a phone number

The following `publish` example publishes the message `Hello world!` to the phone number `+1-555-555-0100`.

```
aws sns publish \
    --message "Hello world!" \
    --phone-number +1-555-555-0100
```

Output:

```
{
    "MessageId": "123a45b6-7890-12c3-45d6-333322221111"
}
```

- For API details, see Publish in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
```

```
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

- For API details, see Publish in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <message> <topicArn>

                Where:
                   message - The message text to send.
                   topicArn - The ARN of the topic to publish.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
        String topicArn = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTopic(snsClient, message, topicArn);
        snsClient.close();
    }

    public static void pubTopic(SnsClient snsClient, String message, String
 topicArn) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .topicArn(topicArn)
                    .build();

            PublishResponse result = snsClient.publish(request);
            System.out
```

```
                    .println(result.messageId() + " Message sent. Status is " +
 result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see Publish in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Publish a message to a topic with group, duplication, and attribute options.

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );
```

```
    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [Publish](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun pubTopic(
    topicArnVal: String,
    messageVal: String,
) {
    val request =
        PublishRequest {
            message = messageVal
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

- For API details, see Publish in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows publishing a message with a single MessageAttribute declared inline.

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
 @{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='Strin
 StringValue ='AnyCity'}}
```

Example 2: This example shows publishing a message with multiple MessageAttributes declared in advance.

```
$cityAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- For API details, see Publish in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows publishing a message with a single MessageAttribute declared inline.

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
 @{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='Strin
 StringValue ='AnyCity'}}
```

Example 2: This example shows publishing a message with multiple MessageAttributes declared in advance.

```
$cityAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
 Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- For API details, see Publish in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Publish a message with attributes so that a subscription can filter based on attributes.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def publish_message(topic, message, attributes):
        """
        Publishes a message, with attributes, to a topic. Subscriptions can be
 filtered
        based on message attributes so that a subscription receives messages only
        when specified attributes are present.

        :param topic: The topic to publish to.
        :param message: The message to publish.
        :param attributes: The key-value attributes to attach to the message.
 Values
                           must be either `str` or `bytes`.
        :return: The ID of the message.
        """
        try:
            att_dict = {}
            for key, value in attributes.items():
                if isinstance(value, str):
                    att_dict[key] = {"DataType": "String", "StringValue": value}
                elif isinstance(value, bytes):
                    att_dict[key] = {"DataType": "Binary", "BinaryValue": value}
            response = topic.publish(Message=message, MessageAttributes=att_dict)
            message_id = response["MessageId"]
            logger.info(
                "Published message with attributes %s to topic %s.",
                attributes,
                topic.arn,
            )
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id
```

Publish a message that takes different forms based on the protocol of the subscriber.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource



    @staticmethod
    def publish_multi_message(
        topic, subject, default_message, sms_message, email_message
    ):
        """
        Publishes a multi-format message to a topic. A multi-format message takes
        different forms based on the protocol of the subscriber. For example,
        an SMS subscriber might receive a short version of the message
        while an email subscriber could receive a longer version.

        :param topic: The topic to publish to.
        :param subject: The subject of the message.
        :param default_message: The default version of the message. This version
 is
```

```
                                      sent to subscribers that have protocols that are 
not
                                      otherwise specified in the structured message.
        :param sms_message: The version of the message sent to SMS subscribers.
        :param email_message: The version of the message sent to email 
subscribers.
        :return: The ID of the message.
        """
        try:
            message = {
                "default": default_message,
                "sms": sms_message,
                "email": email_message,
            }
            response = topic.publish(
                Message=json.dumps(message), Subject=subject, 
MessageStructure="json"
            )
            message_id = response["MessageId"]
            logger.info("Published multi-format message to topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id
```

- For API details, see Publish in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Service class for sending messages using Amazon Simple Notification Service
 (SNS)
class SnsMessageSender
  # Initializes the SnsMessageSender with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Sends a message to a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param message [String] The message to send
  # @return [Boolean] true if message was successfully sent, false otherwise
  def send_message(topic_arn, message)
    @sns_client.publish(topic_arn: topic_arn, message: message)
    @logger.info("Message sent successfully to #{topic_arn}.")
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while sending the message: #{e.message}")
    false
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'SNS_TOPIC_ARN' # Should be replaced with a real topic ARN
  message = 'MESSAGE'         # Should be replaced with the actual message
 content

  sns_client = Aws::SNS::Client.new
  message_sender = SnsMessageSender.new(sns_client)

  @logger.info('Sending message.')
  unless message_sender.send_message(topic_arn, message)
    @logger.error('Message sending failed. Stopping program.')
    exit 1
  end
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see Publish in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);

    Ok(())
}
```

- For API details, see Publish in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_sns->publish(              " oo_result is returned for
 testing purposes. "
          iv_topicarn = iv_topic_arn
          iv_message = iv_message ).
        MESSAGE 'Message published to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see Publish in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS
```

```
        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.publish(
            input: PublishInput(
                message: message,
                topicArn: arn
            )
        )

        guard let messageId = output.messageId else {
            print("No message ID received from Amazon SNS.")
            return
        }

        print("Published message with ID \(messageId)")
```

- For API details, see Publish in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetSMSAttributes` with an AWS SDK or CLI

The following code examples show how to use `SetSMSAttributes`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

How to use Amazon SNS to set the DefaultSMSType attribute.

```
//! Set the default settings for sending SMS messages.
/*!
```

```
  \param smsType: The type of SMS message that you will send by default.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::setSMSType(const Aws::String &smsType,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SetSMSAttributesRequest request;
    request.AddAttributes("DefaultSMSType", smsType);

    const Aws::SNS::Model::SetSMSAttributesOutcome outcome =
 snsClient.SetSMSAttributes(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else {
        std::cerr << "Error while setting SMS Type: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see SetSMSAttributes in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To set SMS message attributes

The following `set-sms-attributes` example sets the default sender ID for SMS messages to `MyName`.

```
aws sns set-sms-attributes \
    --attributes DefaultSenderID=MyName
```

This command produces no output.

- For API details, see SetSMSAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }
    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
 String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                    .attributes(attributes)
                    .build();

            SetSmsAttributesResponse result =
 snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
 Status was "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see SetSMSAttributes in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing
 messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see SetSMSAttributes in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->SetSMSAttributes([
        'attributes' => [
            'DefaultSMSType' => 'Transactional',
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see SetSMSAttributes in *AWS SDK for PHP API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetSubscriptionAttributes` with an AWS SDK or CLI

The following code examples show how to use `SetSubscriptionAttributes`.

CLI

AWS CLI

To set subscription attributes

The following `set-subscription-attributes` example sets the `RawMessageDelivery` attribute to an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name RawMessageDelivery \
    --attribute-value true
```

This command produces no output.

The following `set-subscription-attributes` example sets a `FilterPolicy` attribute to an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name FilterPolicy \
    --attribute-value "{ \"anyMandatoryKey\": [\"any\", \"of\", \"these\"] }"
```

This command produces no output.

The following `set-subscription-attributes` example removes the `FilterPolicy` attribute from an SQS subscription.

```
aws sns set-subscription-attributes \
    --subscription-arn arn:aws:sns:us-
east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
    --attribute-name FilterPolicy \
    --attribute-value "{}"
```

This command produces no output.

- For API details, see SetSubscriptionAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.ArrayList;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class UseMessageFilterPolicy {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <subscriptionArn>

                Where:
                   subscriptionArn - The ARN of a subscription.

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
```

```
                .region(Region.US_EAST_1)
                .build();

        usePolicy(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void usePolicy(SnsClient snsClient, String subscriptionArn) {
        try {
            SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();
            // Add a filter policy attribute with a single value
            fp.addAttribute("store", "example_corp");
            fp.addAttribute("event", "order_placed");

            // Add a prefix attribute
            fp.addAttributePrefix("customer_interests", "bas");

            // Add an anything-but attribute
            fp.addAttributeAnythingBut("customer_interests", "baseball");

            // Add a filter policy attribute with a list of values
            ArrayList<String> attributeValues = new ArrayList<>();
            attributeValues.add("rugby");
            attributeValues.add("soccer");
            attributeValues.add("hockey");
            fp.addAttribute("customer_interests", attributeValues);

            // Add a numeric attribute
            fp.addAttribute("price_usd", "=", 0);

            // Add a numeric attribute with a range
            fp.addAttributeRange("price_usd", ">", 0, "<=", 100);

            // Apply the filter policy attributes to an Amazon SNS subscription
            fp.apply(snsClient, subscriptionArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see SetSubscriptionAttributes in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def add_subscription_filter(subscription, attributes):
        """
        Adds a filter policy to a subscription. A filter policy is a key and a
        list of values that are allowed. When a message is published, it must
 have an
        attribute that passes the filter or it will not be sent to the
 subscription.

        :param subscription: The subscription the filter policy is attached to.
        :param attributes: A dictionary of key-value pairs that define the
 filter.
        """
        try:
            att_policy = {key: [value] for key, value in attributes.items()}
            subscription.set_attributes(
                AttributeName="FilterPolicy",
 AttributeValue=json.dumps(att_policy)
            )
            logger.info("Added filter to subscription %s.", subscription.arn)
        except ClientError:
            logger.exception(
                "Couldn't add filter to subscription %s.", subscription.arn
            )
            raise
```

- For API details, see SetSubscriptionAttributes in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetSubscriptionAttributesRedrivePolicy` with an AWS SDK

The following code example shows how to use `SetSubscriptionAttributesRedrivePolicy`.

Java

SDK for Java 1.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}";

// Set the specified Amazon SQS queue as a dead-letter queue
// of the specified Amazon SNS subscription by setting the RedrivePolicy
 attribute.
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("RedrivePolicy")
    .withAttributeValue(redrivePolicy);
sns.setSubscriptionAttributes(request);
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetTopicAttributes` with an AWS SDK or CLI

The following code examples show how to use `SetTopicAttributes`.

CLI

AWS CLI

To set an attribute for a topic

The following `set-topic-attributes` example sets the `DisplayName` attribute for the specified topic.

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --attribute-name DisplayName \
    --attribute-value MyTopicDisplayName
```

This command produces no output.

- For API details, see SetTopicAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetTopicAttributes {

    public static void main(String[] args) {
        final String usage = """

                Usage:    <attribute> <topicArn> <value>

                Where:
                   attribute - The attribute action to use. Valid parameters are:
 Policy | DisplayName | DeliveryPolicy .
                   topicArn - The ARN of the topic.\s
                   value - The value for the attribute.
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String attribute = args[0];
        String topicArn = args[1];
        String value = args[2];

        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        setTopAttr(snsClient, attribute, topicArn, value);
        snsClient.close();
    }

    public static void setTopAttr(SnsClient snsClient, String attribute, String
 topicArn, String value) {
        try {
            SetTopicAttributesRequest request =
 SetTopicAttributesRequest.builder()
                    .attributeName(attribute)
                    .attributeValue(value)
                    .topicArn(topicArn)
                    .build();

            SetTopicAttributesResponse result =
 snsClient.setTopicAttributes(request);
            System.out.println(
                    "\n\nStatus was " + result.sdkHttpResponse().statusCode() +
 "\n\nTopic " + request.topicArn()
                            + " updated " + request.attributeName() + " to " +
 request.attributeValue());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see SetTopicAttributes in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see SetTopicAttributes in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun setTopAttr(
    attribute: String?,
    topicArnVal: String?,
    value: String?,
) {
    val request =
        SetTopicAttributesRequest {
            attributeName = attribute
            attributeValue = value
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.setTopicAttributes(request)
        println("Topic ${request.topicArn} was updated.")
    }
}
```

- For API details, see SetTopicAttributes in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
$attribute = 'Policy | DisplayName | DeliveryPolicy';
$value = 'First Topic';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->setTopicAttributes([
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see SetTopicAttributes in *AWS SDK for PHP API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Service class to enable an SNS resource with a specified policy
class SnsResourceEnabler
  # Initializes the SnsResourceEnabler with an SNS resource client
  #
  # @param sns_resource [Aws::SNS::Resource] The SNS resource client
  def initialize(sns_resource)
    @sns_resource = sns_resource
    @logger = Logger.new($stdout)
  end

  # Sets a policy on a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param resource_arn [String] The ARN of the resource to include in the policy
  # @param policy_name [String] The name of the policy attribute to set
  def enable_resource(topic_arn, resource_arn, policy_name)
    policy = generate_policy(topic_arn, resource_arn)
    topic = @sns_resource.topic(topic_arn)

    topic.set_attributes({
                           attribute_name: policy_name,
                           attribute_value: policy
                         })
    @logger.info("Policy #{policy_name} set successfully for topic
 #{topic_arn}.")
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Failed to set policy: #{e.message}")
  end

  private

  # Generates a policy string with dynamic resource ARNs
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param resource_arn [String] The ARN of the resource
  # @return [String] The policy as a JSON string
  def generate_policy(topic_arn, resource_arn)
    {
      Version: '2008-10-17',
      Id: '__default_policy_ID',
      Statement: [{
        Sid: '__default_statement_ID',
        Effect: 'Allow',
        Principal: { "AWS": '*' },
        Action: ['SNS:Publish'],
        Resource: topic_arn,
        Condition: {
          ArnEquals: {
            "AWS:SourceArn": resource_arn
          }
        }
      }]
    }.to_json
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'MY_TOPIC_ARN' # Should be replaced with a real topic ARN
  resource_arn = 'MY_RESOURCE_ARN' # Should be replaced with a real resource ARN
  policy_name = 'POLICY_NAME' # Typically, this is "Policy"

  sns_resource = Aws::SNS::Resource.new
  enabler = SnsResourceEnabler.new(sns_resource)

  enabler.enable_resource(topic_arn, resource_arn, policy_name)
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see SetTopicAttributes in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_sns->settopicattributes(
            iv_topicarn = iv_topic_arn
            iv_attributename  = iv_attribute_name
            iv_attributevalue = iv_attribute_value ).
        MESSAGE 'Set/updated SNS topic attributes.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see SetTopicAttributes in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `Subscribe` with an AWS SDK or CLI

The following code examples show how to use `Subscribe`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and publish to a FIFO topic

- Publish messages to queues

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
        /// <summary>
        /// Creates a new subscription to a topic.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object, used
        /// to create an Amazon SNS subscription.</param>
        /// <param name="topicArn">The ARN of the topic to subscribe to.</param>
        /// <returns>A SubscribeResponse object which includes the subscription
        /// ARN for the new subscription.</returns>
        public static async Task<SubscribeResponse> TopicSubscribeAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            SubscribeRequest request = new SubscribeRequest()
            {
                TopicArn = topicArn,
                ReturnSubscriptionArn = true,
                Protocol = "email",
                Endpoint = "recipient@example.com",
            };

            var response = await client.SubscribeAsync(request);

            return response;
        }
```

Subscribe a queue to a topic with optional filters.

```
   /// <summary>
   /// Subscribe a queue to a topic with optional filters.
   /// </summary>
   /// <param name="topicArn">The ARN of the topic.</param>
   /// <param name="useFifoTopic">The optional filtering policy for the 
subscription.</param>
   /// <param name="queueArn">The ARN of the queue.</param>
   /// <returns>The ARN of the new subscription.</returns>
   public async Task<string> SubscribeTopicWithFilter(string topicArn, string? 
filterPolicy, string queueArn)
   {
       var subscribeRequest = new SubscribeRequest()
       {
           TopicArn = topicArn,
           Protocol = "sqs",
           Endpoint = queueArn
       };

       if (!string.IsNullOrEmpty(filterPolicy))
       {
           subscribeRequest.Attributes = new Dictionary<string, string> 
{ { "FilterPolicy", filterPolicy } };
       }

       var subscribeResponse = await 
_amazonSNSClient.SubscribeAsync(subscribeRequest);
       return subscribeResponse.SubscriptionArn;
   }
```

- For API details, see Subscribe in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to an email address.
/*!
  \param topicARN: An SNS topic Amazon Resource Name (ARN).
  \param emailAddress: An email address.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::subscribeEmail(const Aws::String &topicARN,
                                 const Aws::String &emailAddress,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("email");
    request.SetEndpoint(emailAddress);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                  << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
```

```
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

Subscribe a mobile application to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to a mobile app.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param endpointARN: The ARN for a mobile app or device endpoint.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SNS::subscribeApp(const Aws::String &topicARN,
                          const Aws::String &endpointARN,
                          const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("application");
    request.SetEndpoint(endpointARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                  << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }
    return outcome.IsSuccess();
}
```

Subscribe a Lambda function to a topic.

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
 delivery to an AWS Lambda function.
/*!
  \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
  \param lambdaFunctionARN: The ARN for an AWS Lambda function.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::subscribeLambda(const Aws::String &topicARN,
                                  const Aws::String &lambdaFunctionARN,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("lambda");
    request.SetEndpoint(lambdaFunctionARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
 outcome.GetResult().GetSubscriptionArn()
                  << "'." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

Subscribe an SQS queue to a topic.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SNS::SNSClient snsClient(clientConfiguration);

            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);

            Aws::SNS::Model::SubscribeOutcome outcome =
 snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
 outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
 "."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
```

Subscribe with a filter to a topic.

```
        static const Aws::String TONE_ATTRIBUTE("tone");
        static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
 "serious",
                                                       "sincere"};

        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SNS::SNSClient snsClient(clientConfiguration);

            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
                              << "\"'s subscription to the topic \""
                              << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
```

```
               if (askYesNoQuestion(ostringstream.str())) {
                   Aws::String jsonPolicy = getFilterPolicyFromUser();
                   if (!jsonPolicy.empty()) {
                       filteringMessages = true;

                       std::cout << "This is the filter policy for this
subscription."
                                 << std::endl;
                       std::cout << jsonPolicy << std::endl;

                       request.AddAttributes("FilterPolicy", jsonPolicy);
                   }
                   else {
                       std::cout
                               << "Because you did not select any attributes, no
filter "
                               << "will be added to this subscription." <<
std::endl;
                   }
               }
           }  // if (isFifoTopic)
           Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

           if (outcome.IsSuccess()) {
               Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
               std::cout << "The queue '" << queueName
                         << "' has been subscribed to the topic '"
                         << "'" << topicName << "'" << std::endl;
               std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                         << std::endl;
               subscriptionARNS.push_back(subscriptionARN);
           }
           else {
               std::cerr << "Error with TopicsAndQueues::Subscribe. "
                         << outcome.GetError().GetMessage()
                         << std::endl;

               cleanUp(topicARN,
                       queueURLS,
                       subscriptionARNS,
                       snsClient,
                        sqsClient);

                return false;
            }

//! Routine that lets the user select attributes for a subscription filter
 policy.
/*!
 \sa getFilterPolicyFromUser()
 \return Aws::String: The filter policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::getFilterPolicyFromUser() {
    std::cout
            << "You can filter messages by one or more of the following \""
            << TONE_ATTRIBUTE << "\" attributes." << std::endl;

    std::vector<Aws::String> filterSelections;
    int selection;
    do {
        for (size_t j = 0; j < TONES.size(); ++j) {
            std::cout << "  " << (j + 1) << ". " << TONES[j]
                      << std::endl;
        }
        selection = askQuestionForIntRange(
                "Enter a number (or enter zero to stop adding more). ",
                0, static_cast<int>(TONES.size()));

        if (selection != 0) {
            const Aws::String &selectedTone(TONES[selection - 1]);
            // Add the tone to the selection if it is not already added.
            if (std::find(filterSelections.begin(),
                          filterSelections.end(),
                          selectedTone)
                == filterSelections.end()) {
                filterSelections.push_back(selectedTone);
            }
        }
    } while (selection != 0);

    Aws::String result;
    if (!filterSelections.empty()) {
        std::ostringstream jsonPolicyStream;
        jsonPolicyStream << "{ \"" << TONE_ATTRIBUTE << "\": [";
        for (size_t j = 0; j < filterSelections.size(); ++j) {
            jsonPolicyStream << "\"" << filterSelections[j] << "\"";
            if (j < filterSelections.size() - 1) {
                jsonPolicyStream << ",";
            }
        }
        jsonPolicyStream << "] }";

        result = jsonPolicyStream.str();
    }

    return result;
}
```

- For API details, see Subscribe in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To subscribe to a topic

The following `subscribe` command subscribes an email address to the specified topic.

```
aws sns subscribe \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
    --protocol email \
    --notification-endpoint my-email@example.com
```

Output:

```
{
    "SubscriptionArn": "pending confirmation"
}
```

- For API details, see Subscribe in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe a queue to a topic with optional filters.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

- For API details, see Subscribe in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeEmail {
    public static void main(String[] args) {
        final String usage = """
                Usage:     <topicArn> <email>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   email - The email address to use.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String email = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subEmail(snsClient, topicArn, email);
        snsClient.close();
    }

    public static void subEmail(SnsClient snsClient, String topicArn, String
 email) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("email")
                    .endpoint(email)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN: " + result.subscriptionArn() +
 "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Subscribe an HTTP endpoint to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeHTTPS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn> <url>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   url - The HTTPS endpoint that you want to receive
 notifications.
                """;

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String url = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subHTTPS(snsClient, topicArn, url);
        snsClient.close();
    }

    public static void subHTTPS(SnsClient snsClient, String topicArn, String url)
 {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("https")
                    .endpoint(url)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN is " + result.subscriptionArn()
 + "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Subscribe a Lambda function to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeLambda {

    public static void main(String[] args) {

        final String usage = """

                Usage:    <topicArn> <lambdaArn>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   lambdaArn - The ARN of an AWS Lambda function.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String lambdaArn = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnValue = subLambda(snsClient, topicArn, lambdaArn);
        System.out.println("Subscription ARN: " + arnValue);
        snsClient.close();
    }
    public static String subLambda(SnsClient snsClient, String topicArn, String
 lambdaArn) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("lambda")
                    .endpoint(lambdaArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            return result.subscriptionArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see Subscribe in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 a subscription.
 * @param {string} emailAddress - The email address that is subscribed to the
 topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Subscribe a mobile application to a topic.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint
 is created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Subscribe a Lambda function to a topic.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
```

```
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Subscribe an SQS queue to a topic.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Subscribe with a filter to a topic.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute
 set to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see Subscribe in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
suspend fun subEmail(
    topicArnVal: String,
    email: String,
): String {
    val request =
        SubscribeRequest {
            protocol = "email"
            endpoint = email
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        return result.subscriptionArn.toString()
    }
}
```

Subscribe a Lambda function to a topic.

```
suspend fun subLambda(
    topicArnVal: String?,
    lambdaArn: String?,
) {
    val request =
        SubscribeRequest {
            protocol = "lambda"
            endpoint = lambdaArn
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        println(" The subscription Arn is ${result.subscriptionArn}")
    }
}
```

- For API details, see Subscribe in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Subscribe an HTTP endpoint to a topic.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;



/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'https';
$endpoint = 'https://';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSclient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see Subscribe in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def subscribe(topic, protocol, endpoint):
        """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
 subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.

        :param topic: The topic to subscribe to.
        :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
        :param endpoint: The endpoint that receives messages, such as a phone
 number
                         (in E.164 format) for SMS messages, or an email address
 for
                         email messages.
        :return: The newly added subscription.
        """
        try:
            subscription = topic.subscribe(
                Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True
            )
            logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
 topic.arn)
        except ClientError:
            logger.exception(
                "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
 topic.arn
            )
            raise
        else:
            return subscription
```

- For API details, see Subscribe in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
require 'aws-sdk-sns'
require 'logger'
```

```
# Represents a service for creating subscriptions in Amazon Simple Notification
 Service (SNS)
class SubscriptionService
  # Initializes the SubscriptionService with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Attempts to create a subscription to a topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param protocol [String] The subscription protocol (e.g., email)
  # @param endpoint [String] The endpoint that receives the notifications (email
 address)
  # @return [Boolean] true if subscription was successfully created, false
 otherwise
  def create_subscription(topic_arn, protocol, endpoint)
    @sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
 endpoint)
    @logger.info('Subscription created successfully.')
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while creating the subscription: #{e.message}")
    false
  end
end

# Main execution if the script is run directly
if $PROGRAM_NAME == __FILE__
  protocol = 'email'
  endpoint = 'EMAIL_ADDRESS' # Should be replaced with a real email address
  topic_arn = 'TOPIC_ARN'    # Should be replaced with a real topic ARN

  sns_client = Aws::SNS::Client.new
  subscription_service = SubscriptionService.new(sns_client)

  @logger.info('Creating the subscription.')
  unless subscription_service.create_subscription(topic_arn, protocol, endpoint)
    @logger.error('Subscription creation failed. Stopping program.')
    exit 1
  end
```

```
end
```

- For more information, see AWS SDK for Ruby Developer Guide.
- For API details, see Subscribe in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;
    println!("Published message: {:?}", rsp);

    Ok(())
}
```

- For API details, see Subscribe in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
   TRY.
       oo_result = lo_sns->subscribe(                      "oo_result is
returned for testing purposes."
               iv_topicarn = iv_topic_arn
               iv_protocol = 'email'
               iv_endpoint = iv_email_address
               iv_returnsubscriptionarn = abap_true ).
       MESSAGE 'Email address subscribed to SNS topic.' TYPE 'I'.
     CATCH /aws1/cx_snsnotfoundexception.
       MESSAGE 'Topic does not exist.' TYPE 'E'.
     CATCH /aws1/cx_snssubscriptionlmte00.
       MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
   ENDTRY.
```

- For API details, see Subscribe in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.subscribe(
            input: SubscribeInput(
                endpoint: email,
                protocol: "email",
                returnSubscriptionArn: true,
                topicArn: arn
            )
        )

        guard let subscriptionArn = output.subscriptionArn else {
            print("No subscription ARN received from Amazon SNS.")
            return
        }

        print("Subscription \(subscriptionArn) created.")
```

Subscribe a phone number to a topic to receive notifications by SMS.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        let output = try await snsClient.subscribe(
            input: SubscribeInput(
                endpoint: phone,
                protocol: "sms",
                returnSubscriptionArn: true,
                topicArn: arn
            )
        )

        guard let subscriptionArn = output.subscriptionArn else {
            print("No subscription ARN received from Amazon SNS.")
            return
        }

        print("Subscription \(subscriptionArn) created.")
```

- For API details, see Subscribe in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use TagResource with an AWS SDK or CLI

The following code examples show how to use `TagResource`.

CLI

AWS CLI

To add a tag to a topic

The following `tag-resource` example adds a metadata tag to the specified Amazon SNS topic.

```
aws sns tag-resource \
    --resource-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --tags Key=Team,Value=Alpha
```

This command produces no output.

- For API details, see TagResource in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.Tag;
import software.amazon.awssdk.services.sns.model.TagResourceRequest;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AddTags {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn>

                Where:
                   topicArn - The ARN of the topic to which tags are added.

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        addTopicTags(snsClient, topicArn);
        snsClient.close();
    }

    public static void addTopicTags(SnsClient snsClient, String topicArn) {
        try {
            Tag tag = Tag.builder()
                    .key("Team")
                    .value("Development")
                    .build();

            Tag tag2 = Tag.builder()
                    .key("Environment")
                    .value("Gamma")
                    .build();

            List<Tag> tagList = new ArrayList<>();
            tagList.add(tag);
            tagList.add(tag2);

            TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                    .resourceArn(topicArn)
                    .tags(tagList)
                    .build();

            snsClient.tagResource(tagResourceRequest);
            System.out.println("Tags have been added to " + topicArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see TagResource in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun addTopicTags(topicArn: String) {
    val tag =
        Tag {
            key = "Team"
            value = "Development"
        }

    val tag2 =
        Tag {
            key = "Environment"
            value = "Gamma"
        }

    val tagList = mutableListOf<Tag>()
    tagList.add(tag)
    tagList.add(tag2)

    val request =
        TagResourceRequest {
            resourceArn = topicArn
            tags = tagList
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.tagResource(request)
        println("Tags have been added to $topicArn")
    }
}
```

- For API details, see TagResource in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `Unsubscribe` with an AWS SDK or CLI

The following code examples show how to use `Unsubscribe`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Publish messages to queues

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Unsubscribe from a topic by a subscription ARN.

```
    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see Unsubscribe in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Delete a subscription to an Amazon Simple Notification Service (Amazon SNS)
 topic.
/*!
  \param subscriptionARN: The Amazon Resource Name (ARN) for an Amazon SNS topic
 subscription.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::unsubscribe(const Aws::String &subscriptionARN,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::UnsubscribeRequest request;
    request.SetSubscriptionArn(subscriptionARN);

    const Aws::SNS::Model::UnsubscribeOutcome outcome =
 snsClient.Unsubscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else {
        std::cerr << "Error while unsubscribing " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see Unsubscribe in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To unsubscribe from a topic

The following `unsubscribe` example deletes the specified subscription from a topic.

```
aws sns unsubscribe \
    --subscription-arn arn:aws:sns:us-west-2:0123456789012:my-
topic:8a21d249-4329-4871-acc6-7be709c6ea7f
```

This command produces no output.

- For API details, see Unsubscribe in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class Unsubscribe {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <subscriptionArn>

                Where:
                   subscriptionArn - The ARN of the subscription to delete.
                """;

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        unSub(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                    .subscriptionArn(subscriptionArn)
                    .build();

            UnsubscribeResponse result = snsClient.unsubscribe(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode()
                    + "\n\nSubscription was removed for " +
 request.subscriptionArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see Unsubscribe in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client in a separate module and export it.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
 it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Import the SDK and client modules and call the API.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see Unsubscribe in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun unSub(subscriptionArnVal: String) {
    val request =
        UnsubscribeRequest {
            subscriptionArn = subscriptionArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for ${request.subscriptionArn}")
    }
}
```

- For API details, see Unsubscribe in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Deletes a subscription to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $SnSclient->unsubscribe([
        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Unsubscribe in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    @staticmethod
    def delete_subscription(subscription):
        """
        Unsubscribes and deletes a subscription.
        """
        try:
            subscription.delete()
            logger.info("Deleted subscription %s.", subscription.arn)
        except ClientError:
            logger.exception("Couldn't delete subscription %s.",
 subscription.arn)
            raise
```

- For API details, see Unsubscribe in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   TRY.
       lo_sns->unsubscribe( iv_subscriptionarn = iv_subscription_arn ).
       MESSAGE 'Subscription deleted.' TYPE 'I'.
     CATCH /aws1/cx_snsnotfoundexception.
       MESSAGE 'Subscription does not exist.' TYPE 'E'.
     CATCH /aws1/cx_snsinvalidparameterex.
       MESSAGE 'Subscription with "PendingConfirmation" status cannot be
deleted/unsubscribed. Confirm subscription before performing unsubscribe
operation.' TYPE 'E'.
   ENDTRY.
```

- For API details, see Unsubscribe in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSNS

        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(
                subscriptionArn: arn
            )
        )

        print("Unsubscribed.")
```

- For API details, see Unsubscribe in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon SNS using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon SNS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon SNS or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Build an application to submit data to a DynamoDB table
- Build a publish and subscription application that translates messages
- Create a platform endpoint for Amazon SNS push notifications using an AWS SDK
- Create a photo asset management application that lets users manage photos using labels
- Create an Amazon Textract explorer application
- Create and publish to a FIFO Amazon SNS topic using an AWS SDK

- Detect people and objects in a video with Amazon Rekognition using an AWS SDK
- Publish SMS messages to an Amazon SNS topic using an AWS SDK
- Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK
- Publish an Amazon SNS SMS text message using an AWS SDK
- Publish Amazon SNS messages to Amazon SQS queues using an AWS SDK
- Use API Gateway to invoke a Lambda function
- Use scheduled events to invoke a Lambda function

Build an application to submit data to a DynamoDB table

The following code examples show how to build an application that submits data to an Amazon DynamoDB table and notifies you when a user updates the table.

Java

SDK for Java 2.x

Shows how to create a dynamic web application that submits data using the Amazon DynamoDB Java API and sends a text message using the Amazon Simple Notification Service Java API.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

This example shows how to build an app that enables users to submit data to an Amazon DynamoDB table, and send a text message to the administrator using Amazon Simple Notification Service (Amazon SNS).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

This example is also available in the AWS SDK for JavaScript v3 developer guide.

Services used in this example

- DynamoDB
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Build a publish and subscription application that translates messages

The following code examples show how to create an application that has subscription and publish functionality and translates messages.

.NET

SDK for .NET

Shows how to use the Amazon Simple Notification Service .NET API to create a web application that has subscription and publish functionality. In addition, this example application also translates messages.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon SNS
- Amazon Translate

Java

SDK for Java 2.x

Shows how to use the Amazon Simple Notification Service Java API to create a web application that has subscription and publish functionality. In addition, this example application also translates messages.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For complete source code and instructions on how to set up and run the example that uses the Java Async API, see the full example on GitHub.

Services used in this example

- Amazon SNS
- Amazon Translate

Kotlin

SDK for Kotlin

Shows how to use the Amazon SNS Kotlin API to create an application that has subscription and publish functionality. In addition, this example application also translates messages.

For complete source code and instructions on how to create a web app, see the full example on GitHub.

For complete source code and instructions on how to create a native Android app, see the full example on GitHub.

Services used in this example

- Amazon SNS
- Amazon Translate

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a platform endpoint for Amazon SNS push notifications using an AWS SDK

The following code examples show how to create a platform endpoint for Amazon SNS push notifications.

CLI

AWS CLI

To create a platform application endpoint

The following `create-platform-endpoint` example creates an endpoint for the specified platform application using the specified token.

```
aws sns create-platform-endpoint \
    --platform-application-arn arn:aws:sns:us-west-2:123456789012:app/GCM/
MyApplication \
    --token EXAMPLE12345...
```

Output:

```
{
      "EndpointArn": "arn:aws:sns:us-west-2:1234567890:endpoint/GCM/
MyApplication/12345678-abcd-9012-efgh-345678901234"
}
```

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointRequest;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * In addition, create a platform application using the AWS Management Console.
 * See this doc topic:
 *
 * https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-register.html
 *
 * Without the values created by following the previous link, this code examples
 * does not work.
 */

public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = """

            Usage:     <token> <platformApplicationArn>

            Where:
               token - The device token or registration ID of the mobile device.
 This is a unique
               identifier provided by the device platform (e.g., Apple Push
 Notification Service (APNS) for iOS devices, Firebase Cloud Messaging (FCM)
               for Android devices) when the mobile app is registered to receive
 push notifications.

               platformApplicationArn - The ARN value of platform application.
 You can get this value from the AWS Management Console.\s

            """;

        if (args.length != 2) {
            System.out.println(usage);
            return;
        }

        String token = args[0];
        String platformApplicationArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        createEndpoint(snsClient, token, platformApplicationArn);
    }
    public static void createEndpoint(SnsClient snsClient, String token, String
 platformApplicationArn) {
        System.out.println("Creating platform endpoint with token " + token);
        try {
            CreatePlatformEndpointRequest endpointRequest =
 CreatePlatformEndpointRequest.builder()
                    .token(token)
                    .platformApplicationArn(platformApplicationArn)
                    .build();

            CreatePlatformEndpointResponse response =
 snsClient.createPlatformEndpoint(endpointRequest);
            System.out.println("The ARN of the endpoint is " +
 response.endpointArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a photo asset management application that lets users manage photos using labels

The following code examples show how to create a serverless application that lets users manage photos using labels.

.NET

SDK for .NET

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) with Amazon Textract to detect text, form, and table elements in a document image. The input image and Amazon Textract output are shown in a Tkinter application that lets you explore the detected elements.

- Submit a document image to Amazon Textract and explore the output of detected elements.
- Submit images directly to Amazon Textract or through an Amazon Simple Storage Service (Amazon S3) bucket.
- Use asynchronous APIs to start a job that publishes a notification to an Amazon Simple Notification Service (Amazon SNS) topic when the job completes.
- Poll an Amazon Simple Queue Service (Amazon SQS) queue for a job completion message and display the results.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create and publish to a FIFO Amazon SNS topic using an AWS SDK

The following code examples show how to create and publish to a FIFO Amazon SNS topic.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example

- creates an Amazon SNS FIFO topic, two Amazon SQS FIFO queues, and one Standard queue.
- subscribes the queues to the topic and publishes a message to the topic.

The test verifies the receipt of the message to each queue. The complete example also shows the addition of access policies and deletes the resources at the end.

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
```

```
   public final static SqsClient sqsClient = SqsClient.create();

   public static void main(String[] args) {

       final String usage = "\n" +
           "Usage: " +
           "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
<analyticsQueueName>\n\n" +
           "Where:\n" +
           "   fifoTopicName - The name of the FIFO topic that you want to
create. \n\n" +
           "   wholesaleQueueARN - The name of a SQS FIFO queue that will be
created for the wholesale consumer. \n\n"
           +
           "   retailQueueARN - The name of a SQS FIFO queue that will created
for the retail consumer. \n\n" +
           "   analyticsQueueARN - The name of a SQS standard queue that will be
created for the analytics consumer. \n\n";
       if (args.length != 4) {
           System.out.println(usage);
           System.exit(1);
       }

       final String fifoTopicName = args[0];
       final String wholeSaleQueueName = args[1];
       final String retailQueueName = args[2];
       final String analyticsQueueName = args[3];

       // For convenience, the QueueData class holds metadata about a queue:
ARN, URL,
       // name and type.
       List<QueueData> queues = List.of(
           new QueueData(wholeSaleQueueName, QueueType.FIFO),
           new QueueData(retailQueueName, QueueType.FIFO),
           new QueueData(analyticsQueueName, QueueType.Standard));

       // Create queues.
       createQueues(queues);

       // Create a topic.
       String topicARN = createFIFOTopic(fifoTopicName);

       // Subscribe each queue to the topic.
       subscribeQueues(queues, topicARN);
```

```
        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

        // Publish a sample price update message with payload.
        publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

        // Clean up resources.
        deleteSubscriptions(queues);
        deleteQueues(queues);
        deleteTopic(topicARN);
    }

    public static String createFIFOTopic(String topicName) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = Map.of(
                "FifoTopic", "true",
                "ContentBasedDeduplication", "false",
                "FifoThroughputScope", "MessageGroup");

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            String topicArn = response.topicArn();
            System.out.println("The topic ARN is" + topicArn);

            return topicArn;

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void subscribeQueues(List<QueueData> queues, String topicARN) {
        queues.forEach(queue -> {
            SubscribeRequest subscribeRequest = SubscribeRequest.builder()
                .topicArn(topicARN)
                .endpoint(queue.queueARN)
                .protocol("sqs")
                .build();

            // Subscribe to the endpoint by using the SNS service client.
            // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
            // topic.
            SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
            System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]");
            queue.subscriptionARN = subscribeResponse.subscriptionArn();
        });
    }

    public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

            MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
                .dataType("String")
                .stringValue(attributeValue)
                .build();

            Map<String, MessageAttributeValue> attributes = new HashMap<>();
            attributes.put(attributeName, msgAttValue);
            PublishRequest pubRequest = PublishRequest.builder()
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
            System.out.println(response.sequenceNumber());
            System.out.println("Message was published to " + topicArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon SNS FIFO topic, subscribe Amazon SQS FIFO and standard queues to the topic, and publish a message to the topic.

```
def usage_demo():
    """Shows how to subscribe queues to a FIFO topic."""
    print("-" * 88)
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")
    print("-" * 88)

    sns = boto3.resource("sns")
    sqs = boto3.resource("sqs")
    fifo_topic_wrapper = FifoTopicWrapper(sns)
    sns_wrapper = SnsWrapper(sns)

    prefix = "sqs-subscribe-demo-"
    queues = set()
    subscriptions = set()

    wholesale_queue = sqs.create_queue(
        QueueName=prefix + "wholesale.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(wholesale_queue)
    print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

    retail_queue = sqs.create_queue(
        QueueName=prefix + "retail.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(retail_queue)
    print(f"Created FIFO queue with URL: {retail_queue.url}.")

    analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
 Attributes={})
    queues.add(analytics_queue)
    print(f"Created standard queue with URL: {analytics_queue.url}.")

    topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
    print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])

    print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        sub = fifo_topic_wrapper.subscribe_queue_to_topic(
            topic, q.attributes["QueueArn"]
        )
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")

    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)



class FifoTopicWrapper:
    """Encapsulates Amazon SNS FIFO topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.
        Topic names must be made up of only uppercase and lowercase ASCII
letters,
        numbers, underscores, and hyphens, and must be between 1 and 256
characters long.
        For a FIFO topic, the name must end with the .fifo suffix.

        :param topic_name: The name for the topic.
        :return: The new topic.
        """
        try:
            topic = self.sns_resource.create_topic(
                Name=topic_name,
                Attributes={
                    "FifoTopic": str(True),
                    "ContentBasedDeduplication": str(False),
                    "FifoThroughputScope": "MessageGroup",
                },
            )
            logger.info("Created FIFO topic with name=%s.", topic_name)
            return topic
        except ClientError as error:
            logger.exception("Couldn't create topic with name=%s!", topic_name)
            raise error


    @staticmethod
    def add_access_policy(queue, topic_arn):
        """
        Add the necessary access policy to a queue, so
        it can receive messages from a topic.

        :param queue: The queue resource.
        :param topic_arn: The ARN of the topic.
        :return: None.
        """
        try:
            queue.set_attributes(
                Attributes={
                    "Policy": json.dumps(
                        {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Sid": "test-sid",
                                    "Effect": "Allow",
                                    "Principal": {"AWS": "*"},
                                    "Action": "SQS:SendMessage",
                                    "Resource": queue.attributes["QueueArn"],
                                    "Condition": {
                                        "ArnLike": {"aws:SourceArn": topic_arn}
                                    },
                                }
                            ],
                        }
                    )
                }
            )
            logger.info("Added trust policy to the queue.")
        except ClientError as error:
            logger.exception("Couldn't add trust policy to the queue!")
            raise error


    @staticmethod
    def subscribe_queue_to_topic(topic, queue_arn):
        """
        Subscribe a queue to a topic.

        :param topic: The topic resource.
        :param queue_arn: The ARN of the queue.
        :return: The subscription resource.
        """
        try:
            subscription = topic.subscribe(
                Protocol="sqs",
                Endpoint=queue_arn,
            )
            logger.info("The queue is subscribed to the topic.")
            return subscription
        except ClientError as error:
            logger.exception("Couldn't subscribe queue to topic!")
            raise error


    @staticmethod
    def publish_price_update(topic, payload, group_id):
        """
        Compose and publish a message that updates the wholesale price.
        :param topic: The topic to publish to.
        :param payload: The message to publish.
        :param group_id: The group ID for the message.
        :return: The ID of the message.
        """
        try:
            att_dict = {"business": {"DataType": "String", "StringValue":
"wholesale"}}
            dedup_id = uuid.uuid4()
            response = topic.publish(
                Subject="Price Update",
                Message=payload,
                MessageAttributes=att_dict,
                MessageGroupId=group_id,
                MessageDeduplicationId=str(dedup_id),
            )
            message_id = response["MessageId"]
            logger.info("Published message to topic %s.", topic.arn)
        except ClientError as error:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise error
        return message_id


    @staticmethod
    def delete_queue(queue):
        """
        Removes an SQS queue. When run against an AWS account, it can take up to
        60 seconds before the queue is actually deleted.

        :param queue: The queue to delete.
        :return: None
        """
        try:
            queue.delete()
            logger.info("Deleted queue with URL=%s.", queue.url)
        except ClientError as error:
            logger.exception("Couldn't delete queue with URL=%s!", queue.url)
            raise error
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a FIFO topic, subscribe an Amazon SQS FIFO queue to the topic, and publish a message to an Amazon SNS topic.

```
    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_w( iv_value =
 'true' ).
    INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

    TRY.
        DATA(lo_create_result) = lo_sns->createtopic(
               iv_name = iv_topic_name
               it_attributes = lt_tpc_attributes ).
        DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
        ov_topic_arn = lv_topic_arn.                                    "
 ov_topic_arn is returned for testing purposes. "
        MESSAGE 'FIFO topic created' TYPE 'I'.
      CATCH /aws1/cx_snstopiclimitexcdex.
```

```
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
 SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
 to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
               iv_topicarn = lv_topic_arn
               iv_protocol = 'sqs'
               iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.                      "
 ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
 'String'

 iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
             iv_topicarn = lv_topic_arn
             iv_message = 'The price of your mobile plan has been increased from
 $19 to $23'
             iv_subject = 'Changes to mobile plan'
             iv_messagegroupid = 'Update-2'
             iv_messagededuplicationid = 'Update-2.1'
```

```
            it_messageattributes = lt_msg_attributes ).
       ov_message_id = lo_result->get_messageid( ).                    "
ov_message_id is returned for testing purposes. "
       MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
     CATCH /aws1/cx_snsnotfoundexception.
       MESSAGE 'Topic does not exist.' TYPE 'E'.
   ENDTRY.
```

- For API details, see the following topics in *AWS SDK for SAP ABAP API reference*.
  - CreateTopic
  - Publish
  - Subscribe

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Detect people and objects in a video with Amazon Rekognition using an AWS SDK

The following code examples show how to detect people and objects in a video with Amazon Rekognition.

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition

- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python (Boto3)

Use Amazon Rekognition to detect faces, objects, and people in videos by starting asynchronous detection jobs. This example also configures Amazon Rekognition to notify an Amazon Simple Notification Service (Amazon SNS) topic when jobs complete and subscribes an Amazon Simple Queue Service (Amazon SQS) queue to the topic. When the queue receives a message about a job, the job is retrieved and the results are output.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Publish SMS messages to an Amazon SNS topic using an AWS SDK

The following code example shows how to:

- Create an Amazon SNS topic.
- Subscribe phone numbers to the topic.

- Publish SMS messages to the topic so that all subscribed phone numbers receive the message at once.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

Create a topic and return its ARN.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicName>

                Where:
                   topicName - The name of the topic to create (for example,
 mytopic).

                """;
        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicName = args[0];
        System.out.println("Creating a topic with name: " + topicName);
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String arnVal = createSNSTopic(snsClient, topicName);
        System.out.println("The topic ARN is" + arnVal);
        snsClient.close();
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                    .name(topicName)
                    .build();

            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

Subscribe an endpoint to a topic.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SubscribeTextSMS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <topicArn> <phoneNumber>

                Where:
                   topicArn - The ARN of the topic to subscribe.
                   phoneNumber - A mobile phone number that receives
 notifications (for example, +1XXX5550100).
                """;

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        subTextSNS(snsClient, topicArn, phoneNumber);
        snsClient.close();
    }

    public static void subTextSNS(SnsClient snsClient, String topicArn, String
 phoneNumber) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                    .protocol("sms")
                    .endpoint(phoneNumber)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN: " + result.subscriptionArn() +
 "\n\n Status is "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Set attributes on the message, such as the ID of the sender, the maximum price, and its type. Message attributes are optional.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
 String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                    .attributes(attributes)
                    .build();

            SetSmsAttributesResponse result =
 snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
 Status was "
                    + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Publish a message to a topic. The message is sent to every subscriber.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <message> <phoneNumber>

                Where:
                   message - The message text to send.
                   phoneNumber - The mobile phone number to which a message is
 sent (for example, +1XXX5550100).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
 phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .phoneNumber(phoneNumber)
                    .build();

            PublishResponse result = snsClient.publish(request);
            System.out
                    .println(result.messageId() + " Message sent. Status was " +
 result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Publish a large message to Amazon SNS with Amazon S3 using an AWS SDK

The following code example shows how to publish a large message to Amazon SNS using Amazon S3 to store the message payload.

Java

SDK for Java 1.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

To publish a large message, use the Amazon SNS Extended Client Library for Java. The message that you send references an Amazon S3 object containing the actual message content.

```
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;
import com.amazonaws.services.sns.util.Topics;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import software.amazon.sns.AmazonSNSExtendedClient;
import software.amazon.sns.SNSExtendedClientConfiguration;

public class Example {

        public static void main(String[] args) {
                final String BUCKET_NAME = "extended-client-bucket";
                final String TOPIC_NAME = "extended-client-topic";
                final String QUEUE_NAME = "extended-client-queue";
                final Regions region = Regions.DEFAULT_REGION;

                // Message threshold controls the maximum message size that will
 be allowed to
                // be published
                // through SNS using the extended client. Payload of messages
 exceeding this
                // value will be stored in
                // S3. The default value of this parameter is 256 KB which is the
 maximum
                // message size in SNS (and SQS).
                final int EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD = 32;

                // Initialize SNS, SQS and S3 clients
                final AmazonSNS snsClient =
 AmazonSNSClientBuilder.standard().withRegion(region).build();
                final AmazonSQS sqsClient =
 AmazonSQSClientBuilder.standard().withRegion(region).build();
                final AmazonS3 s3Client =
 AmazonS3ClientBuilder.standard().withRegion(region).build();

                // Create bucket, topic, queue and subscription
                s3Client.createBucket(BUCKET_NAME);
                final String topicArn = snsClient.createTopic(
                                new
 CreateTopicRequest().withName(TOPIC_NAME)).getTopicArn();
                final String queueUrl = sqsClient.createQueue(
                                new
 CreateQueueRequest().withQueueName(QUEUE_NAME)).getQueueUrl();
                final String subscriptionArn = Topics.subscribeQueue(
                                snsClient, sqsClient, topicArn, queueUrl);
```

```
               // To read message content stored in S3 transparently through SQS 
extended
               // client,
               // set the RawMessageDelivery subscription attribute to TRUE
               final SetSubscriptionAttributesRequest 
subscriptionAttributesRequest = new SetSubscriptionAttributesRequest();
                
subscriptionAttributesRequest.setSubscriptionArn(subscriptionArn);
                
subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
               subscriptionAttributesRequest.setAttributeValue("TRUE");
                
snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);

               // Initialize SNS extended client
               // PayloadSizeThreshold triggers message content storage in S3 
when the
               // threshold is exceeded
               // To store all messages content in S3, use AlwaysThroughS3 flag
               final SNSExtendedClientConfiguration 
snsExtendedClientConfiguration = new SNSExtendedClientConfiguration()
                               .withPayloadSupportEnabled(s3Client, BUCKET_NAME)
                                
.withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
               final AmazonSNSExtendedClient snsExtendedClient = new 
AmazonSNSExtendedClient(snsClient,
                               snsExtendedClientConfiguration);

               // Publish message via SNS with storage in S3
               final String message = "This message is stored in S3 as it 
exceeds the threshold of 32 bytes set above.";
               snsExtendedClient.publish(topicArn, message);

               // Initialize SQS extended client
               final ExtendedClientConfiguration sqsExtendedClientConfiguration 
= new ExtendedClientConfiguration()
                               .withPayloadSupportEnabled(s3Client, 
BUCKET_NAME);
               final AmazonSQSExtendedClient sqsExtendedClient = new 
AmazonSQSExtendedClient(sqsClient,
                               sqsExtendedClientConfiguration);

               // Read the message from the queue
```

```
                final ReceiveMessageResult result =
 sqsExtendedClient.receiveMessage(queueUrl);
                System.out.println("Received message is " +
 result.getMessages().get(0).getBody());
        }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Publish an Amazon SNS SMS text message using an AWS SDK

The following code examples show how to publish SMS messages using Amazon SNS.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
        /// Initializes a new instance of the <see cref="SNSMessage"/> class.
        /// Constructs a new SNSMessage object initializing the Amazon Simple
        /// Notification Service (Amazon SNS) client using the supplied
        /// Region endpoint.
        /// </summary>
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
        /// sending test messages with this object.</param>
        public SNSMessage(RegionEndpoint regionEndpoint)
        {
            snsClient = new
 AmazonSimpleNotificationServiceClient(regionEndpoint);
        }

        /// <summary>
        /// Sends the SMS message passed in the text parameter to the phone
 number
        /// in phoneNum.
        /// </summary>
        /// <param name="phoneNum">The ten-digit phone number to which the text
        /// message will be sent.</param>
        /// <param name="text">The text of the message to send.</param>
        /// <returns>Async task.</returns>
        public async Task SendTextMessageAsync(string phoneNum, string text)
        {
            if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
            {
                return;
            }

            // Now actually send the message.
            var request = new PublishRequest
            {
                Message = text,
                PhoneNumber = phoneNum,
            };

            try
            {
                var response = await snsClient.PublishAsync(request);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error sending message: {ex}");
            }
        }
    }
```

```
}
```

- For API details, see Publish in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 * Publish SMS: use Amazon Simple Notification Service (Amazon SNS) to send an
 SMS text message to a phone number.
 * Note: This requires additional AWS configuration prior to running example.
 *
 *  NOTE: When you start using Amazon SNS to send SMS messages, your AWS account
 is in the SMS sandbox and you can only
 *  use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
latest/dg/sns-sms-sandbox.html.
 *  NOTE: If destination is in the US, you also have an additional restriction
 that you have use a dedicated
 *  origination ID (phone number). You can request an origination number using
 Amazon Pinpoint for a fee.
 *  See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-
origination-numbers-with-amazon-sns/
 *  for more information.
 *
 *  <phone_number_value> input parameter uses E.164 format.
 *  For example, in United States, this input value should be of the form:
 +12223334444
 */

//! Send an SMS text message to a phone number.
/*!
  \param message: The message to publish.
  \param phoneNumber: The phone number of the recipient in E.164 format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::publishSms(const Aws::String &message,
                             const Aws::String &phoneNumber,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetPhoneNumber(phoneNumber);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with message id, '"
                  << outcome.GetResult().GetMessageId() << "'."
                  << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see Publish in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <message> <phoneNumber>

                Where:
                   message - The message text to send.
                   phoneNumber - The mobile phone number to which a message is
 sent (for example, +1XXX5550100).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
 phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                    .message(message)
                    .phoneNumber(phoneNumber)
                    .build();

            PublishResponse result = snsClient.publish(request);
            System.out
                    .println(result.messageId() + " Message sent. Status was " +
 result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see Publish in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun pubTextSMS(
    messageVal: String?,
    phoneNumberVal: String?,
) {
    val request =
        PublishRequest {
            message = messageVal
            phoneNumber = phoneNumberVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
```

```
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

- For API details, see Publish in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Sends a text message (SMS message) directly to a phone number using Amazon
 SNS.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';
```

```
try {
    $result = $SnSclient->publish([
        'Message' => $message,
        'PhoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource


    def publish_text_message(self, phone_number, message):
        """
        Publishes a text message directly to a phone number without need for a
        subscription.
        :param phone_number: The phone number that receives the message. This
must be
                             in E.164 format. For example, a United States phone
                             number might be +12065550101.
        :param message: The message to send.
        :return: The ID of the message.
        """
        try:
            response = self.sns_resource.meta.client.publish(
                PhoneNumber=phone_number, Message=message
            )
            message_id = response["MessageId"]
            logger.info("Published message to %s.", phone_number)
        except ClientError:
            logger.exception("Couldn't publish message to %s.", phone_number)
            raise
        else:
            return message_id
```

- For API details, see Publish in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Publish Amazon SNS messages to Amazon SQS queues using an AWS SDK

The following code examples show how to:

- Create topic (FIFO or non-FIFO).
- Subscribe several queues to the topic with an option to apply a filter.
- Publish messages to the topic.
- Poll the queues for messages received.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is 
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario: 
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and 
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for 
configuring the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou can then post to the topic and see the 
results in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                          $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                          $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic,
 deduplication is supported." +
                              $"\r\nDeduplication IDs are either set in the
 message or automatically generated " +
                              $"\r\nfrom content using a hash function.\r\n" +
                              $"\r\nIf a message is successfully published to an
 SNS FIFO topic, any message " +
                              $"\r\npublished and determined to have the same
 deduplication ID, " +
                              $"\r\nwithin the five-minute deduplication
 interval, is accepted but not delivered.\r\n" +
                              $"\r\nFor more information about deduplication, " +
                              $"\r\nsee https://docs.aws.amazon.com/sns/latest/
dg/fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName,
 _useFifoTopic, _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS
 queue: ", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
 appended to the queue name.");
                }

                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
 _useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must
 be attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
 queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" :
 "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is
 a sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must
 set a message group ID." +
                                  "\r\nAll messages within the same group will be
 received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID
 for this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
 deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this
 message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID
 for this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
```

```
                   }
               }

               var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                   _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

               Console.WriteLine($"Message published with id {messageID}.");
           }

           keepSendingMessages = GetYesNoResponse("Send another message?",
false);
       }
   }

   /// <summary>
   /// Poll for the published messages to see the results of the user's choices.
   /// </summary>
   /// <returns>Async task.</returns>
   public static async Task<List<Message>> PollForMessages(string queueUrl)
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                         "\r\nPress any key to continue.");
       if (UseConsole)
       {
           Console.ReadLine();
       }

       var moreMessages = true;
       var messages = new List<Message>();
       while (moreMessages)
       {
           var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

           moreMessages = newMessages.Any();
           if (moreMessages)
           {
               messages.AddRange(newMessages);
           }
       }
       Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

       foreach (var message in messages)
       {
           Console.WriteLine("\tMessage:" +
                             $"\n\t{message.Body}");
       }

       Console.WriteLine(new string('-', 80));
       return messages;
   }

   /// <summary>
   /// Delete the message using handles in a batch.
   /// </summary>
   /// <returns>Async task.</returns>
   public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
       await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
       Console.WriteLine(new string('-', 80));
   }

   /// <summary>
   /// Clean up the resources from the scenario.
   /// </summary>
   /// <returns>Async task.</returns>
   private static async Task CleanupResources()
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine($"Clean up resources.");

       try
       {
           foreach (var queueUrl in _queueUrls)
           {
               if (!string.IsNullOrEmpty(queueUrl))
               {
                   var deleteQueue =
                       GetYesNoResponse($"Delete queue with url {queueUrl}?");
```

```
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
 {ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer =
 true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Create a class that wraps Amazon SQS operations.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;
    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await
 _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
 topicArn, string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\":
 \"{topicArn}\"" +
                                            "}" +
                                        "}" +
                                "}]" +
                             "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
 queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Create a class that wraps Amazon SNS operations.

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
```

```
           if (!topicName.EndsWith(".fifo"))
           {
               createTopicRequest.Name = topicName + ".fifo";
           }

           // Add the attributes from the method parameters.
           createTopicRequest.Attributes = new Dictionary<string, string>
           {
               { "FifoTopic", "true" }
           };
           if (useContentBasedDeduplication)
           {
               createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
           }
       }

       var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
       return createResponse.TopicArn;
   }

   /// <summary>
   /// Subscribe a queue to a topic with optional filters.
   /// </summary>
   /// <param name="topicArn">The ARN of the topic.</param>
   /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
   /// <param name="queueArn">The ARN of the queue.</param>
   /// <returns>The ARN of the new subscription.</returns>
   public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
   {
       var subscribeRequest = new SubscribeRequest()
       {
           TopicArn = topicArn,
           Protocol = "sqs",
           Endpoint = queueArn
       };

       if (!string.IsNullOrEmpty(filterPolicy))
       {
           subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
```

```
        }

        var subscribeResponse = await
 _amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
 message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
 attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await
 _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Workflow for messaging with topics and queues using Amazon SNS and Amazon
 SQS.
/*!
 \param clientConfig Aws client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::TopicsAndQueues::messagingWithTopicsAndQueues(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << "Welcome to messaging with topics and queues." << std::endl;
    printAsterisksLine();
    std::cout << "In this workflow, you will create an SNS topic and subscribe "
```

```
             << NUMBER_OF_QUEUES <<
             " SQS queues to the topic." << std::endl;
   std::cout
           << "You can select from several options for configuring the topic and
the subscriptions for the "
           << NUMBER_OF_QUEUES << " queues." << std::endl;
   std::cout << "You can then post to the topic and see the results in the
queues."
             << std::endl;

   Aws::SNS::SNSClient snsClient(clientConfiguration);

   printAsterisksLine();

   std::cout << "SNS topics can be configured as FIFO (First-In-First-Out)."
             << std::endl;
   std::cout
           << "FIFO topics deliver messages in order and support deduplication
and message filtering."
           << std::endl;
   bool isFifoTopic = askYesNoQuestion(
           "Would you like to work with FIFO topics? (y/n) ");

   bool contentBasedDeduplication = false;
   Aws::String topicName;
   if (isFifoTopic) {
       printAsterisksLine();
       std::cout << "Because you have chosen a FIFO topic, deduplication is
supported."
                 << std::endl;
       std::cout
               << "Deduplication IDs are either set in the message or
automatically generated "
               << "from content using a hash function." << std::endl;
       std::cout
               << "If a message is successfully published to an SNS FIFO topic,
any message "
               << "published and determined to have the same deduplication ID, "
               << std::endl;
       std::cout
               << "within the five-minute deduplication interval, is accepted
but not delivered."
               << std::endl;
       std::cout
```

```
                << "For more information about deduplication, "
                << "see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html."
                << std::endl;
        contentBasedDeduplication = askYesNoQuestion(
                "Use content-based deduplication instead of entering a
 deduplication ID? (y/n) ");
    }

    printAsterisksLine();

    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::Vector<Aws::String> queueURLS;
    Aws::Vector<Aws::String> subscriptionARNS;

    Aws::String topicARN;
    {
        topicName = askQuestion("Enter a name for your SNS topic. ");

        // 1.  Create an Amazon SNS topic, either FIFO or non-FIFO.
        Aws::SNS::Model::CreateTopicRequest request;

        if (isFifoTopic) {
            request.AddAttributes("FifoTopic", "true");
            if (contentBasedDeduplication) {
                request.AddAttributes("ContentBasedDeduplication", "true");
            }
            topicName = topicName + FIFO_SUFFIX;

            std::cout
                    << "Because you have selected a FIFO topic, '.fifo' must be
 appended to the topic name."
                    << std::endl;
        }

        request.SetName(topicName);

        Aws::SNS::Model::CreateTopicOutcome outcome =
 snsClient.CreateTopic(request);

        if (outcome.IsSuccess()) {
            topicARN = outcome.GetResult().GetTopicArn();
            std::cout << "Your new topic with the name '" << topicName
```

```
                     << "' and the topic Amazon Resource Name (ARN) " <<
std::endl;
           std::cout << "'" << topicARN << "' has been created." << std::endl;

       }
       else {
           std::cerr << "Error with TopicsAndQueues::CreateTopic. "
                     << outcome.GetError().GetMessage()
                     << std::endl;

           cleanUp(topicARN,
                   queueURLS,
                   subscriptionARNS,
                   snsClient,
                   sqsClient);

           return false;
       }
   }

   printAsterisksLine();

   std::cout << "Now you will create " << NUMBER_OF_QUEUES
             << " SQS queues to subscribe to the topic." << std::endl;
   Aws::Vector<Aws::String> queueNames;
   bool filteringMessages = false;
   bool first = true;
   for (int i = 1; i <= NUMBER_OF_QUEUES; ++i) {
       Aws::String queueURL;
       Aws::String queueName;
       {
           printAsterisksLine();
           std::ostringstream ostringstream;
           ostringstream << "Enter a name for " << (first ? "an" : "the next")
                         << " SQS queue. ";
           queueName = askQuestion(ostringstream.str());

           // 2.  Create an SQS queue.
           Aws::SQS::Model::CreateQueueRequest request;
           if (isFifoTopic) {

request.AddAttributes(Aws::SQS::Model::QueueAttributeName::FifoQueue,
                                     "true");
               queueName = queueName + FIFO_SUFFIX;
```

```
                if (first) // Only explain this once.
                {
                    std::cout
                            << "Because you are creating a FIFO SQS queue,
'.fifo' must "
                            << "be appended to the queue name." << std::endl;
                }
            }

            request.SetQueueName(queueName);
            queueNames.push_back(queueName);

            Aws::SQS::Model::CreateQueueOutcome outcome =
                    sqsClient.CreateQueue(request);

            if (outcome.IsSuccess()) {
                queueURL = outcome.GetResult().GetQueueUrl();
                std::cout << "Your new SQS queue with the name '" << queueName
                          << "' and the queue URL " << std::endl;
                std::cout << "'" << queueURL << "' has been created." <<
std::endl;
            }
            else {
                std::cerr << "Error with SQS::CreateQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
        queueURLS.push_back(queueURL);

        if (first) // Only explain this once.
        {
            std::cout
                    << "The queue URL is used to retrieve the queue ARN, which is
"
```

```
                << "used to create a subscription." << std::endl;
    }

    Aws::String queueARN;
    {
        // 3.  Get the SQS queue ARN attribute.
        Aws::SQS::Model::GetQueueAttributesRequest request;
        request.SetQueueUrl(queueURL);

request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

        Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                sqsClient.GetQueueAttributes(request);

        if (outcome.IsSuccess()) {
            const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                    outcome.GetResult().GetAttributes();
            const auto &iter = attributes.find(
                    Aws::SQS::Model::QueueAttributeName::QueueArn);
            if (iter != attributes.end()) {
                queueARN = iter->second;
                std::cout << "The queue ARN '" << queueARN
                          << "' has been retrieved."
                          << std::endl;
            }
            else {
                std::cerr
                        << "Error ARN attribute not returned by
GetQueueAttribute."
                        << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
        else {
            std::cerr << "Error with SQS::GetQueueAttributes. "
                      << outcome.GetError().GetMessage()
```

```
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        if (first) {
            std::cout
                    << "An IAM policy must be attached to an SQS queue, enabling
 it to receive "
                       "messages from an SNS topic." << std::endl;
        }

        {
            // 4.  Set the SQS queue policy attribute with a policy enabling the
 receipt of SNS messages.
            Aws::SQS::Model::SetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);
            Aws::String policy = createPolicyForQueue(queueARN, topicARN);
            request.AddAttributes(Aws::SQS::Model::QueueAttributeName::Policy,
                                  policy);

            Aws::SQS::Model::SetQueueAttributesOutcome outcome =
                    sqsClient.SetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                std::cout << "The attributes for the queue '" << queueName
                          << "' were successfully updated." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::SetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                         sqsClient);

                return false;
            }
        }

        printAsterisksLine();

        {
            // 5.  Subscribe the SQS queue to the SNS topic.
            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
                              << "\"'s subscription to the topic \""
                              << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
                if (askYesNoQuestion(ostringstream.str())) {
                    Aws::String jsonPolicy = getFilterPolicyFromUser();
                    if (!jsonPolicy.empty()) {
                        filteringMessages = true;
                        std::cout << "This is the filter policy for this
subscription."
                                  << std::endl;
                        std::cout << jsonPolicy << std::endl;

                        request.AddAttributes("FilterPolicy", jsonPolicy);
                    }
                    else {
                        std::cout
                                << "Because you did not select any attributes, no
filter "
                                << "will be added to this subscription." <<
std::endl;
                    }
                }
            }  // if (isFifoTopic)
            Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
        first = false;
    }

    first = true;
    do {
        printAsterisksLine();

        // 6.  Publish a message to the SNS topic.
        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);
        if (isFifoTopic) {
            if (first) {
                std::cout
                        << "Because you are using a FIFO topic, you must set a
message group ID."
                        << std::endl;
                std::cout
                        << "All messages within the same group will be received
in the "
                        << "order they were published." << std::endl;
            }
            Aws::String messageGroupID = askQuestion(
                    "Enter a message group ID for this message. ");
            request.SetMessageGroupId(messageGroupID);
            if (!contentBasedDeduplication) {
                if (first) {
                    std::cout
                            << "Because you are not using content-based
deduplication, "
                            << "you must enter a deduplication ID." << std::endl;
                }
                Aws::String deduplicationID = askQuestion(
                        "Enter a deduplication ID for this message. ");
                request.SetMessageDeduplicationId(deduplicationID);
            }
        }

        if (filteringMessages && askYesNoQuestion(
                "Add an attribute to this message? (y/n) ")) {
            for (size_t i = 0; i < TONES.size(); ++i) {
                std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
            }
            int selection = askQuestionForIntRange(
                    "Enter a number for an attribute. ",
                    1, static_cast<int>(TONES.size()));
            Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
            messageAttributeValue.SetDataType("String");
            messageAttributeValue.SetStringValue(TONES[selection - 1]);
            request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
        }

        Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

        if (outcome.IsSuccess()) {
            std::cout << "Your message was successfully published." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Publish. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }

        first = false;
    } while (askYesNoQuestion("Post another message? (y/n) "));

    printAsterisksLine();

    std::cout << "Now the SQS queue will be polled to retrieve the messages."
              << std::endl;
    askQuestion("Press any key to continue...", alwaysTrueTest);

    for (size_t i = 0; i < queueURLS.size(); ++i) {
        // 7.  Poll an SQS queue for its messages.
        std::vector<Aws::String> messages;
        std::vector<Aws::String> receiptHandles;
        while (true) {
            Aws::SQS::Model::ReceiveMessageRequest request;
            request.SetMaxNumberOfMessages(10);
            request.SetQueueUrl(queueURLS[i]);

            // Setting WaitTimeSeconds to non-zero enables long polling.
            // For information about long polling, see
            // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
            request.SetWaitTimeSeconds(1);
            Aws::SQS::Model::ReceiveMessageOutcome outcome =
                    sqsClient.ReceiveMessage(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SQS::Model::Message> &newMessages =
 outcome.GetResult().GetMessages();
                if (newMessages.empty()) {
                    break;
                }
                else {
                    for (const Aws::SQS::Model::Message &message: newMessages) {
                        messages.push_back(message.GetBody());
                        receiptHandles.push_back(message.GetReceiptHandle());
                    }
                }
            }
            else {
                std::cerr << "Error with SQS::ReceiveMessage. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        printAsterisksLine();

        if (messages.empty()) {
            std::cout << "No messages were ";
        }
        else if (messages.size() == 1) {
            std::cout << "One message was ";
        }
        else {
            std::cout << messages.size() << " messages were ";
        }
        std::cout << "received by the queue '" << queueNames[i]
                  << "'." << std::endl;
        for (const Aws::String &message: messages) {
            std::cout << "  Message : '" << message << "'."
                      << std::endl;
        }

        // 8.  Delete a batch of messages from an SQS queue.
        if (!receiptHandles.empty()) {
            Aws::SQS::Model::DeleteMessageBatchRequest request;
            request.SetQueueUrl(queueURLS[i]);
            int id = 1; // Ids must be unique within a batch delete request.
            for (const Aws::String &receiptHandle: receiptHandles) {
                Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
                entry.SetId(std::to_string(id));
                ++id;
                entry.SetReceiptHandle(receiptHandle);
                request.AddEntries(entry);
            }

            Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                    sqsClient.DeleteMessageBatch(request);

            if (outcome.IsSuccess()) {
                std::cout << "The batch deletion of messages was successful."
                          << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteMessageBatch. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
    }

    return cleanUp(topicARN,
                   queueURLS,
                   subscriptionARNS,
                   snsClient,
                   sqsClient,
                   true); // askUser
}


bool AwsDoc::TopicsAndQueues::cleanUp(const Aws::String &topicARN,
                                      const Aws::Vector<Aws::String> &queueURLS,
                                      const Aws::Vector<Aws::String>
 &subscriptionARNS,
                                      const Aws::SNS::SNSClient &snsClient,
                                      const Aws::SQS::SQSClient &sqsClient,
                                      bool askUser) {
    bool result = true;
    printAsterisksLine();
    if (!queueURLS.empty() && askUser &&
        askYesNoQuestion("Delete the SQS queues? (y/n) ")) {

        for (const auto &queueURL: queueURLS) {
            // 9.  Delete an SQS queue.
            Aws::SQS::Model::DeleteQueueRequest request;
            request.SetQueueUrl(queueURL);

            Aws::SQS::Model::DeleteQueueOutcome outcome =
                    sqsClient.DeleteQueue(request);

            if (outcome.IsSuccess()) {
                std::cout << "The queue with URL '" << queueURL
                          << "' was successfully deleted." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
```

```
       for (const auto &subscriptionARN: subscriptionARNS) {
           // 10. Unsubscribe an SNS subscription.
           Aws::SNS::Model::UnsubscribeRequest request;
           request.SetSubscriptionArn(subscriptionARN);

           Aws::SNS::Model::UnsubscribeOutcome outcome =
                   snsClient.Unsubscribe(request);

           if (outcome.IsSuccess()) {
               std::cout << "Unsubscribe of subscription ARN '" <<
subscriptionARN
                         << "' was successful." << std::endl;
           }
           else {
               std::cerr << "Error with TopicsAndQueues::Unsubscribe. "
                         << outcome.GetError().GetMessage()
                         << std::endl;
               result = false;
           }
       }
   }

   printAsterisksLine();
   if (!topicARN.empty() && askUser &&
       askYesNoQuestion("Delete the SNS topic? (y/n) ")) {

       // 11. Delete an SNS topic.
       Aws::SNS::Model::DeleteTopicRequest request;
       request.SetTopicArn(topicARN);

       Aws::SNS::Model::DeleteTopicOutcome outcome =
snsClient.DeleteTopic(request);

       if (outcome.IsSuccess()) {
           std::cout << "The topic with ARN '" << topicARN
                     << "' was successfully deleted." << std::endl;
       }
       else {
           std::cerr << "Error with TopicsAndQueues::DeleteTopicRequest. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           result = false;
       }
```

```
    }

    return result;
}

//! Create an IAM policy that gives an SQS queue permission to receive messages
 from an SNS topic.
/*!
 \sa createPolicyForQueue()
 \param queueARN: The SQS queue Amazon Resource Name (ARN).
 \param topicARN: The SNS topic ARN.
 \return Aws::String: The policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::createPolicyForQueue(const Aws::String
 &queueARN,
                                                          const Aws::String
 &topicARN) {
    std::ostringstream policyStream;
    policyStream << R"({
        "Statement": [
        {
            "Effect": "Allow",
                    "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "sqs:SendMessage",
                    "Resource": ")" << queueARN << R"(",
                    "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": ")" << topicARN << R"("
                }
            }
        }
        ]
    })";

    return policyStream.str();
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - CreateQueue
  - CreateTopic

- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)
```

```
const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
 so that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
 bool, bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
 standard.\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
   "Deduplication IDs are either set in the message or are automatically
 generated\n" +
   "from content using a hash function. If a message is successfully published to
\n" +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
 \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
 queue. ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
```

```
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string,
 topicArn string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n"
 +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
    "from the %v topic? (y/n) ", queueName, topicName), "y")
  if wantFiltering {
   log.Println("You can filter messages by one or more of the following \"tone\"
 attributes.")

   var toneSelections []string
   askAboutTones := true
   for askAboutTones {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelections = append(toneSelections, ToneChoices[toneIndex])
    askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
 the filter? (y/n) ", "y")
   }
   log.Printf("Your subscription will be filtered to only pass the following
 tones: %v\n", toneSelections)
   filterPolicy = map[string][]string{TONE_KEY: toneSelections}
  }
 }

 subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
 filterPolicy)
 if err != nil {
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
 string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
   log.Println("Because you are using a FIFO topic, you must set a message group
 ID.\n" +
    "All messages within the same group will be received in the order they were
 published.")
   groupId = runner.questioner.Ask("Enter a message group ID: ")
   if !contentBasedDeduplication {
    log.Println("Because you are not using content-based deduplication,\n" +
     "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
   }
  }
  if usingFilters {
   if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelection = ToneChoices[toneIndex]
   }
  }

  err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
 TONE_KEY, toneSelection)
  if err != nil {
   panic(err)
  }
  log.Println(("Your message was published."))

  publishMore = runner.questioner.AskBool("Do you want to publish another
 messsage? (y/n) ", "y")
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
   }
   messages = append(messages, currentMsgs...)
  }
  if len(messages) == 0 {
   log.Printf("No messages were received by queue %v.\n", queueUrl)
  } else if len(messages) == 1 {
   log.Printf("One message was received by queue %v:\n", queueUrl)
  } else {
   log.Printf("%v messages were received by queue %v:\n", len(messages),
 queueUrl)
  }
  for msgIndex, message := range messages {
   messageBody := MessageBody{}
   err := json.Unmarshal([]byte(*message.Body), &messageBody)
   if err != nil {
    panic(err)
   }
   log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
  }

  if len(messages) > 0 {
   log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
   err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
   if err != nil {
    panic(err)
   }
  }
 }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to
 the\n"+
  "topic. You can select from several options for configuring the topic and the
\n"+
  "subscriptions for the queues. You can then post to the topic and see the
 results\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.
\n", queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
 for _, ordinal := range ordinals {
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl,
 topicName, topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }
 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources
 created for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps Amazon SNS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}
```

```
// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

```
// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
```

```
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

Define a struct that wraps Amazon SQS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)
```

```
// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can
 specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
 (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
 to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
 to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 } else {
  messages = result.Messages
 }
 return messages, err
}
```

```
// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
 messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

Clean up resources.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management
 Console\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
  if err != nil {
   panic(err)
  }
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package com.example.sns;

import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.MessageAttributeValue;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import
 software.amazon.awssdk.services.sns.model.SetSubscriptionAttributesRequest;
```

```
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonPrimitive;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Gives the user three options to choose from.
 * 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 * 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 * 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 * 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 * 6. Subscribes to the SQS queue.
 * 7. Publishes a message to the topic.
 * 8. Displays the messages.
 * 9. Deletes the received message.
 * 10. Unsubscribes from the topic.
 * 11. Deletes the SNS topic.
 */
public class SNSWorkflow {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:\n" +
            "    <fifoQueueARN>\n\n" +
            "Where:\n" +
            "    accountId - Your AWS account Id value.";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        SqsClient sqsClient = SqsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        Scanner in = new Scanner(System.in);
        String accountId = args[0];
        String useFIFO;
        String duplication = "n";
        String topicName;
        String deduplicationID = null;
        String groupId = null;
        String topicArn;
        String sqsQueueName;
        String sqsQueueUrl;
        String sqsQueueArn;
        String subscriptionArn;
        boolean selectFIFO = false;

        String message;
        List<Message> messageList;
        List<String> filterList = new ArrayList<>();
        String msgAttValue = "";

        System.out.println(DASHES);
        System.out.println("Welcome to messaging with topics and queues.");
        System.out.println("In this scenario, you will create an SNS topic and
 subscribe an SQS queue to the topic.\n" +
            "You can select from several options for configuring the topic and
 the subscriptions for the queue.\n" +
            "You can then post to the topic and see the results in the queue.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("SNS topics can be configured as FIFO (First-In-First-
Out).\n" +
            "FIFO topics deliver messages in order and support deduplication and
 message filtering.\n" +
            "Would you like to work with FIFO topics? (y/n)");
        useFIFO = in.nextLine();
        if (useFIFO.compareTo("y") == 0) {
            selectFIFO = true;
            System.out.println("You have selected FIFO");
            System.out.println(" Because you have chosen a FIFO topic,
 deduplication is supported.\n" +
                "        Deduplication IDs are either set in the message or
 automatically generated from content using a hash function.\n"
                +
                "        If a message is successfully published to an SNS FIFO
 topic, any message published and determined to have the same deduplication ID,
\n"
                +
                "        within the five-minute deduplication interval, is
 accepted but not delivered.\n" +
                "        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.");
```

```
            System.out.println(
                "Would you like to use content-based deduplication instead of 
entering a deduplication ID? (y/n)");
            duplication = in.nextLine();
            if (duplication.compareTo("y") == 0) {
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            } else {
                System.out.println("Please enter deduplication Id value");
                deduplicationID = in.nextLine();
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            }
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a topic.");
        System.out.println("Enter a name for your SNS topic.");
        topicName = in.nextLine();
        if (selectFIFO) {
            System.out.println("Because you have selected a FIFO topic, '.fifo' 
must be appended to the topic name.");
            topicName = topicName + ".fifo";
            System.out.println("The name of the topic is " + topicName);
            topicArn = createFIFO(snsClient, topicName, duplication);
            System.out.println("The ARN of the FIFO topic is " + topicArn);

        } else {
            System.out.println("The name of the topic is " + topicName);
            topicArn = createSNSTopic(snsClient, topicName);
            System.out.println("The ARN of the non-FIFO topic is " + topicArn);

        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Create an SQS queue.");
        System.out.println("Enter a name for your SQS queue.");
        sqsQueueName = in.nextLine();
        if (selectFIFO) {
            sqsQueueName = sqsQueueName + ".fifo";
        }
```

```
       sqsQueueUrl = createQueue(sqsClient, sqsQueueName, selectFIFO);
       System.out.println("The queue URL is " + sqsQueueUrl);
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("4. Get the SQS queue ARN attribute.");
       sqsQueueArn = getSQSQueueAttrs(sqsClient, sqsQueueUrl);
       System.out.println("The ARN of the new queue is " + sqsQueueArn);
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("5. Attach an IAM policy to the queue.");

       // Define the policy to use. Make sure that you change the REGION if you
are
       // running this code
       // in a different region.
       String policy = """
       {
            "Statement": [
            {
                "Effect": "Allow",
                        "Principal": {
                    "Service": "sns.amazonaws.com"
                },
                "Action": "sqs:SendMessage",
                        "Resource": "arn:aws:sqs:us-east-1:%s:%s",
                        "Condition": {
                    "ArnEquals": {
                        "aws:SourceArn": "arn:aws:sns:us-east-1:%s:%s"
                    }
                }
            }
            ]
        }
       """.formatted(accountId, sqsQueueName, accountId, topicName);

       setQueueAttr(sqsClient, sqsQueueUrl, policy);
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("6. Subscribe to the SQS queue.");
       if (selectFIFO) {
           System.out.println(
                "If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.\n"
                    +
                    "For information about message filtering, see https://
docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n"
                    +
                    "For this example, you can filter messages by a \"tone\"
 attribute.");
            System.out.println("Would you like to filter messages for " +
 sqsQueueName + "'s subscription to the topic "
                + topicName + "?  (y/n)");
            String filterAns = in.nextLine();
            if (filterAns.compareTo("y") == 0) {
                boolean moreAns = false;
                System.out.println("You can filter messages by one or more of the
 following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                while (!moreAns) {
                    System.out.println("Select a number or choose 0 to end.");
                    String ans = in.nextLine();
                    switch (ans) {
                        case "1":
                            filterList.add("cheerful");
                            break;
                        case "2":
                            filterList.add("funny");
                            break;
                        case "3":
                            filterList.add("serious");
                            break;
                        case "4":
                            filterList.add("sincere");
                            break;
                        default:
                            moreAns = true;
                            break;
                    }
                }
            }
        }
        subscriptionArn = subQueue(snsClient, topicArn, sqsQueueArn, filterList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Publish a message to the topic.");
        if (selectFIFO) {
            System.out.println("Would you like to add an attribute to this
 message?  (y/n)");
            String msgAns = in.nextLine();
            if (msgAns.compareTo("y") == 0) {
                System.out.println("You can filter messages by one or more of the
 following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                System.out.println("Select a number or choose 0 to end.");
                String ans = in.nextLine();
                switch (ans) {
                    case "1":
                        msgAttValue = "cheerful";
                        break;
                    case "2":
                        msgAttValue = "funny";
                        break;
                    case "3":
                        msgAttValue = "serious";
                        break;
                    default:
                        msgAttValue = "sincere";
                        break;
                }

                System.out.println("Selected value is " + msgAttValue);
            }
            System.out.println("Enter a message.");
            message = in.nextLine();
            pubMessageFIFO(snsClient, message, topicArn, msgAttValue,
 duplication, groupId, deduplicationID);

        } else {
            System.out.println("Enter a message.");
            message = in.nextLine();
            pubMessage(snsClient, message, topicArn);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Display the message. Press any key to continue.");
        in.nextLine();
        messageList = receiveMessages(sqsClient, sqsQueueUrl, msgAttValue);
        for (Message mes : messageList) {
            System.out.println("Message Id: " + mes.messageId());
            System.out.println("Full Message: " + mes.body());
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Delete the received message. Press any key to
 continue.");
        in.nextLine();
        deleteMessages(sqsClient, sqsQueueUrl, messageList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Unsubscribe from the topic and delete the queue.
 Press any key to continue.");
        in.nextLine();
        unSub(snsClient, subscriptionArn);
        deleteSQSQueue(sqsClient, sqsQueueName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Delete the topic. Press any key to continue.");
        in.nextLine();
        deleteSNSTopic(snsClient, topicArn);

        System.out.println(DASHES);
        System.out.println("The SNS/SQS workflow has completed successfully.");
        System.out.println(DASHES);
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                .topicArn(topicArn)
                .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
```

```
           System.out.println("Status was " +
result.sdkHttpResponse().statusCode());

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
       try {
           GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
               .queueName(queueName)
               .build();

           String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
           DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
               .queueUrl(queueUrl)
               .build();

           sqsClient.deleteQueue(deleteQueueRequest);
           System.out.println(queueName + " was successfully deleted.");

       } catch (SqsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void unSub(SnsClient snsClient, String subscriptionArn) {
       try {
           UnsubscribeRequest request = UnsubscribeRequest.builder()
               .subscriptionArn(subscriptionArn)
               .build();

           UnsubscribeResponse result = snsClient.unsubscribe(request);
           System.out.println("Status was " +
result.sdkHttpResponse().statusCode()
               + "\nSubscription was removed for " + request.subscriptionArn());

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
       try {
           List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();
           for (Message msg : messages) {
               DeleteMessageBatchRequestEntry entry =
DeleteMessageBatchRequestEntry.builder()
                   .id(msg.messageId())
                   .build();

               entries.add(entry);
           }

           DeleteMessageBatchRequest deleteMessageBatchRequest =
DeleteMessageBatchRequest.builder()
               .queueUrl(queueUrl)
               .entries(entries)
               .build();

           sqsClient.deleteMessageBatch(deleteMessageBatchRequest);
           System.out.println("The batch delete of messages was successful");

       } catch (SqsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl, String msgAttValue) {
       try {
           if (msgAttValue.isEmpty()) {
               ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                   .queueUrl(queueUrl)
                   .maxNumberOfMessages(5)
                   .build();
               return
sqsClient.receiveMessage(receiveMessageRequest).messages();
           } else {
               // We know there are filters on the message.
               ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                   .queueUrl(queueUrl)
                   .messageAttributeNames(msgAttValue) // Include other message
attributes if needed.
                   .maxNumberOfMessages(5)
                   .build();

               return sqsClient.receiveMessage(receiveRequest).messages();
           }

       } catch (SqsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return null;
   }

   public static void pubMessage(SnsClient snsClient, String message, String
topicArn) {
       try {
           PublishRequest request = PublishRequest.builder()
               .message(message)
               .topicArn(topicArn)
               .build();

           PublishResponse result = snsClient.publish(request);
           System.out
               .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void pubMessageFIFO(SnsClient snsClient,
                                     String message,
                                     String topicArn,
                                     String msgAttValue,
                                     String duplication,
                                     String groupId,
                                     String deduplicationID) {
```

```
        try {
            PublishRequest request;
            // Means the user did not choose to use a message attribute.
            if (msgAttValue.isEmpty()) {
                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                } else {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                }

            } else {
                Map<String, MessageAttributeValue> messageAttributes = new 
HashMap<>();
                messageAttributes.put(msgAttValue, 
MessageAttributeValue.builder()
                    .dataType("String")
                    .stringValue("true")
                    .build());

                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                } else {
                    // Create a publish request with the message and attributes.
                    request = PublishRequest.builder()
                        .topicArn(topicArn)
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .messageAttributes(messageAttributes)
                        .build();
```

```
               }
           }

           // Publish the message to the topic.
           PublishResponse result = snsClient.publish(request);
           System.out
               .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   // Subscribe to the SQS queue.
   public static String subQueue(SnsClient snsClient, String topicArn, String
queueArn, List<String> filterList) {
       try {
           SubscribeRequest request;
           if (filterList.isEmpty()) {
               // No filter subscription is added.
               request = SubscribeRequest.builder()
                   .protocol("sqs")
                   .endpoint(queueArn)
                   .returnSubscriptionArn(true)
                   .topicArn(topicArn)
                   .build();

               SubscribeResponse result = snsClient.subscribe(request);
               System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                   "with the subscription ARN " + result.subscriptionArn());
               return result.subscriptionArn();
           } else {
               request = SubscribeRequest.builder()
                   .protocol("sqs")
                   .endpoint(queueArn)
                   .returnSubscriptionArn(true)
                   .topicArn(topicArn)
                   .build();

               SubscribeResponse result = snsClient.subscribe(request);
               System.out.println("The queue " + queueArn + " has been 
subscribed to the topic " + topicArn + "\n" +
                   "with the subscription ARN " + result.subscriptionArn());

               String attributeName = "FilterPolicy";
               Gson gson = new Gson();
               String jsonString = "{\"tone\": []}";
               JsonObject jsonObject = gson.fromJson(jsonString, 
JsonObject.class);
               JsonArray toneArray = jsonObject.getAsJsonArray("tone");
               for (String value : filterList) {
                   toneArray.add(new JsonPrimitive(value));
               }

               String updatedJsonString = gson.toJson(jsonObject);
               System.out.println(updatedJsonString);
               SetSubscriptionAttributesRequest attRequest = 
SetSubscriptionAttributesRequest.builder()
                   .subscriptionArn(result.subscriptionArn())
                   .attributeName(attributeName)
                   .attributeValue(updatedJsonString)
                   .build();

               snsClient.setSubscriptionAttributes(attRequest);
               return result.subscriptionArn();
           }

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return "";
   }

   // Attach a policy to the queue.
   public static void setQueueAttr(SqsClient sqsClient, String queueUrl, String 
policy) {
       try {
           Map<software.amazon.awssdk.services.sqs.model.QueueAttributeName, 
String> attrMap = new HashMap<>();
           attrMap.put(QueueAttributeName.POLICY, policy);

           SetQueueAttributesRequest attributesRequest = 
SetQueueAttributesRequest.builder()
```

```
               .queueUrl(queueUrl)
               .attributes(attrMap)
               .build();

           sqsClient.setQueueAttributes(attributesRequest);
           System.out.println("The policy has been successfully attached.");

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static String getSQSQueueAttrs(SqsClient sqsClient, String queueUrl) {
       // Specify the attributes to retrieve.
       List<QueueAttributeName> atts = new ArrayList<>();
       atts.add(QueueAttributeName.QUEUE_ARN);

       GetQueueAttributesRequest attributesRequest = 
GetQueueAttributesRequest.builder()
           .queueUrl(queueUrl)
           .attributeNames(atts)
           .build();

       GetQueueAttributesResponse response = 
sqsClient.getQueueAttributes(attributesRequest);
       Map<String, String> queueAtts = response.attributesAsStrings();
       for (Map.Entry<String, String> queueAtt : queueAtts.entrySet())
           return queueAtt.getValue();

       return "";
   }

   public static String createQueue(SqsClient sqsClient, String queueName, 
Boolean selectFIFO) {
       try {
           System.out.println("\nCreate Queue");
           if (selectFIFO) {
               Map<QueueAttributeName, String> attrs = new HashMap<>();
               attrs.put(QueueAttributeName.FIFO_QUEUE, "true");
               CreateQueueRequest createQueueRequest = 
CreateQueueRequest.builder()
                   .queueName(queueName)
                   .attributes(attrs)
                   .build();

               sqsClient.createQueue(createQueueRequest);
               System.out.println("\nGet queue url");
               GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
               return getQueueUrlResponse.queueUrl();
           } else {
               CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                   .queueName(queueName)
                   .build();

               sqsClient.createQueue(createQueueRequest);
               System.out.println("\nGet queue url");
               GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
               return getQueueUrlResponse.queueUrl();
           }

       } catch (SqsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return "";
   }

   public static String createSNSTopic(SnsClient snsClient, String topicName) {
       CreateTopicResponse result;
       try {
           CreateTopicRequest request = CreateTopicRequest.builder()
               .name(topicName)
               .build();

           result = snsClient.createTopic(request);
           return result.topicArn();

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return "";
```

```
    }

    public static String createFIFO(SnsClient snsClient, String topicName, String
 duplication) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = new HashMap<>();
            if (duplication.compareTo("n") == 0) {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "false");
            } else {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "true");
            }

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            return response.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish

- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This is the entry point for this scenario.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

The preceding code provides the necessary dependencies and starts the scenario. The next section contains the bulk of the example.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }
  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }
  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package com.example.sns

import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.CreateTopicRequest
import aws.sdk.kotlin.services.sns.model.DeleteTopicRequest
import aws.sdk.kotlin.services.sns.model.PublishRequest
import aws.sdk.kotlin.services.sns.model.SetSubscriptionAttributesRequest
import aws.sdk.kotlin.services.sns.model.SubscribeRequest
import aws.sdk.kotlin.services.sns.model.UnsubscribeRequest
import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.model.CreateQueueRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequestEntry
import aws.sdk.kotlin.services.sqs.model.DeleteQueueRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueAttributesRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueUrlRequest
import aws.sdk.kotlin.services.sqs.model.Message
import aws.sdk.kotlin.services.sqs.model.QueueAttributeName
import aws.sdk.kotlin.services.sqs.model.ReceiveMessageRequest
import aws.sdk.kotlin.services.sqs.model.SetQueueAttributesRequest
import com.google.gson.Gson
import com.google.gson.JsonObject
import com.google.gson.JsonPrimitive
import java.util.Scanner

/**
Before running this Kotlin code example, set up your development environment,
including your AWS credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This Kotlin example performs the following tasks:

 1. Gives the user three options to choose from.
 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 6. Subscribes to the SQS queue.
 7. Publishes a message to the topic.
 8. Displays the messages.
 9. Deletes the received message.
 10. Unsubscribes from the topic.
 11. Deletes the SNS topic.
 */

val DASHES: String = String(CharArray(80)).replace("\u0000", "-")
suspend fun main() {
    val input = Scanner(System.`in`)
    val useFIFO: String
    var duplication = "n"
```

```
   var topicName: String
   var deduplicationID: String? = null
   var groupId: String? = null
   val topicArn: String?
   var sqsQueueName: String
   val sqsQueueUrl: String?
   val sqsQueueArn: String
   val subscriptionArn: String?
   var selectFIFO = false
   val message: String
   val messageList: List<Message?>?
   val filterList = ArrayList<String>()
   var msgAttValue = ""

   println(DASHES)
   println("Welcome to the AWS SDK for Kotlin messaging with topics and 
queues.")
   println(
       """
               In this scenario, you will create an SNS topic and subscribe an 
SQS queue to the topic.
               You can select from several options for configuring the topic and 
the subscriptions for the queue.
               You can then post to the topic and see the results in the queue.
       """.trimIndent(),
   )
   println(DASHES)

   println(DASHES)
   println(
       """
               SNS topics can be configured as FIFO (First-In-First-Out).
               FIFO topics deliver messages in order and support deduplication 
and message filtering.
               Would you like to work with FIFO topics? (y/n)
       """.trimIndent(),
   )
   useFIFO = input.nextLine()
   if (useFIFO.compareTo("y") == 0) {
       selectFIFO = true
       println("You have selected FIFO")
       println(
           """ Because you have chosen a FIFO topic, deduplication is supported.
```

```
        Deduplication IDs are either set in the message or automatically
 generated from content using a hash function.
        If a message is successfully published to an SNS FIFO topic, any message
 published and determined to have the same deduplication ID,
        within the five-minute deduplication interval, is accepted but not
 delivered.
        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.""",
        )

        println("Would you like to use content-based deduplication instead of
 entering a deduplication ID? (y/n)")
        duplication = input.nextLine()
        if (duplication.compareTo("y") == 0) {
            println("Enter a group id value")
            groupId = input.nextLine()
        } else {
            println("Enter deduplication Id value")
            deduplicationID = input.nextLine()
            println("Enter a group id value")
            groupId = input.nextLine()
        }
    }
    println(DASHES)

    println(DASHES)
    println("2. Create a topic.")
    println("Enter a name for your SNS topic.")
    topicName = input.nextLine()
    if (selectFIFO) {
        println("Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.")
        topicName = "$topicName.fifo"
        println("The name of the topic is $topicName")
        topicArn = createFIFO(topicName, duplication)
        println("The ARN of the FIFO topic is $topicArn")
    } else {
        println("The name of the topic is $topicName")
        topicArn = createSNSTopic(topicName)
        println("The ARN of the non-FIFO topic is $topicArn")
    }
    println(DASHES)

    println(DASHES)
    println("3. Create an SQS queue.")
    println("Enter a name for your SQS queue.")
    sqsQueueName = input.nextLine()
    if (selectFIFO) {
        sqsQueueName = "$sqsQueueName.fifo"
    }
    sqsQueueUrl = createQueue(sqsQueueName, selectFIFO)
    println("The queue URL is $sqsQueueUrl")
    println(DASHES)

    println(DASHES)
    println("4. Get the SQS queue ARN attribute.")
    sqsQueueArn = getSQSQueueAttrs(sqsQueueUrl)
    println("The ARN of the new queue is $sqsQueueArn")
    println(DASHES)

    println(DASHES)
    println("5. Attach an IAM policy to the queue.")
    // Define the policy to use.
    val policy = """{
     "Statement": [
     {
         "Effect": "Allow",
                 "Principal": {
             "Service": "sns.amazonaws.com"
         },
         "Action": "sqs:SendMessage",
                 "Resource": "$sqsQueueArn",
                 "Condition": {
             "ArnEquals": {
                 "aws:SourceArn": "$topicArn"
             }
         }
     }
     ]
     }"""
    setQueueAttr(sqsQueueUrl, policy)
    println(DASHES)

    println(DASHES)
    println("6. Subscribe to the SQS queue.")
    if (selectFIFO) {
        println(
            """If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.
For information about message filtering, see https://docs.aws.amazon.com/sns/
latest/dg/sns-message-filtering.html
For this example, you can filter messages by a "tone" attribute.""",
        )
        println("Would you like to filter messages for $sqsQueueName's
 subscription to the topic $topicName?  (y/n)")
        val filterAns: String = input.nextLine()
        if (filterAns.compareTo("y") == 0) {
            var moreAns = false
            println("You can filter messages by using one or more of the
 following \"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
            println("4. sincere")
            while (!moreAns) {
                println("Select a number or choose 0 to end.")
                val ans: String = input.nextLine()
                when (ans) {
                    "1" -> filterList.add("cheerful")
                    "2" -> filterList.add("funny")
                    "3" -> filterList.add("serious")
                    "4" -> filterList.add("sincere")
                    else -> moreAns = true
                }
            }
        }
    }
    subscriptionArn = subQueue(topicArn, sqsQueueArn, filterList)
    println(DASHES)

    println(DASHES)
    println("7. Publish a message to the topic.")
    if (selectFIFO) {
        println("Would you like to add an attribute to this message?  (y/n)")
        val msgAns: String = input.nextLine()
        if (msgAns.compareTo("y") == 0) {
            println("You can filter messages by one or more of the following
 \"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
```

```
           println("4. sincere")
           println("Select a number or choose 0 to end.")
           val ans: String = input.nextLine()
           msgAttValue = when (ans) {
               "1" -> "cheerful"
               "2" -> "funny"
               "3" -> "serious"
               else -> "sincere"
           }
           println("Selected value is $msgAttValue")
       }
       println("Enter a message.")
       message = input.nextLine()
       pubMessageFIFO(message, topicArn, msgAttValue, duplication, groupId,
deduplicationID)
   } else {
       println("Enter a message.")
       message = input.nextLine()
       pubMessage(message, topicArn)
   }
   println(DASHES)

   println(DASHES)
   println("8. Display the message. Press any key to continue.")
   input.nextLine()
   messageList = receiveMessages(sqsQueueUrl, msgAttValue)
   if (messageList != null) {
       for (mes in messageList) {
           println("Message Id: ${mes.messageId}")
           println("Full Message: ${mes.body}")
       }
   }
   println(DASHES)

   println(DASHES)
   println("9. Delete the received message. Press any key to continue.")
   input.nextLine()
   if (messageList != null) {
       deleteMessages(sqsQueueUrl, messageList)
   }
   println(DASHES)

   println(DASHES)
```

```
    println("10. Unsubscribe from the topic and delete the queue. Press any key
 to continue.")
    input.nextLine()
    unSub(subscriptionArn)
    deleteSQSQueue(sqsQueueName)
    println(DASHES)

    println(DASHES)
    println("11. Delete the topic. Press any key to continue.")
    input.nextLine()
    deleteSNSTopic(topicArn)
    println(DASHES)

    println(DASHES)
    println("The SNS/SQS workflow has completed successfully.")
    println(DASHES)
}

suspend fun deleteSNSTopic(topicArnVal: String?) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was deleted")
    }
}

suspend fun deleteSQSQueue(queueNameVal: String) {
    val getQueueRequest = GetQueueUrlRequest {
        queueName = queueNameVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val queueUrlVal = sqsClient.getQueueUrl(getQueueRequest).queueUrl
        val deleteQueueRequest = DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

        sqsClient.deleteQueue(deleteQueueRequest)
        println("$queueNameVal was successfully deleted.")
    }
}
```

```
suspend fun unSub(subscripArn: String?) {
    val request = UnsubscribeRequest {
        subscriptionArn = subscripArn
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for $subscripArn")
    }
}

suspend fun deleteMessages(queueUrlVal: String?, messages: List<Message>) {
    val entriesVal: MutableList<DeleteMessageBatchRequestEntry> = mutableListOf()
    for (msg in messages) {
        val entry = DeleteMessageBatchRequestEntry {
            id = msg.messageId
        }
        entriesVal.add(entry)
    }

    val deleteMessageBatchRequest = DeleteMessageBatchRequest {
        queueUrl = queueUrlVal
        entries = entriesVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteMessageBatch(deleteMessageBatchRequest)
        println("The batch delete of messages was successful")
    }
}

suspend fun receiveMessages(queueUrlVal: String?, msgAttValue: String):
 List<Message>? {
    if (msgAttValue.isEmpty()) {
        val request = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(request).messages
        }
    } else {
        val receiveRequest = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            waitTimeSeconds = 1
            maxNumberOfMessages = 5
        }
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(receiveRequest).messages
        }
    }
}

suspend fun pubMessage(messageVal: String?, topicArnVal: String?) {
    val request = PublishRequest {
        message = messageVal
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}

suspend fun pubMessageFIFO(
    messageVal: String?,
    topicArnVal: String?,
    msgAttValue: String,
    duplication: String,
    groupIdVal: String?,
    deduplicationID: String?,
) {
    // Means the user did not choose to use a message attribute.
    if (msgAttValue.isEmpty()) {
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            val request = PublishRequest {
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        }
    } else {
        val messAttr = aws.sdk.kotlin.services.sns.model.MessageAttributeValue {
            dataType = "String"
            stringValue = "true"
        }

        val mapAtt: Map<String,
 aws.sdk.kotlin.services.sns.model.MessageAttributeValue> =
            mapOf(msgAttValue to messAttr)
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            // Create a publish request with the message and attributes.
            val request = PublishRequest {
                topicArn = topicArnVal
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
                messageAttributes = mapAtt
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
```

```
            }
        }
    }
}

// Subscribe to the SQS queue.
suspend fun subQueue(topicArnVal: String?, queueArnVal: String, filterList:
 List<String?>): String? {
    val request: SubscribeRequest
    if (filterList.isEmpty()) {
        // No filter subscription is added.
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println(
                "The queue " + queueArnVal + " has been subscribed to the topic "
 + topicArnVal + "\n" +
                    "with the subscription ARN " + result.subscriptionArn,
            )
            return result.subscriptionArn
        }
    } else {
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println("The queue $queueArnVal has been subscribed to the topic
 $topicArnVal with the subscription ARN ${result.subscriptionArn}")

            val attributeNameVal = "FilterPolicy"
            val gson = Gson()
            val jsonString = "{\"tone\": []}"
            val jsonObject = gson.fromJson(jsonString, JsonObject::class.java)
            val toneArray = jsonObject.getAsJsonArray("tone")
            for (value: String? in filterList) {
                toneArray.add(JsonPrimitive(value))
            }

            val updatedJsonString: String = gson.toJson(jsonObject)
            println(updatedJsonString)
            val attRequest = SetSubscriptionAttributesRequest {
                subscriptionArn = result.subscriptionArn
                attributeName = attributeNameVal
                attributeValue = updatedJsonString
            }

            snsClient.setSubscriptionAttributes(attRequest)
            return result.subscriptionArn
        }
    }
}

suspend fun setQueueAttr(queueUrlVal: String?, policy: String) {
    val attrMap: MutableMap<String, String> = HashMap()
    attrMap[QueueAttributeName.Policy.toString()] = policy

    val attributesRequest = SetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributes = attrMap
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.setQueueAttributes(attributesRequest)
        println("The policy has been successfully attached.")
    }
}

suspend fun getSQSQueueAttrs(queueUrlVal: String?): String {
    val atts: MutableList<QueueAttributeName> = ArrayList()
    atts.add(QueueAttributeName.QueueArn)

    val attributesRequest = GetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributeNames = atts
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.getQueueAttributes(attributesRequest)
        val mapAtts = response.attributes
        if (mapAtts != null) {
            mapAtts.forEach { entry ->
                println("${entry.key} : ${entry.value}")
                return entry.value
            }
        }
    }
    return ""
}

suspend fun createQueue(queueNameVal: String?, selectFIFO: Boolean): String? {
    println("\nCreate Queue")
    if (selectFIFO) {
        val attrs = mutableMapOf<String, String>()
        attrs[QueueAttributeName.FifoQueue.toString()] = "true"

        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
            attributes = attrs
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("\nGet queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    } else {
        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("Get queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    }
}

suspend fun createSNSTopic(topicName: String?): String? {
    val request = CreateTopicRequest {
        name = topicName
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn
    }
}

suspend fun createFIFO(topicName: String?, duplication: String): String? {
    val topicAttributes: MutableMap<String, String> = HashMap()
    if (duplication.compareTo("n") == 0) {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "false"
    } else {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "true"
    }

    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        return response.topicArn
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - CreateQueue
  - CreateTopic

- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "queue-scenario",
        abstract: """
        This example interactively demonstrates how to use Amazon Simple
        Notification Service (Amazon SNS) and Amazon Simple Queue Service
        (Amazon SQS) together to publish and receive messages using queues.
        """,
        discussion: """
        Supports filtering using a "tone" attribute.
        """
    )

    /// Prompt for an input string. Only non-empty strings are allowed.
    ///
    /// - Parameter prompt: The prompt to display.
    ///
    /// - Returns: The string input by the user.
    func stringRequest(prompt: String) -> String {
        var str: String?

        while str == nil {
            print(prompt, terminator: "")
            str = readLine()

            if str != nil && str?.count == 0 {
                str = nil
            }
        }

        return str!
    }

    /// Ask a yes/no question.
    ///
    /// - Parameter prompt: A prompt string to print.
    ///
    /// - Returns: `true` if the user answered "Y", otherwise `false`.
    func yesNoRequest(prompt: String) -> Bool {
        while true {
            let answer = stringRequest(prompt: prompt).lowercased()
            if answer == "y" || answer == "n" {
                return answer == "y"
            }
        }
    }

    /// Display a menu of options then request a selection.
    ///
    /// - Parameters:
    ///   - prompt: A prompt string to display before the menu.
    ///   - options: An array of strings giving the menu options.
    ///
    /// - Returns: The index number of the selected option or 0 if no item was
    ///   selected.
    func menuRequest(prompt: String, options: [String]) -> Int {
        let numOptions = options.count

        if numOptions == 0 {
            return 0
        }

        print(prompt)

        for (index, value) in options.enumerated() {
            print("(\(index)) \(value)")
        }

        repeat {
            print("Enter your selection (0 - \(numOptions-1)): ", terminator: "")
            if let answer = readLine() {
                guard let answer = Int(answer) else {
                    print("Please enter the number matching your selection.")
                    continue
                }

                if answer >= 0 && answer < numOptions {
                    return answer
                } else {
                    print("Please enter the number matching your selection.")
                }
            }
        } while true
    }

    /// Ask the user too press RETURN. Accepts any input but ignores it.
    ///
    /// - Parameter prompt: The text prompt to display.
    func returnRequest(prompt: String) {
        print(prompt, terminator: "")
        _ = readLine()
    }

    var attrValues = [
        "<none>",
        "cheerful",
        "funny",
        "serious",
        "sincere"
    ]

    /// Ask the user to choose one of the attribute values to use as a filter.
    ///
    /// - Parameters:
    ///   - message: A message to display before the menu of values.
    ///   - attrValues: An array of strings giving the values to choose from.
    ///
    /// - Returns: The string corresponding to the selected option.
    func askForFilter(message: String, attrValues: [String]) -> String? {
        print(message)
        for (index, value) in attrValues.enumerated() {
            print("  [\(index)] \(value)")
        }

        var answer: Int?
        repeat {
            answer = Int(stringRequest(prompt: "Select an value for the 'tone' 
 attribute or 0 to end: "))
        } while answer == nil || answer! < 0 || answer! > attrValues.count + 1

        if answer == 0 {
            return nil
        }
        return attrValues[answer!]
    }

    /// Prompts the user for filter terms and constructs the attribute
    /// record that specifies them.
    ///
    /// - Returns: A mapping of "FilterPolicy" to a JSON string representing
    ///   the user-defined filter.
    func buildFilterAttributes() -> [String:String] {
        var attr: [String:String] = [:]
        var filterString = ""

        var first = true

        while let ans = askForFilter(message: "Choose a value to apply to the 
 'tone' attribute.",
                                    attrValues: attrValues) {
            if !first {
                filterString += ","
            }
            first = false

            filterString += "\"\(ans)\""
        }

        let filterJSON = "{ \"tone\": [\(filterString)]}"
        attr["FilterPolicy"] = filterJSON

        return attr
    }
    /// Create a queue, returning its URL string.
    ///
    /// - Parameters:
    ///   - prompt: A prompt to ask for the queue name.
    ///   - isFIFO: Whether or not to create a FIFO queue.
    ///
    /// - Returns: The URL of the queue.
    func createQueue(prompt: String, sqsClient: SQSClient, isFIFO: Bool) async
 throws -> String? {
        repeat {
            var queueName = stringRequest(prompt: prompt)
            var attributes: [String: String] = [:]

            if isFIFO {
                queueName += ".fifo"
                attributes["FifoQueue"] = "true"
            }

            do {
                let output = try await sqsClient.createQueue(
                    input: CreateQueueInput(
                        attributes: attributes,
                        queueName: queueName
                    )
                )
                guard let url = output.queueUrl else {
                    return nil
                }

                return url
            } catch _ as QueueDeletedRecently {
                print("You need to use a different queue name. A queue by that
name was recently deleted.")
                continue
            }
        } while true
    }

    /// Return the ARN of a queue given its URL.
    ///
    /// - Parameter queueUrl: The URL of the queue for which to return the
    ///   ARN.
    ///
    /// - Returns: The ARN of the specified queue.
    func getQueueARN(sqsClient: SQSClient, queueUrl: String) async throws ->
String? {
        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [.queuearn],
                queueUrl: queueUrl
            )
        )

        guard let attributes = output.attributes else {
            return nil
        }

        return attributes["QueueArn"]
    }

    /// Applies the needed policy to the specified queue.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The queue to apply the policy to.
    ///   - queueArn: The ARN of the queue to apply the policy to.
    ///   - topicArn: The topic that should have access via the policy.
    ///
    /// - Throws: Errors from the SQS `SetQueueAttributes` action.
    func setQueuePolicy(sqsClient: SQSClient, queueUrl: String,
                        queueArn: String, topicArn: String) async throws {
        _ = try await sqsClient.setQueueAttributes(
            input: SetQueueAttributesInput(
                attributes: [
                    "Policy":
                        """
                        {
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Principal": {
                                        "Service": "sns.amazonaws.com"
                                    },
                                    "Action": "sqs:SendMessage",
                                    "Resource": "\(queueArn)",
                                    "Condition": {
                                        "ArnEquals": {
                                            "aws:SourceArn": "\(topicArn)"
                                        }
                                    }
                                }
                            ]
                        }
                        """

                ],
                queueUrl: queueUrl
            )
        )
    }

    /// Receive the available messages on a queue, outputting them to the
    /// screen. Returns a dictionary you pass to DeleteMessageBatch to delete
    /// all the received messages.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The SQS queue on which to receive messages.
    ///
    /// - Throws: Errors from `SQSClient.receiveMessage()`
    ///
    /// - Returns: An array of SQSClientTypes.DeleteMessageBatchRequestEntry
    ///   items, each describing one received message in the format needed to
    ///   delete it.
    func receiveAndListMessages(sqsClient: SQSClient, queueUrl: String) async
 throws
                                ->
 [SQSClientTypes.DeleteMessageBatchRequestEntry] {
        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: 10,
                queueUrl: queueUrl
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return []
        }

        var deleteList: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []

        // Print out all the messages that were received, including their
        // attributes, if any.

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print("Message JSON:   \(message.body ?? "<body missing>")")

            if message.receiptHandle != nil {
                deleteList.append(
                    SQSClientTypes.DeleteMessageBatchRequestEntry(
                        id: message.messageId,
                        receiptHandle: message.receiptHandle
                    )
                )
            }
        }

        return deleteList
    }

    /// Delete all the messages in the specified list.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The SQS queue to delete messages from.
    ///   - deleteList: A list of `DeleteMessageBatchRequestEntry` objects
    ///     describing the messages to delete.
    ///
    /// - Throws: Errors from `SQSClient.deleteMessageBatch()`.
    func deleteMessageList(sqsClient: SQSClient, queueUrl: String,
                           deleteList:
 [SQSClientTypes.DeleteMessageBatchRequestEntry]) async throws {
        let output = try await sqsClient.deleteMessageBatch(
            input: DeleteMessageBatchInput(entries: deleteList, queueUrl:
 queueUrl)
        )

        if let failed = output.failed {
            print("\(failed.count) errors occurred deleting messages from the
 queue.")
            for message in failed {
                print("---> Failed to delete message \(message.id ?? "<unknown
 ID>") with error: \(message.code ?? "<unknown>") (\(message.message ?? "..."))")
            }
        }
    }

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let rowOfStars = String(repeating: "*", count: 75)

        print("""
              \(rowOfStars)
              Welcome to the cross-service messaging with topics and queues
 example.
              In this workflow, you'll create an SNS topic, then create two SQS
              queues which will be subscribed to that topic.

              You can specify several options for configuring the topic, as well
 as
              the queue subscriptions. You can then post messages to the topic
 and
              receive the results on the queues.
              \(rowOfStars)\n
              """
        )

        // 0. Create SNS and SQS clients.

        let snsConfig = try await SNSClient.SNSClientConfiguration(region:
 region)
        let snsClient = SNSClient(config: snsConfig)
```

```
        let sqsConfig = try await SQSClient.SQSClientConfiguration(region:
 region)
        let sqsClient = SQSClient(config: sqsConfig)

        // 1. Ask the user whether to create a FIFO topic. If so, ask whether
        //    to use content-based deduplication instead of requiring a
        //    deduplication ID.

        let isFIFO = yesNoRequest(prompt: "Do you want to create a FIFO topic (Y/
N)? ")
        var isContentBasedDeduplication = false

        if isFIFO {
            print("""
                  \(rowOfStars)
                  Because you've chosen to create a FIFO topic, deduplication is
                  supported.

                  Deduplication IDs are either set in the message or are
 automatically
                  generated from the content using a hash function.

                  If a message is successfully published to an SNS FIFO topic,
 any
                  message published and found to have the same deduplication ID
                  (within a five-minute deduplication interval), is accepted but
                  not delivered.

                  For more information about deduplication, see:
                  https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html.
                  """
            )

            isContentBasedDeduplication = yesNoRequest(
                prompt: "Use content-based deduplication instead of entering a
 deduplication ID (Y/N)? ")
            print(rowOfStars)
        }

        var topicName = stringRequest(prompt: "Enter the name of the topic to
 create: ")

        // 2. Create the topic. Append ".fifo" to the name if FIFO was
        //    requested, and set the "FifoTopic" attribute to "true" if so as
        //    well. Set the "ContentBasedDeduplication" attribute to "true" if
        //    content-based deduplication was requested.

        if isFIFO {
            topicName += ".fifo"
        }

        print("Topic name: \(topicName)")

        var attributes = [
            "FifoTopic": (isFIFO ? "true" : "false")
        ]

        // If it's a FIFO topic with content-based deduplication, set the
        // "ContentBasedDeduplication" attribute.

        if isContentBasedDeduplication {
            attributes["ContentBasedDeduplication"] = "true"
        }

        // Create the topic and retrieve the ARN.

        let output = try await snsClient.createTopic(
            input: CreateTopicInput(
                attributes: attributes,
                name: topicName
            )
        )

        guard let topicArn = output.topicArn else {
            print("No topic ARN returned!")
            return
        }

        print("""
              Topic '\(topicName) has been created with the
              topic ARN \(topicArn)."
              """
        )

        print(rowOfStars)

        // 3. Create an SQS queue. Append ".fifo" to the name if one of the
        //    FIFO topic configurations was chosen, and set "FifoQueue" to
        //    "true" if the topic is FIFO.

        print("""
              Next, you will create two SQS queues that will be subscribed
              to the topic you just created.\n
              """
        )

        let q1Url = try await createQueue(prompt: "Enter the name of the first 
queue: ",
                                          sqsClient: sqsClient, isFIFO: isFIFO)
        guard let q1Url else {
            print("Unable to create queue 1!")
            return
        }

        // 4. Get the SQS queue's ARN attribute using `GetQueueAttributes`.

        let q1Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q1Url)

        guard let q1Arn else {
            print("Unable to get ARN of queue 1!")
            return
        }
        print("Got queue 1 ARN: \(q1Arn)")

        // 5. Attach an AWS IAM policy to the queue using
        //    `SetQueueAttributes`.

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q1Url,
                                 queueArn: q1Arn, topicArn: topicArn)

        // 6. Subscribe the SQS queue to the SNS topic. Set the topic ARN in
        //    the request. Set the protocol to "sqs". Set the queue ARN to the
        //    ARN just received in step 5. For FIFO topics, give the option to
        //    apply a filter. A filter allows only matching messages to enter
        //    the queue.

        var q1Attributes: [String:String]? = nil

        if isFIFO {
            print(
                """
                If you add a filter to this subscription, then only the filtered
 messages will
                be received in the queue. For information about message
 filtering, see
                https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html
                For this example, you can filter messages by a 'tone' attribute.

                """
            )

            let subPrompt = """
                Would you like to filter messages for the first queue's
 subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q1Attributes = buildFilterAttributes()
            }
        }

        let sub1Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q1Attributes,
                endpoint: q1Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q1SubscriptionArn = sub1Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 7. Repeat steps 3-6 for the second queue.

        let q2Url = try await createQueue(prompt: "Enter the name of the second
 queue: ",
                                sqsClient: sqsClient, isFIFO: isFIFO)

        guard let q2Url else {
            print("Unable to create queue 2!")
            return
        }

        let q2Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q2Url)

        guard let q2Arn else {
            print("Unable to get ARN of queue 2!")
            return
        }
        print("Got queue 2 ARN: \(q2Arn)")

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q2Url,
                                 queueArn: q2Arn, topicArn: topicArn)

        var q2Attributes: [String:String]? = nil

        if isFIFO {
            let subPrompt = """
                Would you like to filter messages for the second queue's
subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q2Attributes = buildFilterAttributes()
            }
        }

        let sub2Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q2Attributes,
                endpoint: q2Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q2SubscriptionArn = sub2Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 8. Let the user publish messages to the topic, asking for a message
        //    body for each message. Handle the types of topic correctly (SEE
        //    MVP INFORMATION AND FIX THESE COMMENTS!!!
        print("\n\(rowOfStars)\n")

        var first = true

        repeat {
            var publishInput = PublishInput(
                topicArn: topicArn
            )

            publishInput.message = stringRequest(prompt: "Enter message text to
 publish: ")

            // If using a FIFO topic, a message group ID must be set on the
            // message.

            if isFIFO {
                if first {
                    print("""
                        Because you're using a FIFO topic, you must set a message
                        group ID. All messages within the same group will be
                        received in the same order in which they were published.
\n
                        """
                    )
                }
                publishInput.messageGroupId = stringRequest(prompt: "Enter a
 message group ID for this message: ")

                if !isContentBasedDeduplication {
                    if first {
                        print("""
                              Because you're not using content-based
 deduplication, you
                              must enter a deduplication ID. If other messages
 with the
                              same deduplication ID are published within the same
                              deduplication interval, they will not be delivered.
                              """
                        )
                    }
                    publishInput.messageDeduplicationId = stringRequest(prompt:
 "Enter a deduplication ID for this message: ")
                }
            }

            // Allow the user to add a value for the "tone" attribute if they
            // wish to do so.

            var messageAttributes: [String:SNSClientTypes.MessageAttributeValue]
= [:]
            let attrValSelection = menuRequest(prompt: "Choose a tone to apply to
this message.", options: attrValues)

            if attrValSelection != 0 {
                let val = SNSClientTypes.MessageAttributeValue(dataType:
"String", stringValue: attrValues[attrValSelection])
                messageAttributes["tone"] = val
            }

            publishInput.messageAttributes = messageAttributes

            // Publish the message and display its ID.

            let publishOutput = try await snsClient.publish(input: publishInput)

            guard let messageID = publishOutput.messageId else {
                print("Unable to get the published message's ID!")
                return
            }

            print("Message published with ID \(messageID).")
            first = false

            // 9. Repeat step 8 until the user says they don't want to post
            //    another.

        } while (yesNoRequest(prompt: "Post another message (Y/N)? "))

        // 10. Display a list of the messages in each queue by using
        //     `ReceiveMessage`. Show at least the body and the attributes.

        print(rowOfStars)
        print("Contents of queue 1:")
        let q1DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q1Url)
        print("\n\nContents of queue 2:")
        let q2DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q2Url)
        print(rowOfStars)

        returnRequest(prompt: "\nPress return to clean up: ")

        // 11. Delete the received messages using `DeleteMessageBatch`.

        print("Deleting the messages from queue 1...")
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q1Url,
deleteList: q1DeleteList)
        print("\nDeleting the messages from queue 2...")
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q2Url,
deleteList: q2DeleteList)

        // 12. Unsubscribe and delete both queues.

        print("\nUnsubscribing from queue 1...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q1SubscriptionArn)
        )

        print("Unsubscribing from queue 2...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q2SubscriptionArn)
        )

        print("Deleting queue 1...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q1Url)
        )

        print("Deleting queue 2...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q2Url)
        )

        // 13. Delete the topic.

        print("Deleting the SNS topic...")
        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: topicArn)
        )
    }
```

```
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Swift API reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use API Gateway to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by Amazon API Gateway.

Java

SDK for Java 2.x

Shows how to create an AWS Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create a Lambda function invoked by Amazon API Gateway that scans an Amazon DynamoDB table for work anniversaries and uses Amazon Simple Notification Service (Amazon SNS) to send a text message to your employees that congratulates them at their one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Shows how to create an AWS Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create a Lambda function invoked by Amazon API Gateway that scans an Amazon DynamoDB table for work anniversaries and uses Amazon Simple Notification Service (Amazon SNS) to send a text message to your employees that congratulates them at their one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

This example is also available in the AWS SDK for JavaScript v3 developer guide.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

This example shows how to create and use an Amazon API Gateway REST API that targets an AWS Lambda function. The Lambda handler demonstrates how to route based on HTTP methods; how to get data from the query string, header, and body; and how to return a JSON response.

- Deploy a Lambda function.
- Create an API Gateway REST API.
- Create a REST resource that targets the Lambda function.
- Grant permission to let API Gateway invoke the Lambda function.
- Use the Requests package to send requests to the REST API.
- Clean up all resources created during the demo.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use scheduled events to invoke a Lambda function

The following code examples show how to create an AWS Lambda function invoked by an Amazon EventBridge scheduled event.

Java

SDK for Java 2.x

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda Java runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Shows how to create an Amazon EventBridge scheduled event that invokes an AWS Lambda function. Configure EventBridge to use a cron expression to schedule when the Lambda function is invoked. In this example, you create a Lambda function by using the Lambda JavaScript runtime API. This example invokes different AWS services to perform a specific use case. This example demonstrates how to create an app that sends a mobile text message to your employees that congratulates them at the one year anniversary date.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

This example is also available in the AWS SDK for JavaScript v3 developer guide.

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

This example shows how to register an AWS Lambda function as the target of a scheduled Amazon EventBridge event. The Lambda handler writes a friendly message and the full event data to Amazon CloudWatch Logs for later retrieval.

- Deploys a Lambda function.
- Creates an EventBridge scheduled event and makes the Lambda function the target.
- Grants permission to let EventBridge invoke the Lambda function.
- Prints the latest data from CloudWatch Logs to show the result of the scheduled invocations.
- Cleans up all resources created during the demo.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Serverless examples for Amazon SNS

The following code examples show how to use Amazon SNS with AWS SDKs.

Examples

- Invoke a Lambda function from an Amazon SNS trigger

Invoke a Lambda function from an Amazon SNS trigger

The following code examples show how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
 {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;


import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consuming an SNS event with Lambda using TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK for PHP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
 marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
  raise
end
```

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features
 = ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SNS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon SNS security

The AWS shared responsibility model applies to data protection in Amazon Simple Notification Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.
- Message data protection
  - Message data protection is a new major feature of Amazon SNS
  - Use MDP to scan message for confidential or sensitive information
  - Provide message auditing to all content flowing through the topic
  - Provide content access controls to messages published to the topic and messages delivered by the topic

> ⚠ **Important**
>
> We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a Name field. This includes when you work with Amazon SNS or other Amazon Web Services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Amazon SNS data encryption

Data protection refers to protecting data while in-transit (as it travels to and from Amazon SNS) and at rest (while it is stored on disks in Amazon SNS data centers). You can protect data in transit using Secure Sockets Layer (SSL) or client-side encryption. By default, Amazon SNS stores messages and files using disk encryption. You can protect data at rest by requesting Amazon SNS to encrypt your messages before saving them to the encrypted file system in its data centers. Amazon SNS recommends using SSE for optimized data encryption.

Securing Amazon SNS data with server-side encryption

Server-side encryption (SSE) lets you store sensitive data in encrypted topics by protecting the contents of messages in Amazon SNS topics using keys managed in AWS Key Management Service (AWS KMS).

SSE encrypts messages as soon as Amazon SNS receives them. The messages are stored in encrypted form, and only decrypted when they are sent.

- For information about managing SSE using the AWS Management Console or the AWS SDK for Java (by setting the `KmsMasterKeyId` attribute using the `CreateTopic` and `SetTopicAttributes` API actions), see Setting up Amazon SNS topic encryption with server-side encryption.
- For information about creating encrypted topics using AWS CloudFormation (by setting the `KmsMasterKeyId` property using the `AWS::SNS::Topic` resource), see the *AWS CloudFormation User Guide*.

> ⚠ **Important**
>
> All requests to topics with SSE enabled must use HTTPS and Signature Version 4.
> For information about compatibility of other services with encrypted topics, see your service documentation.
> Amazon SNS only supports symmetric encryption KMS keys. You cannot use any other type of KMS key to encrypt your service resources. For help determining whether a KMS key is a symmetric encryption key, see Identifying asymmetric KMS keys.

AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. When you use Amazon SNS with AWS KMS, the data keys that encrypt your message data are also encrypted and stored with the data they protect.

The following are benefits of using AWS KMS:

- You can create and manage the AWS KMS key yourself.
- You can also use AWS-managed KMS keys for Amazon SNS, which are unique for each account and region.
- The AWS KMS security standards can help you meet encryption-related compliance requirements.

For more information, see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide*.

Encryption scope

SSE encrypts the body of a message in an Amazon SNS topic.

SSE doesn't encrypt the following:

- Topic metadata (topic name and attributes)
- Message metadata (subject, message ID, timestamp, and attributes)
- Data protection policy
- Per-topic metrics

Note

- A message is encrypted only if it is sent after the encryption of a topic is enabled. Amazon SNS doesn't encrypt backlogged messages.
- Any encrypted message remains encrypted even if the encryption of its topic is disabled.

Key terms

The following key terms can help you better understand the functionality of SSE. For detailed descriptions, see the *Amazon Simple Notification Service API Reference*.

Data key

The data encryption key (DEK) responsible for encrypting the contents of Amazon SNS messages.

For more information, see Data Keys in the *AWS Key Management Service Developer Guide* and Envelope Encryption in the *AWS Encryption SDK Developer Guide*.

AWS KMS key ID

The alias, alias ARN, key ID, or key ARN of an AWS KMS key, or a custom AWS KMS—in your account or in another account. While the alias of the AWS managed AWS KMS for Amazon SNS is always `alias/aws/sns`, the alias of a custom AWS KMS can, for example, be `alias/`*`MyAlias`*. You can use these AWS KMS keys to protect the messages in Amazon SNS topics.

Note

Keep the following in mind:

- The first time you use the AWS Management Console to specify the AWS managed KMS for Amazon SNS for a topic, AWS KMS creates the AWS managed KMS for Amazon SNS.
- Alternatively, the first time you use the `Publish` action on a topic with SSE enabled, AWS KMS creates the AWS managed KMS for Amazon SNS.

You can create AWS KMS keys, define the policies that control how AWS KMS keys can be used, and audit AWS KMS usage using the **AWS KMS keys** section of the AWS KMS console or the `CreateKey` AWS KMS action. For more information, see AWS KMS keys and Creating Keys in the *AWS Key Management Service Developer Guide*. For more examples of AWS KMS identifiers, see KeyId in the *AWS Key Management Service API Reference*. For information about finding AWS KMS identifiers, see Find the Key ID and ARN in the *AWS Key Management Service Developer Guide*.

⚠ **Important**

There are additional charges for using AWS KMS. For more information, see Estimating AWS KMS costs and AWS Key Management Service Pricing.

Managing Amazon SNS encryption keys and costs

The following sections provide information about working with keys managed in AWS Key Management Service (AWS KMS).

ⓘ **Note**

Amazon SNS only supports symmetric encryption KMS keys. You cannot use any other type of KMS key to encrypt your service resources. For help determining whether a KMS key is a symmetric encryption key, see Identifying asymmetric KMS keys.

Estimating AWS KMS costs

To predict costs and better understand your AWS bill, you might want to know how often Amazon SNS uses your AWS KMS key.

ⓘ **Note**

Although the following formula can give you a very good idea of expected costs, actual costs might be higher because of the distributed nature of Amazon SNS.

To calculate the number of API requests (R) *per topic*, use the following formula:

```
R = B / D * (2 * P)
```

B is the billing period (in seconds).

D is the data key reuse period (in seconds—Amazon SNS reuses a data key for up to 5 minutes).

P is the number of publishing principals that send to the Amazon SNS topic.

The following are example calculations. For exact pricing information, see AWS Key Management Service Pricing.

Example 1: Calculating the number of AWS KMS API calls for 1 publisher and 1 topic

This example assumes the following:

- The billing period is January 1-31 (2,678,400 seconds).
- The data key reuse period is 5 minutes (300 seconds).
- There is 1 topic.
- There is 1 publishing principal.

```
2,678,400 / 300 * (2 * 1) = 17,856
```

Example 2: Calculating the number of AWS KMS API calls for multiple publishers and 2 topics

This example assumes the following:

- The billing period is February 1-28 (2,419,200 seconds).
- The data key reuse period is 5 minutes (300 seconds).
- There are 2 topics.
- The first topic has 3 publishing principals.
- The second topic has 5 publishing principals.

```
(2,419,200 / 300 * (2 * 3)) + (2,419,200 / 300 * (2 * 5)) = 129,024
```

Configuring AWS KMS permissions

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of topics and encryption and decryption of messages. For examples and more information about AWS KMS permissions, see AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*. For details on how to set up an Amazon SNS topic with server-side encryption, see Additional information.

> **Note**
>
> You can also manage permissions for symmetric encryption KMS keys using IAM policies. For more information, see Using IAM Policies with AWS KMS.
> While you can configure global permissions to send to and receive from Amazon SNS, AWS KMS requires explicitly naming the full ARN of KMSs in specific regions in the `Resource` section of an IAM policy.

You must also ensure that the key policies of the AWS KMS key allow the necessary permissions. To do this, name the principals that produce and consume encrypted messages in Amazon SNS as users in the KMS key policy.

Alternatively, you can specify the required AWS KMS actions and KMS ARN in an IAM policy assigned to the principals that publish and subscribe to receive encrypted messages in Amazon SNS. For more information, see Managing Access to AWS KMS in the *AWS Key Management Service Developer Guide*.

If selecting a customer-managed key for your Amazon SNS topic and you are using aliases to control access to KMS keys using IAM policies or KMS key policies with the condition key `kms:ResourceAliases`, ensure that the customer-managed key that is selected also has an alias associated. For more information on using alias to control access to KMS keys, see Using aliases to control access to KMS keys in the *AWS Key Management Service Developer Guide*.

Allow a user to send messages to a topic with SSE

The publisher must have the `kms:GenerateDataKey*` and `kms:Decrypt` permissions for the AWS KMS key.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey*",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-
east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:123456789012:MyTopic"
  }]
}
```

Enable compatibility between event sources from AWS services and encrypted topics

Several AWS services publish events to Amazon SNS topics. To allow these event sources to work with encrypted topics, you must perform the following steps.

1. Use a customer managed key. For more information, see Creating Keys in the *AWS Key Management Service Developer Guide*.
2. To allow the AWS service to have the `kms:GenerateDataKey*` and `kms:Decrypt` permissions, add the following statement to the KMS policy.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey*",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }]
}
```

Event source	Service principal
Amazon CloudWatch	`cloudwatch.amazonaws.com`
Amazon CloudWatch Events	`events.amazonaws.com`
AWS CodeCommit	`codecommit.amazonaws.com`
AWS Database Migration Service	`dms.amazonaws.com`
AWS Directory Service	`ds.amazonaws.com`
Amazon DynamoDB	`dynamodb.amazonaws.com`
Amazon Inspector	`inspector.amazonaws.com`
Amazon Redshift	`redshift.amazonaws.com`
Amazon RDS	`events.rds.amazonaws.com`
Amazon S3 Glacier	`glacier.amazonaws.com`
Amazon Simple Email Service	`ses.amazonaws.com`
Amazon Simple Storage Service	`s3.amazonaws.com`
AWS Snowball Edge	`importexport.amazonaws.com`
AWS Systems Manager Incident Manager	AWS Systems Manager Incident Manager consists of two service principles: `ssm-incidents.amazonaws.com` ; `ssm-contacts.amazonaws.com`

Note

Some Amazon SNS event sources require you to provide an IAM role (rather than the service principal) in the AWS KMS key policy:

- Amazon EC2 Auto Scaling

- Amazon Elastic Transcoder
- AWS CodePipeline
- AWS Config
- AWS Elastic Beanstalk
- AWS IoT
- EC2 Image Builder

3. Add the `aws:SourceAccount` and `aws:SourceArn` condition keys to the KMS resource policy to further protect the KMS key from confused deputy attacks. Refer to service specific documentation list (above) for exact details in each case.

> ⚠ **Important**
>
> Adding the `aws:SourceAccount`, `aws:SourceArn`, and `aws:SourceOrgID` to a AWS KMS policy is not supported for EventBridge-to-encrypted topics.

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "service.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey*",
    "kms:Decrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "customer-account-id"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:service:region:customer-account-id:resource-
type:customer-resource-id"
    }
  }
}
```

4. Enable SSE for your topic using your KMS.

5. Provide the ARN of the encrypted topic to the event source.

AWS KMS errors

When you work with Amazon SNS and AWS KMS, you might encounter errors. The following list describes the errors and possible troubleshooting solutions.

KMSAccessDeniedException

The ciphertext references a key that doesn't exist or that you don't have access to.

HTTP Status Code: 400

KMSDisabledException

The request was rejected because the specified KMS isn't enabled.

HTTP Status Code: 400

KMSInvalidStateException

The request was rejected because the state of the specified resource isn't valid for this request. For more information, see Key states of AWS KMS keys in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

KMSNotFoundException

The request was rejected because the specified entity or resource can't be found.

HTTP Status Code: 400

KMSOptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

KMSThrottlingException

The request was denied due to request throttling. For more information about throttling, see Quotas in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

Setting up Amazon SNS topic encryption with server-side encryption

Amazon SNS supports server-side encryption (SSE) to protect the contents of messages using AWS Key Management Service (AWS KMS). Follow the instructions below to enable SSE using the Amazon SNS console or CDK.

Option 1: Enable encryption using the AWS Management Console

1. Sign in to the Amazon SNS console.
2. Navigate to the **Topics** page, select your **topic**, and choose **Edit**.
3. Expand the **Encryption** section and do the following:
   - Toggle encryption to **Enable**.
   - Select the **AWS managed SNS Key** (alias/aws/sns) as the encryption key. This is selected by default.
4. Choose **Save changes**.

> **Note**
>
> - The AWS managed key is automatically created if it doesn’t already exist.
> - If you don’t see the key or have insufficient permissions, ask your administrator for `kms:ListAliases` and `kms:DescribeKey`.

Option 2: Enable encryption using AWS CDK

To use the AWS managed SNS key in your CDK application, add the following snippet:

```
import software.amazon.awscdk.services.sns.*;
import software.amazon.awscdk.services.kms.*;
import software.amazon.awscdk.core.*;

public class SnsEncryptionExample extends Stack {
    public SnsEncryptionExample(final Construct scope, final String id) {
        super(scope, id);

        // Define the managed SNS key
        IKey snsKey = Alias.fromAliasName(this, "helloKey", "alias/aws/sns");
        // Create the SNS Topic with encryption enabled
        Topic.Builder.create(this, "MyEncryptedTopic")
            .masterKey(snsKey)
            .build();
    }
}
```

Additional information

- **Custom KMS key** – You can specify a custom key if required. In the Amazon SNS console, select your custom KMS key from the list or enter the ARN.
- **Permissions for custom KMS keys** – If using a custom KMS key, include the following in the key policy to allow Amazon SNS to encrypt and decrypt messages:

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "sns.amazonaws.com"
     },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:service:region:customer-account-id:resource-
type/customer-resource-id"
        },
        "StringEquals": {
            "kms:EncryptionContext:aws:sns:topicArn":
 "arn:aws:sns:your_region:customer-account-id:your_sns_topic_name"
        }
    }
}
```

Impact on consumers

Enabling SSE does not change how subscribers consume messages. AWS manages encryption and decryption transparently. Messages remain encrypted at rest and are automatically decrypted before delivery to subscribers. For optimal security, AWS recommends enabling HTTPS for all endpoints to ensure secure transmission of messages.

Setting up Amazon SNS topic encryption with encrypted Amazon SQS queue subscription

You can enable server-side encryption (SSE) for a topic to protect its data. To allow Amazon SNS to send messages to encrypted Amazon SQS queues, the customer managed key associated with the Amazon SQS queue must have a policy statement that grants Amazon SNS service-principal access to the AWS KMS API actions `GenerateDataKey` and `Decrypt`. For more information about using SSE, see Securing Amazon SNS data with server-side encryption.

This topic explains how to enable SSE for an Amazon SNS topic with an encrypted Amazon SQS queue subscription using the AWS Management Console.

Step 1: Create a custom KMS key

1. Sign in to the AWS KMS console with a user that has at least the `AWSKeyManagementServicePowerUser` policy.
2. Choose **Create a key**.
3. To create a symmetric encryption KMS key, for **Key type** choose **Symmetric**.

   For information about how to create an asymmetric KMS key in the AWS KMS console, see Creating asymmetric KMS keys (console).
4. In **Key usage**, the **Encrypt and decrypt** option is selected for you.

   For information about how to create KMS keys that generate and verify MAC codes, see Creating HMAC KMS keys.

   For information about the **Advanced options**, see Special-purpose keys.
5. Choose **Next**.
6. Type an alias for the KMS key. The alias name cannot begin with **`aws/`**. The **`aws/`** prefix is reserved by Amazon Web Services to represent AWS managed keys in your account.

> **Note**
>
> Adding, deleting, or updating an alias can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using aliases to control access to KMS keys.

An alias is a display name that you can use to identify the KMS key. We recommend that you choose an alias that indicates the type of data you plan to protect or the application you plan to use with the KMS key.

Aliases are required when you create a KMS key in the AWS Management Console. They are optional when you use the CreateKey operation.

7. (Optional) Type a description for the KMS key.

   You can add a description now or update it any time unless the key state is `Pending Deletion` or `Pending Replica Deletion`. To add, change, or delete the description of an existing customer managed key, edit the description in the AWS Management Console or use the UpdateKeyDescription operation.

8. (Optional) Type a tag key and an optional tag value. To add more than one tag to the KMS key, choose **Add tag**.

   > **Note**
   >
   > Tagging or untagging a KMS key can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using tags to control access to KMS keys.

   When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

9. Choose **Next**.
10. Select the IAM users and roles that can administer the KMS key.

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to manage the KMS key. For details, see Default key policy.
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the IAM User Guide.

11. (Optional) To prevent the selected IAM users and roles from deleting this KMS key, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.
12. Choose **Next**.
13. Select the IAM users and roles that can use the key in cryptographic operations. Choose **Next**.
14. On the **Review and edit key policy** page, add the following statement to the key policy, and then choose **Finish**.

```
{
    "Sid": "Allow Amazon SNS to use this key",
    "Effect": "Allow",
    "Principal": {
        "Service": "sns.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*"
}
```

Your new customer managed key appears in the list of keys.

Step 2: Create an encrypted Amazon SNS topic

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Topics**.

3. Choose **Create topic**.
4. On the **Create new topic** page, for **Name**, enter a topic name (for example, `MyEncryptedTopic`) and then choose **Create topic**.
5. Expand the **Encryption** section and do the following:

   a. Choose **Enable server-side encryption**.

   b. Specify the customer managed key. For more information, see Key terms.

      For each customer managed key type, the **Description**, **Account**, and customer managed key **ARN** are displayed.

      > ⚠ **Important**
      >
      > If you aren't the owner of the customer managed key, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the customer managed key on the Amazon SNS console.
      > Ask the owner of the customer managed key to grant you these permissions.
      > For more information, see the AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*.

   c. For **customer managed key**, choose **MyCustomKey** which you created earlier and then choose **Enable server-side encryption**.
6. Choose **Save changes**.

   SSE is enabled for your topic and the **MyTopic** page is displayed.

   The topic's **Encryption** status, AWS **Account**, **customer managed key**, customer managed key **ARN**, and **Description** are displayed on the **Encryption** tab.

Your new encrypted topic appears in the list of topics.

Step 3: Create and subscribe encrypted Amazon SQS queues

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue**.
3. On the **Create New Queue** page, do the following:

   a. Enter a **Queue Name** (for example, `MyEncryptedQueue1`).
   b. Choose **Standard Queue**, and then choose **Configure Queue**.
   c. Choose **Use SSE**.
   d. For **AWS KMS key**, choose **MyCustomKey** which you created earlier, and then choose **Create Queue**.

4. Repeat the process to create a second queue (for example, named `MyEncryptedQueue2`).

   Your new encrypted queues appear in the list of queues.

5. On the Amazon SQS console, choose `MyEncryptedQueue1` and `MyEncryptedQueue2` and then choose **Queue Actions**, **Subscribe Queues to SNS Topic**.
6. In the **Subscribe to a Topic** dialog box, for **Choose a Topic** select **MyEncryptedTopic**, and then choose **Subscribe**.

   Your encrypted queues' subscriptions to your encrypted topic are displayed in the **Topic Subscription Result** dialog box.

7. Choose **OK**.

Step 4: Publish a message to your encrypted topic

1. Sign in to the Amazon SNS console.
2. On the navigation panel, choose **Topics**.
3. From the list of topics, choose **MyEncryptedTopic** and then choose **Publish message**.
4. On the **Publish a message** page, do the following:

   a. (Optional) In the **Message details** section, enter the **Subject** (for example, `Testing message publishing`).
   b. In the **Message body** section, enter the message body (for example, `My message body is encrypted at rest.`).
   c. Choose **Publish message**.

Your message is published to your subscribed encrypted queues.

Step 5: Verify message delivery

1. Sign in to the Amazon SQS console.
2. From the list of queues, choose **MyEncryptedQueue1** and then choose **Send and receive messages**.
3. On the **Send and receive messages in MyEncryptedQueue1** page, choose **Poll for messages**.

   The message that you sent earlier is displayed.
4. Choose **More Details** to view your message.
5. When you're finished, choose **Close**.
6. Repeat the process for **MyEncryptedQueue2**.

Securing Amazon SNS traffic with VPC endpoints

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon SNS is a logical entity within a VPC that allows connectivity only to Amazon SNS. The VPC routes requests to Amazon SNS and routes responses back to the VPC. The following sections provide information about working with VPC endpoints and creating VPC endpoint policies.

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and Amazon SNS. With this connection, you can publish messages to your Amazon SNS topics without sending them through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. To connect your VPC to Amazon SNS, you define an *interface VPC endpoint*. This type of endpoint enables you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to Amazon SNS without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon VPC User Guide*.

The information in this section is for users of Amazon VPC. For more information, and to get started with creating a VPC, see Plan your VPC in the *Amazon VPC User Guide*.

> **Note**
>
> VPC endpoints don't allow you to subscribe an Amazon SNS topic to a private IP address.

Creating an Amazon VPC endpoint for Amazon SNS

To publish messages to your Amazon SNS topics from an Amazon VPC, create an interface VPC endpoint. Then, you can publish messages to your topics while keeping the traffic within the network that you manage with the VPC.

Use the following information to create the endpoint and test the connection between your VPC and Amazon SNS. Or, for a walkthrough that helps you start from scratch, see Publishing an Amazon SNS message from Amazon VPC.

Creating the endpoint

You can create an Amazon SNS endpoint in your VPC using the AWS Management Console, the AWS CLI, an AWS SDK, the Amazon SNS API, or AWS CloudFormation.

For information about creating and configuring an endpoint using the Amazon VPC console or the AWS CLI, see Creating an Interface Endpoint in the *Amazon VPC User Guide*.

> ⚠ **Important**
>
> You can use Amazon Virtual Private Cloud only with HTTPS Amazon SNS endpoints.
> When you create an endpoint, specify Amazon SNS as the service that you want your VPC to connect to. In the Amazon VPC console, service names vary based on the region. For example, if you choose US East (N. Virginia), the service name is **com.amazonaws.us-east-1.sns**.
> When you configure Amazon SNS to send messages from Amazon VPC, you must enable private DNS and specify endpoints in the format `sns.`*`us-east-2`*`.amazonaws.com`. Private DNS doesn't support legacy endpoints such as `queue.amazonaws.com` or *`us-east-2`*`.queue.amazonaws.com`.

For information about creating and configuring an endpoint using AWS CloudFormation, see the `AWS::EC2::VPCEndpoint` resource in the *AWS CloudFormation User Guide*.

Testing the connection between your VPC and Amazon SNS

After you create an endpoint for Amazon SNS, you can publish messages from your VPC to your Amazon SNS topics. To test this connection, do the following:

1. Connect to an Amazon EC2 instance that resides in your VPC. For information about connecting, see Connect to Your Linux Instance or Connecting to Your Windows Instance in the Amazon EC2 documentation.

   For example, to connect to a Linux instance using an SSH client, run the following command from a terminal:

   ```
   $ ssh -i ec2-key-pair.pem ec2-user@instance-hostname
   ```

   Where:

   - *ec2-key-pair.pem* is the file that contains the key pair that Amazon EC2 provided when you created the instance.
   - *instance-hostname* is the public hostname of the instance. To get the hostname in the Amazon EC2 console: Choose **Instances**, choose your instance, and find the value for **Public DNS**.

2. From your instance, use the Amazon SNS `publish` command with the AWS CLI. You can send a simple message to a topic with the following command:

   ```
   $ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
   ```

   Where:

   - *aws-region* is the AWS Region that the topic is located in.
   - *sns-topic-arn* is the Amazon Resource Name (ARN) of the topic. To get the ARN from the Amazon SNS console: Choose **Topics**, find your topic, and find the value in the **ARN** column.

   If the message is successfully received by Amazon SNS, the terminal prints a message ID, like the following:

   ```
   {
      "MessageId": "6c96dfff-0fdf-5b37-88d7-8cba910a8b64"
   }
   ```

Creating an Amazon VPC endpoint policy for Amazon SNS

You can create a policy for Amazon VPC endpoints for Amazon SNS in which you specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*.

The following example VPC endpoint policy specifies that the IAM user `MyUser` is allowed to publish to the Amazon SNS topic `MyTopic`.

```
{
  "Statement": [{
    "Action": ["sns:Publish"],
    "Effect": "Allow",
    "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

The following are denied:

- Other Amazon SNS API actions, such as `sns:Subscribe` and `sns:Unsubscribe`.
- Other IAM users and rules which attempt to use this VPC endpoint.
- `MyUser` publishing to a different Amazon SNS topic.

> **Note**
>
> The IAM user can still use other Amazon SNS API actions from *outside* the VPC.

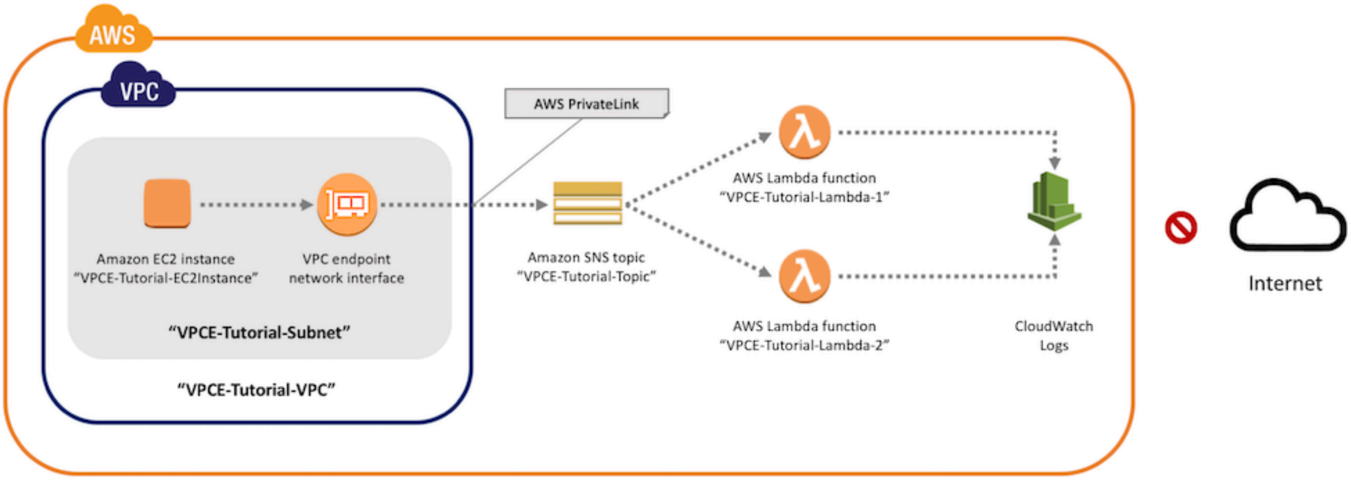
Publishing an Amazon SNS message from Amazon VPC

This section describes how to publish to an Amazon SNS topic while keeping the messages secure in a private network. You publish a message from an Amazon EC2 instance that's hosted in Amazon Virtual Private Cloud (Amazon VPC). The message stays within the AWS network without traveling the public internet. By publishing messages privately from a VPC, you can improve the security of the traffic between your applications and Amazon SNS. This security is important when you publish personally identifiable information (PII) about your customers, or when your application is subject to market regulations. For example, publishing privately is helpful if you have a healthcare system that must comply with the Health Insurance Portability and Accountability Act (HIPAA), or a financial system that must comply with the Payment Card Industry Data Security Standard (PCI DSS).

The general steps are as follows:

- Use an AWS CloudFormation template to automatically create a temporary private network in your AWS account.
- Create a VPC endpoint that connects the VPC with Amazon SNS.
- Log in to an Amazon EC2 instance and publish a message privately to an Amazon SNS topic.
- Verify that the message was delivered successfully.
- Delete the resources that you created during this process so that they don't remain in your AWS account.

The following diagram depicts the private network that you create in your AWS account as you complete these steps:



This network consists of a VPC that contains an Amazon EC2 instance. The instance connects to Amazon SNS through an *interface VPC endpoint*. This type of endpoint connects to services that are powered by AWS PrivateLink. With this connection established, you can log in to the Amazon EC2 instance and publish messages to the Amazon SNS topic, even though the network is disconnected from the public internet. The topic fans out the messages that it receives to two subscribing AWS Lambda functions. These functions log the messages that they receive in Amazon CloudWatch Logs.

It takes about 20 minutes to complete these steps.

Topics

Before you begin

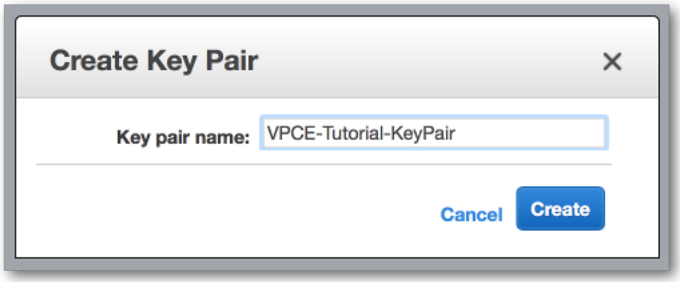
Before you start, you need an Amazon Web Services (AWS) account. When you sign up, your account is automatically signed up for all services in AWS, including Amazon SNS and Amazon VPC. If you haven't created an account already, go to https://aws.amazon.com/, and then choose **Create a Free Account**.

Step 1: Create an Amazon EC2 key pair

A *key pair* is used to log in to an Amazon EC2 instance. It consists of a public key that's used to encrypt your login information, and a private key that's used to decrypt it. When you create a key pair, you download a copy of the private key. Later, you use the key pair to log in to an Amazon EC2 instance. To log in, you specify the name of the key pair, and you provide the private key.

To create the key pair

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation menu on the left, find the **Network & Security** section. Then, choose **Key Pairs**.
3. Choose **Create Key Pair**.
4. In the **Create Key Pair** window, for **Key pair name**, type **`VPCE-Tutorial-KeyPair`**. Then, choose **Create**.



Create Key Pair

Key pair name: VPCE-Tutorial-KeyPair

5. The private key file is automatically downloaded by your browser. Save it in a safe place. Amazon EC2 gives the file an extension of `.pem`.
6. (Optional) If you're using an SSH client on a Mac or Linux computer to connect to your instance, use the `chmod` command to set the permissions of your private key file so that only you can read it:

   a. Open a terminal and navigate to the directory that contains the private key:

   ```
   $ cd /filepath_to_private_key/
   ```

   b. Set the permissions using the following command:

   ```
   $ chmod 400 VPCE-Tutorial-KeyPair.pem
   ```

Step 2: Create the AWS resources

To set up the infrastructure, you use an AWS CloudFormation *template*. A template is a file that acts as a blueprint for building AWS resources, such as Amazon EC2 instances and Amazon SNS topics. The template for this process is provided on GitHub for you to download.

You provide the template to AWS CloudFormation, and AWS CloudFormation provisions the resources that you need as a *stack* in your AWS account. A stack is a collection of resources that you manage as a single unit. When you finish these steps, you can use AWS CloudFormation to delete all of the resources in the stack at once. These resources don't remain in your AWS account, unless you want them to.

The stack for this process includes the following resources:

- A VPC and the associated networking resources, including a subnet, a security group, an internet gateway, and a route table.
- An Amazon EC2 instance that's launched into the subnet in the VPC.
- An Amazon SNS topic.
- Two AWS Lambda functions. These functions receive messages that are published to the Amazon SNS topic, and they log events in CloudWatch Logs.
- Amazon CloudWatch metrics and logs.
- An IAM role that allows the Amazon EC2 instance to use Amazon SNS, and an IAM role that allows the Lambda functions to write to CloudWatch logs.

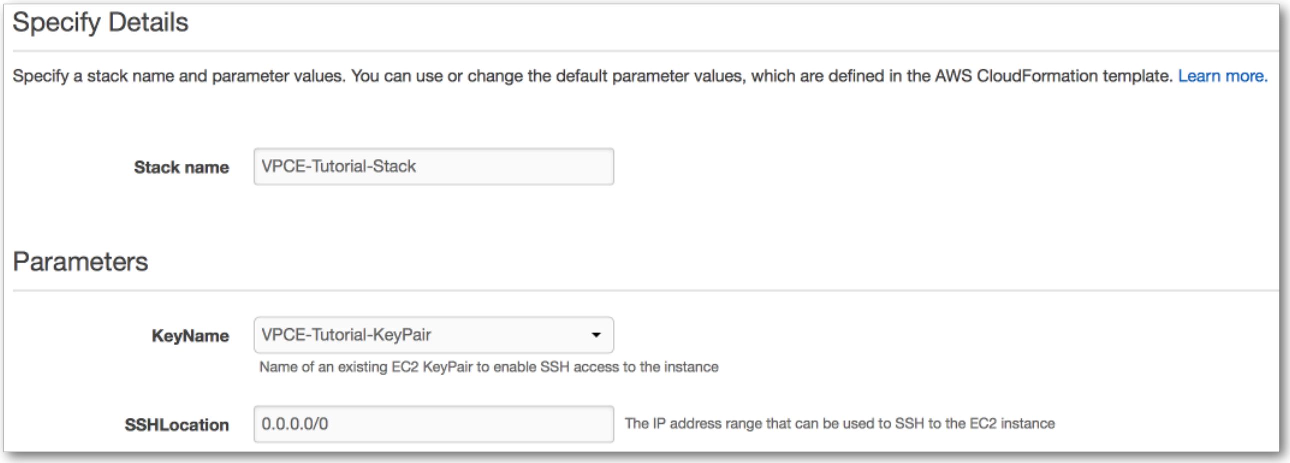
To create the AWS resources

1. Download the template file from the GitHub website.
2. Sign in to the AWS CloudFormation console.
3. Choose **Create Stack**.
4. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose the file, and choose **Next**.
5. On the **Specify Details** page, specify stack and key names:

    a. For **Stack name**, type **`VPCE-Tutorial-Stack`**.

    b. For **KeyName**, choose **VPCE-Tutorial-KeyPair**.

    c. For **SSHLocation**, keep the default value of **`0.0.0.0/0`**.



Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.

Stack name: VPCE-Tutorial-Stack

Parameters

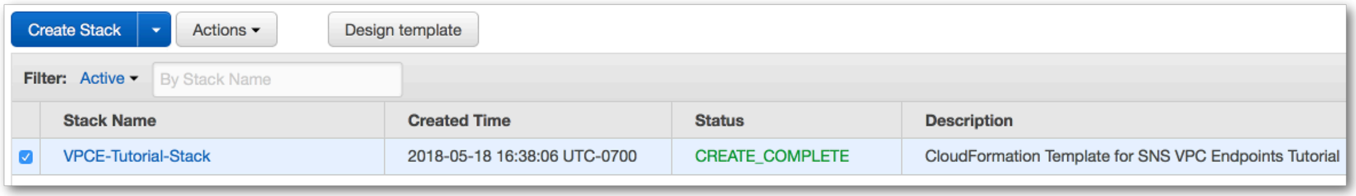
KeyName: VPCE-Tutorial-KeyPair
Name of an existing EC2 KeyPair to enable SSH access to the instance

SSHLocation: 0.0.0.0/0
The IP address range that can be used to SSH to the EC2 instance

d. Choose **Next**.

6. On the **Options** page, keep all of the default values, and choose **Next**.
7. On the **Review** page, verify the stack details.
8. Under **Capabilities**, acknowledge that AWS CloudFormation might create IAM resources with custom names.
9. Choose **Create**.

The AWS CloudFormation console opens the **Stacks** page. The VPCE-Tutorial-Stack has a status of **CREATE_IN_PROGRESS**. In a few minutes, after the creation process completes, the status changes to **CREATE_COMPLETE**.



	Stack Name	Created Time	Status	Description
☑	VPCE-Tutorial-Stack	2018-05-18 16:38:06 UTC-0700	CREATE_COMPLETE	CloudFormation Template for SNS VPC Endpoints Tutorial

Tip

Choose the **Refresh** button to see the latest stack status.

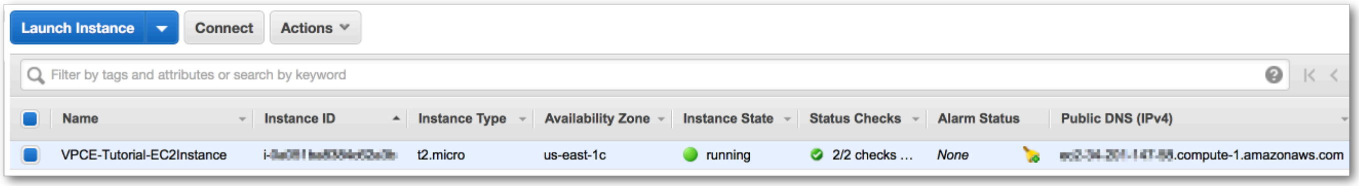
Step 3: Confirm that your Amazon EC2 instance lacks internet access

The Amazon EC2 instance that was launched in your VPC in the previous step lacks internet access. It disallows outbound traffic, and it's unable to publish messages to Amazon SNS. Verify this by logging in to the instance. Then, attempt to connect to a public endpoint, and attempt to message Amazon SNS.

At this point, the publish attempt fails. In a later step, after you create a VPC endpoint for Amazon SNS, your publish attempt succeeds.

To connect to your Amazon EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation menu on the left, find the **Instances** section. Then, choose **Instances**.
3. In the list of instances, select **VPCE-Tutorial-EC2Instance**.
4. Copy the hostname that's provided in the **Public DNS** column.



☐	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
■	VPCE-Tutorial-EC2Instance	[redacted]	t2.micro	us-east-1c	● running	✓ 2/2 checks ...	None	[redacted].compute-1.amazonaws.com

5. Open a terminal. From the directory that contains the key pair, connect to the instance using the following command, where *instance-hostname* is the hostname that you copied from the Amazon EC2 console:

```
$ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
```

To verify that the instance lacks internet connectivity

- In your terminal, attempt to connect to any public endpoint, such as amazon.com:

```
$ ping amazon.com
```

  Because the connection attempt fails, you can cancel at any time (Ctrl + C on Windows or Command + C on macOS).

To verify that the instance lacks connectivity to Amazon SNS

1. Sign in to the Amazon SNS console.
2. In the navigation menu on the left, choose **Topics**.

3. On the **Topics** page, copy the Amazon Resource Name (ARN) for the topic **VPCE-Tutorial-Topic**.

4. In your terminal, attempt to publish a message to the topic:

   ```
   $ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
   ```

   Because the publish attempt fails, you can cancel at any time.

Step 4: Create an Amazon VPC endpoint for Amazon SNS

To connect the VPC to Amazon SNS, you define an interface VPC endpoint. After you add the endpoint, you can log in to the Amazon EC2 instance in your VPC, and from there you can use the Amazon SNS API. You can publish messages to the topic, and the messages are published privately. They stay within the AWS network, and they don't travel the public internet.

> **Note**
>
> The instance still lacks access to other AWS services and endpoints on the internet.

To create the endpoint

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

2. In the navigation menu on the left, choose **Endpoints**.

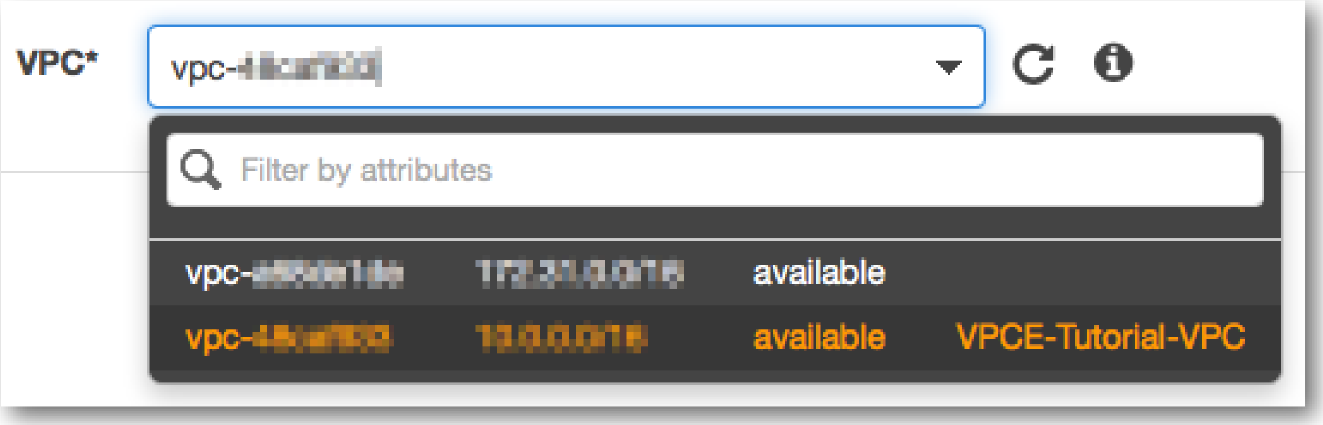
3. Choose **Create Endpoint**.

4. On the **Create Endpoint** page, for **Service category**, keep the default choice of **AWS services**.

5. For **Service Name**, choose the service name for Amazon SNS.

   The service names vary based on the chosen region. For example, if you chose US East (N. Virginia), the service name is **com.amazonaws.*us-east-1*.sns**.

6. For **VPC**, choose the VPC that has the name **VPCE-Tutorial-VPC**.

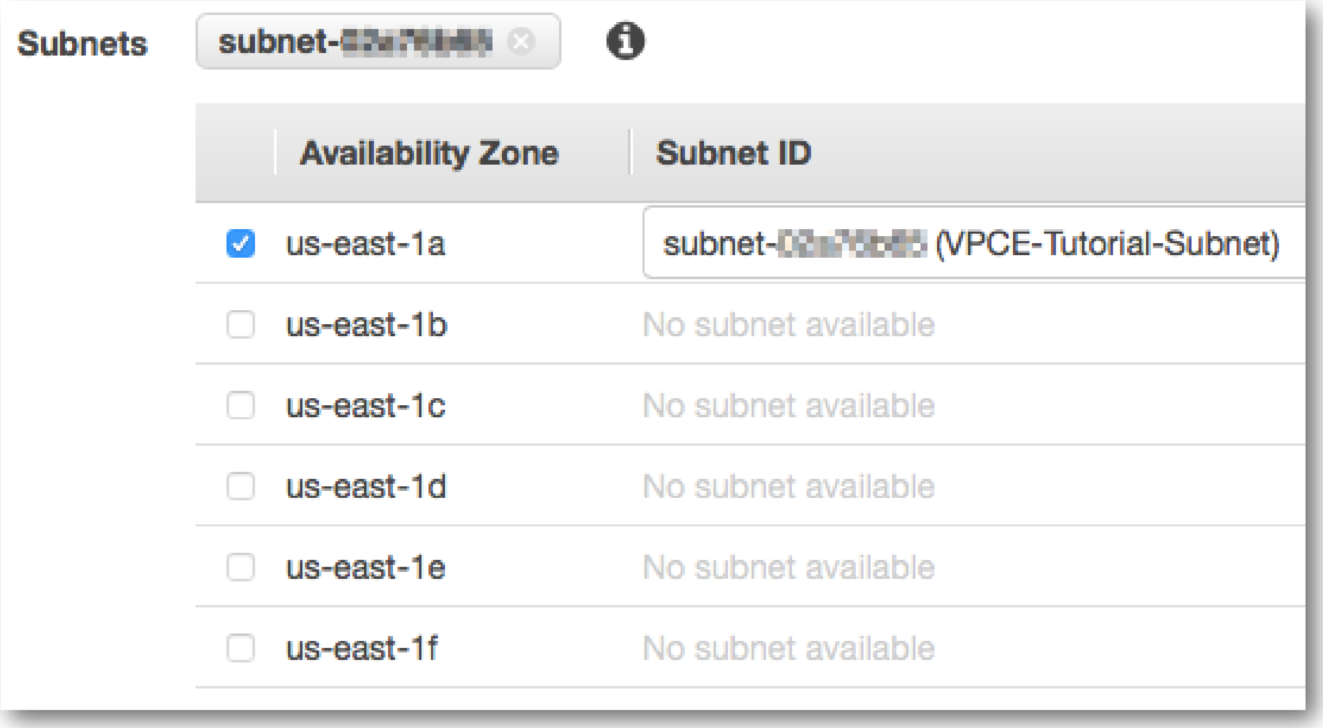


VPC* vpc-[redacted]

Filter by attributes

vpc-[redacted]	[redacted]	available	
vpc-[redacted]	[redacted]	available	VPCE-Tutorial-VPC

7. For **Subnets**, choose the subnet that has *VPCE-Tutorial-Subnet* in the subnet ID.

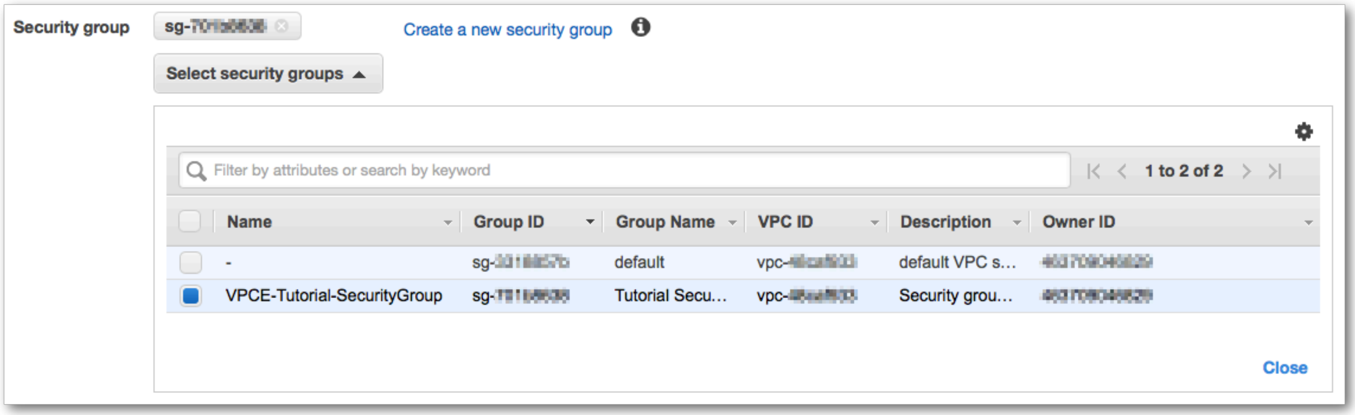


Subnets subnet-[redacted]

	Availability Zone	Subnet ID
☑	us-east-1a	subnet-[redacted] (VPCE-Tutorial-Subnet)
□	us-east-1b	No subnet available
□	us-east-1c	No subnet available
□	us-east-1d	No subnet available
□	us-east-1e	No subnet available
□	us-east-1f	No subnet available

8. For **Enable Private DNS Name**, select **Enable for this endpoint**.

9. For **Security group**, choose **Select security group**, and choose **VPCE-Tutorial-SecurityGroup**.



Security group sg-[redacted] Create a new security group

Select security groups ▲

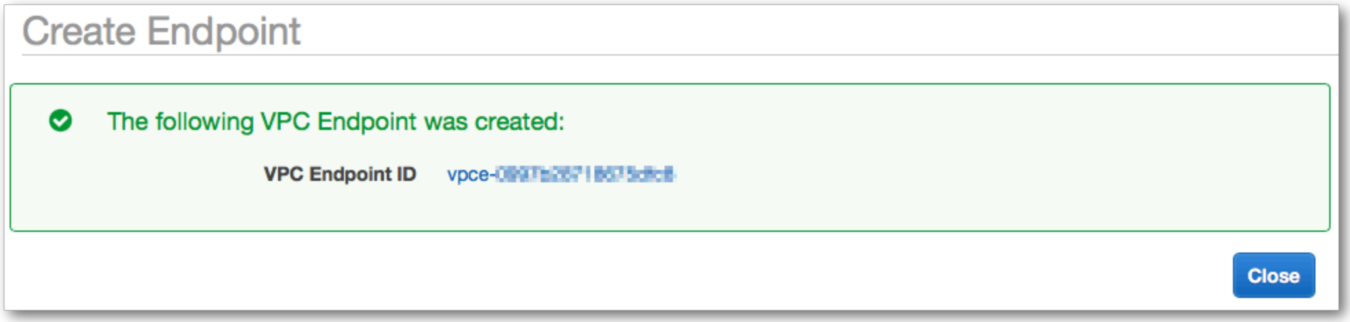
Filter by attributes or search by keyword

1 to 2 of 2

	Name	Group ID	Group Name	VPC ID	Description	Owner ID
□	-	sg-[redacted]	default	vpc-[redacted]	default VPC s...	[redacted]
☑	VPCE-Tutorial-SecurityGroup	sg-[redacted]	Tutorial Secu...	vpc-[redacted]	Security grou...	[redacted]

Close

10. Choose **Create endpoint**. The Amazon VPC console confirms that a VPC endpoint was created.



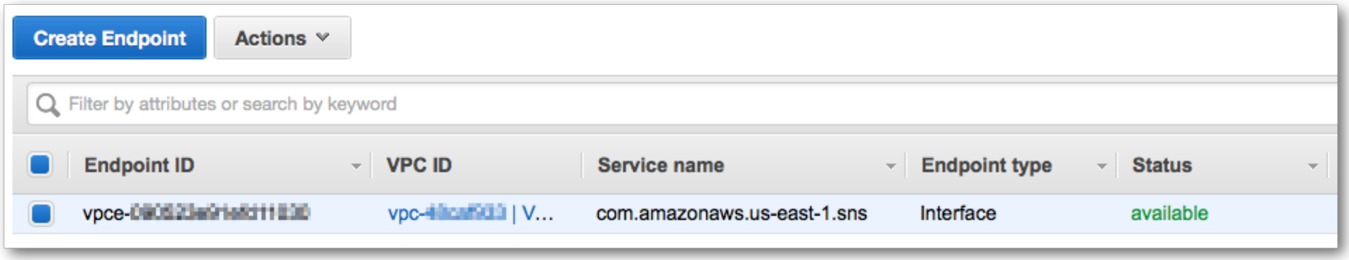
Create Endpoint

The following VPC Endpoint was created:

VPC Endpoint ID vpce-[redacted]

11. Choose **Close**.

    The Amazon VPC console opens the **Endpoints** page. The new endpoint has a status of **pending**. In a few minutes, after the creation process completes, the status changes to **available**.



Filter by attributes or search by keyword

☑	Endpoint ID	VPC ID	Service name	Endpoint type	Status
☑	vpce-[redacted]	vpc-[redacted] | V...	com.amazonaws.us-east-1.sns	Interface	available

Step 5: Publish a message to your Amazon SNS topic

Now that your VPC includes an endpoint for Amazon SNS, you can log in to the Amazon EC2 instance and publish messages to the topic.

To publish a message

1. If your terminal is no longer connected to your Amazon EC2 instance, connect again:

   ```
   $ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
   ```

2. Run the same command that you did previously to publish a message to your Amazon SNS topic. This time, the publish attempt succeeds, and Amazon SNS returns a message ID:

   ```
   $ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"


   {
       "MessageId": "5b111270-d169-5be6-9042-410dfc9e86de"
   }
   ```

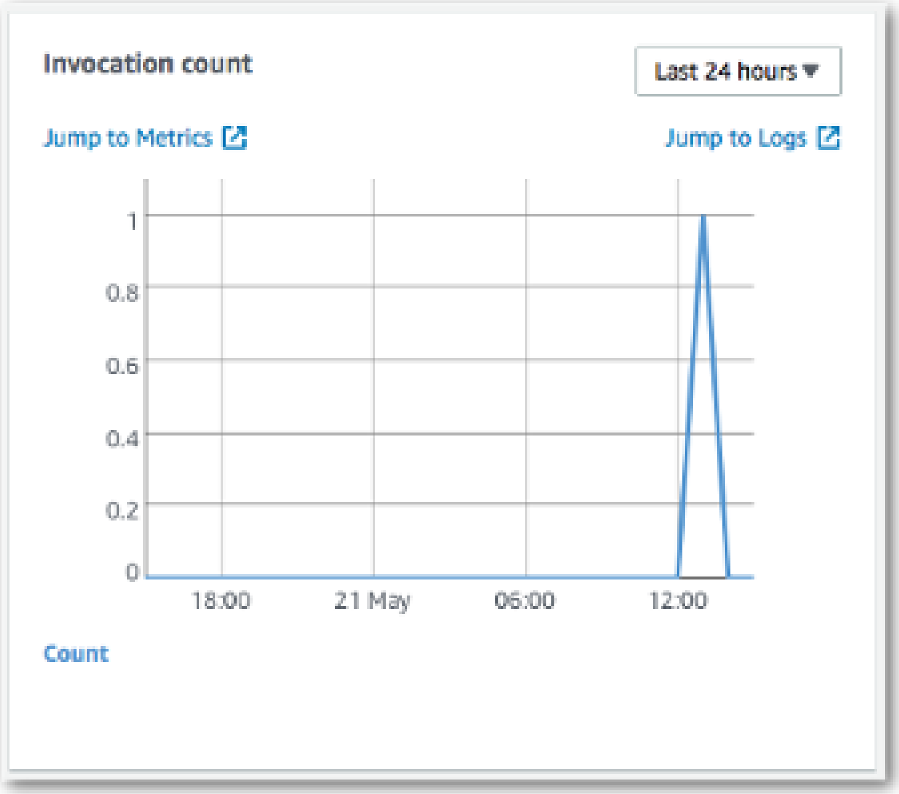
Step 6: Verify your message deliveries

When the Amazon SNS topic receives a message, it fans out the message by sending it to the two subscribing Lambda functions. When these functions receive the message, they log the event to CloudWatch logs. To verify that your message delivery succeeded, check that the functions were invoked, and check that the CloudWatch logs were updated.

To verify that the Lambda functions were invoked

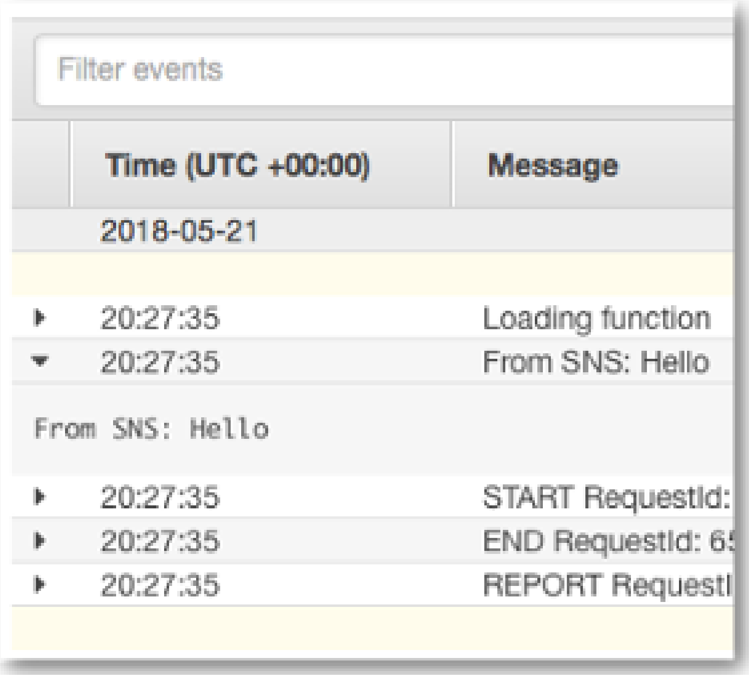
1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. On the **Functions** page, choose **VPCE-Tutorial-Lambda-1**.
3. Choose **Monitoring**.
4. Check the **Invocation count** graph. This graph shows the number of times that the Lambda function has been run.

   The invocation count matches the number of times you published a message to the topic.



To verify that the CloudWatch logs were updated

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation menu on the left, choose **Logs**.
3. Check the logs that were written by the Lambda functions:
   a. Choose the **/aws/lambda/VPCE-Tutorial-Lambda-1/** log group.
   b. Choose the log stream.
   c. Check that the log includes the entry `From SNS: Hello`.



	Time (UTC +00:00)	Message
	2018-05-21	
▸	20:27:35	Loading function
▾	20:27:35	From SNS: Hello

```
From SNS: Hello
```

▸	20:27:35	START RequestId:
▸	20:27:35	END RequestId: 6
▸	20:27:35	REPORT RequestI

d. Choose **Log Groups** at the top of the console to return the **Log Groups** page. Then, repeat the preceding steps for the /aws/lambda/VPCE-Tutorial-Lambda-2/ log group.

Congratulations! By adding an endpoint for Amazon SNS to a VPC, you were able to publish a message to a topic from within the network that's managed by the VPC. The message was published privately without being exposed to the public internet.

Step 7: Clean up

Unless you want to retain the resources that you created, you can delete them now. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

First, delete your VPC endpoint using the Amazon VPC console. Then, delete the other resources that you created by deleting the stack in the AWS CloudFormation console. When you delete a stack, AWS CloudFormation removes the stack's resources from your AWS account.

To delete your VPC endpoint

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation menu on the left, choose **Endpoints**.
3. Select the endpoint that you created.
4. Choose **Actions**, and then choose **Delete Endpoint**.

5. In the **Delete Endpoint** window, choose **Yes, Delete**.

   The endpoint status changes to **deleting**. When the deletion completes, the endpoint is removed from the page.

To delete your AWS CloudFormation stack

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Select the stack **VPCE-Tutorial-Stack**.
3. Choose **Actions**, and then choose **Delete Stack**.
4. In the **Delete Stack** window, choose **Yes, Delete**.

   The stack status changes to **DELETE_IN_PROGRESS**. When the deletion completes, the stack is removed from the page.

Related resources

For more information, see the following resources.

- AWS Security Blog: Securing messages published to Amazon SNS with AWS PrivateLink
- What Is Amazon VPC?
- VPC Endpoints
- What Is Amazon EC2?
- AWS CloudFormation Concepts

Connect to Amazon SNS using Dual-stack (IPv4 and IPv6) endpoints

Dual-stack endpoints support both IPv4 and IPv6 traffic. When you make a request to a dual-stack endpoint, the endpoint URL resolves to an IPv4 or an IPv6 address. For more information on dual-stack and FIPS endpoints, see the SDK Reference guide.

Amazon SNS supports Regional dual-stack endpoints, which means that you must specify the AWS Region as part of the endpoint name. Dual-stack endpoint names use the following naming convention: `sns.`*`Region`*`.amazonaws.com`. For example, the dual-stack endpoint name for the `eu-west-1` Region is `sns.`*`eu-west-1`*`.amazonaws.com`.

For the full list of Amazon SNS endpoints, see the AWS General Reference.

Enhancing Amazon SNS security with Message Data Protection

- Message Data Protection is a feature in Amazon SNS used to define your own rules and policies to audit and control the content for data in motion, as opposed to data at rest.
- Message Data Protection provides governance, compliance, and auditing services for enterprise applications that are message-centric, so data ingress and egress can be controlled by the Amazon SNS topic owner, and content flows can be tracked and logged.
- You can write payload-based governance rules to stop unauthorized payload content from entering your message streams.
- You can grant different content-access permissions to individual subscribers, and audit the entire content-flow process.

Identity and access management in Amazon SNS

Access to Amazon SNS requires credentials that AWS can use to authenticate your requests. These credentials must have permissions to access AWS resources, such an Amazon SNS topics and messages. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon SNS to help secure your resources by controlling access to them.

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon SNS resources. IAM is an AWS service that you can use with no additional charge.

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon SNS.

Service user – If you use the Amazon SNS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon SNS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon SNS, see Troubleshooting Amazon Simple Notification Service identity and access.

Service administrator – If you're in charge of Amazon SNS resources at your company, you probably have full access to Amazon SNS. It's your job to determine which Amazon SNS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon SNS, see How Amazon SNS works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon SNS. To view example Amazon SNS identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Simple Notification Service.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the

principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

Access control

Amazon SNS has its own resource-based permissions system that uses policies written in the same language used for AWS Identity and Access Management (IAM) policies. This means that you can achieve similar things with Amazon SNS policies and IAM policies.

> **Note**
>
> It is important to understand that all AWS accounts can delegate their permissions to users under their accounts. Cross-account access allows you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, see Enabling Cross-Account Access in the *IAM User Guide*.

Amazon SNS access control use cases

You have a great deal of flexibility in how you grant or deny access to a resource. However, the typical use cases are fairly simple:

- You want to grant another AWS account a particular type of topic action (for example, Publish). For more information, see Grant AWS account access to a topic.
- You want to limit subscriptions to your topic to only the HTTPS protocol. For more information, see Limit subscriptions to HTTPS.
- You want to allow Amazon SNS to publish messages to your Amazon SQS queue. For more information, see Publish messages to an Amazon SQS queue.

Key Amazon SNS access policy 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

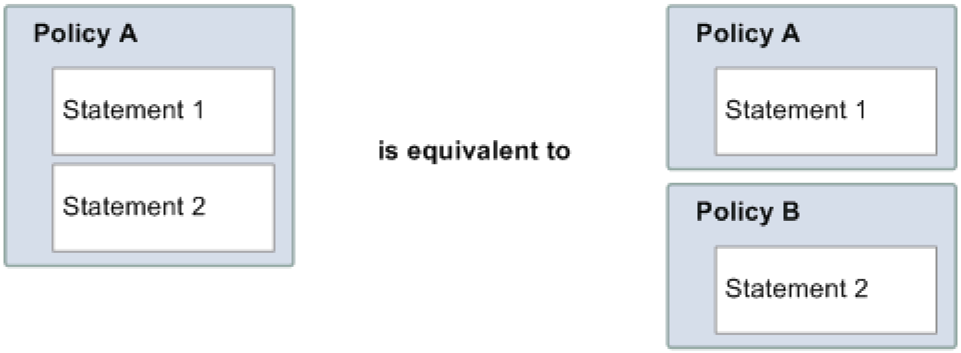
A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *publish* (B) to *TopicA* (C) as long as *she uses the HTTP protocol* (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 Topic A. As shown in the following figure, an equivalent scenario would be to have two policies, one that states that Jane can subscribe using the email protocol, and another that states that Bob cannot publish to Topic A.



Only ASCII characters are allowed in policy documents. You can utilize `aws:SourceAccount` and `aws:SourceOwner` to work around the scenario where you need to plug-in other AWS services' ARNs that contain non-ASCII characters. See the difference between [`aws:SourceAccount` versus `aws:SourceOwner`].

Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (that is, you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address.

Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. For example, Jane sends a request to Amazon SNS with `Action=Subscribe`. You can specify one or multiple actions in a policy.

Resource

The *resource* is the object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies."

Conditions and keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (for example, the request must arrive before a specific day)
- IP address (for example, the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called `aws:CurrentTime` and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The AWS service itself (for example, Amazon SQS or Amazon SNS) might also define service-specific keys. For more information, see Amazon SNS API permissions: Actions and resources reference.

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.

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.

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.

Default deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

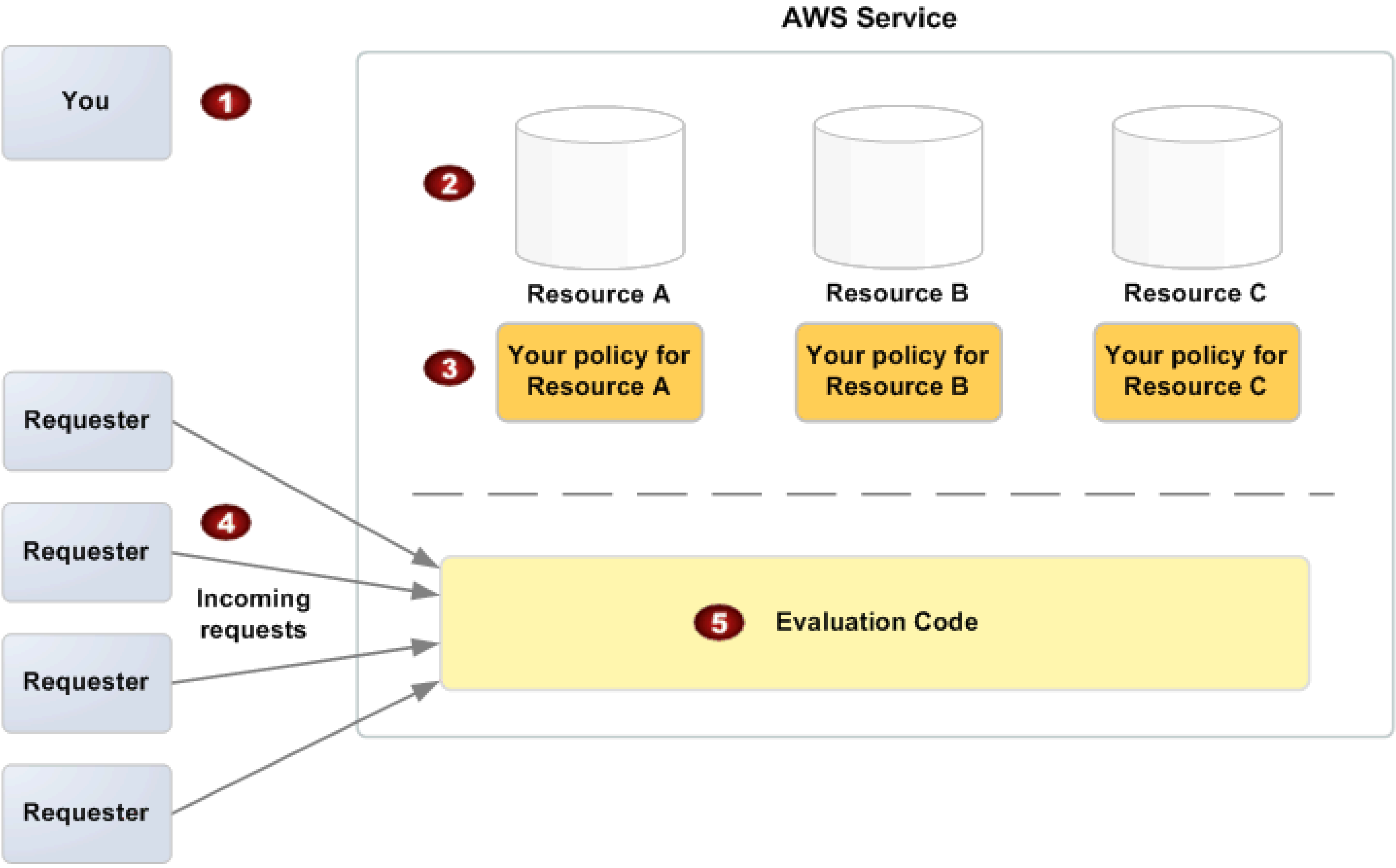
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

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.

Amazon SNS access control architecture overview

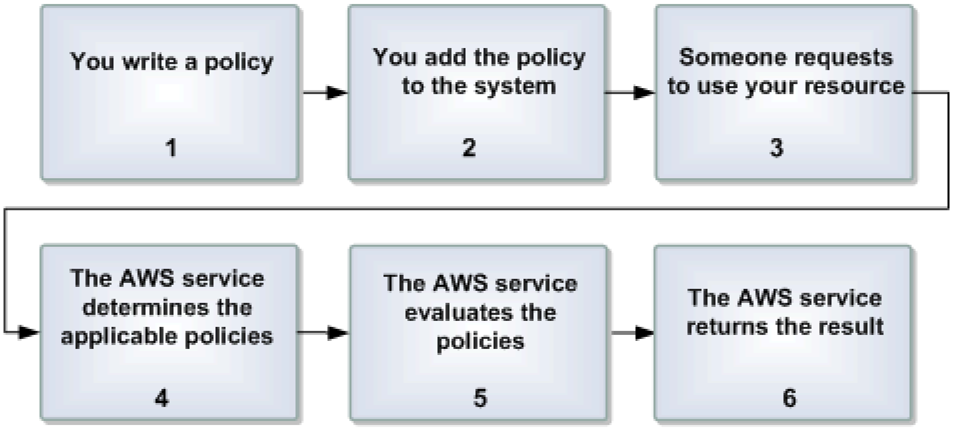
The following figure and table describe the main components that interact to provide access control for your resources.



1	You, the resource owner.
2	Your resources (contained within the AWS service; for example, Amazon SQS queues).
3	Your policies. Typically you have one policy per resource, although you could have multiple. The AWS service itself provides an API you use to upload and manage your policies.
4	Requesters and their incoming requests to the AWS service.
5	The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see Evaluation logic.

Using the Access Policy Language in Amazon SNS

The following figure and table describe the general process of how access control works with the access policy language.


You write a policy
1
You add the policy to the system
2
Someone requests to use your resource
3
The AWS service determines the applicable policies
4
The AWS service evaluates the policies
5
The AWS service returns the result
6

Process for using access control with the Access Policy Language

1	You write a policy for your resource. For example, you write a policy to specify permissions for your Amazon SNS topics.

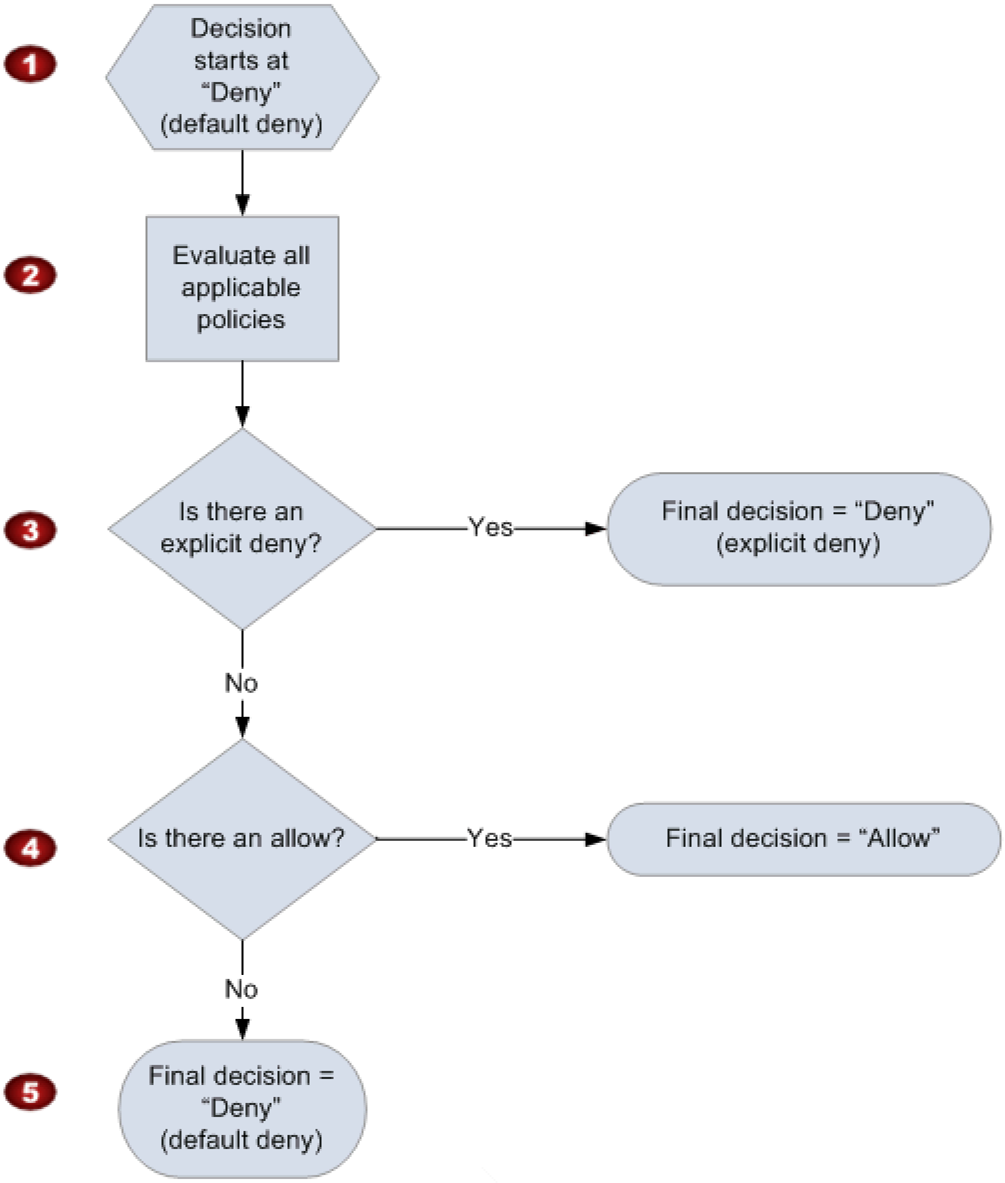
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SNS `SetTopicAttributes` action to upload a policy for a particular Amazon SNS topic.
3	Someone sends a request to use your resource. For example, a user sends a request to Amazon SNS to use one of your topics.
4	The AWS service determines which policies are applicable to the request. For example, Amazon SNS looks at all the available Amazon SNS policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies. For example, Amazon SNS evaluates the policies and determines if the requester is allowed to use your topic or not. For information about the decision logic, see Evaluation logic.
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

Evaluation logic

The goal at evaluation time is to decide whether a grant request should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.


1
Decision starts at "Deny" (default deny)
2
Evaluate all applicable policies
3
Is there an explicit deny?
Yes
Final decision = "Deny" (explicit deny)
No
4
Is there an allow?
Yes
Final decision = "Allow"
No
5
Final decision = "Deny" (default deny)

1	The decision starts with a default deny.
2	The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.
3	In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request.

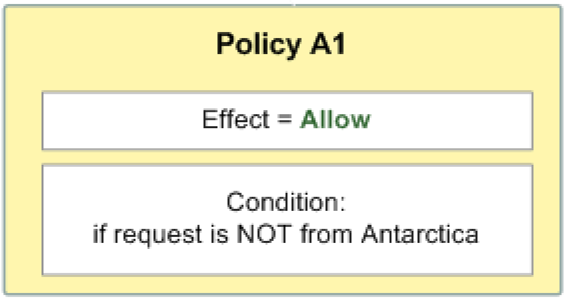
	If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit deny).
4	If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a *default deny* (for more information, see Default deny).

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.

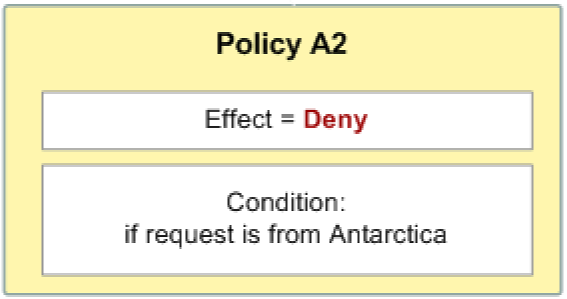
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met, 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.



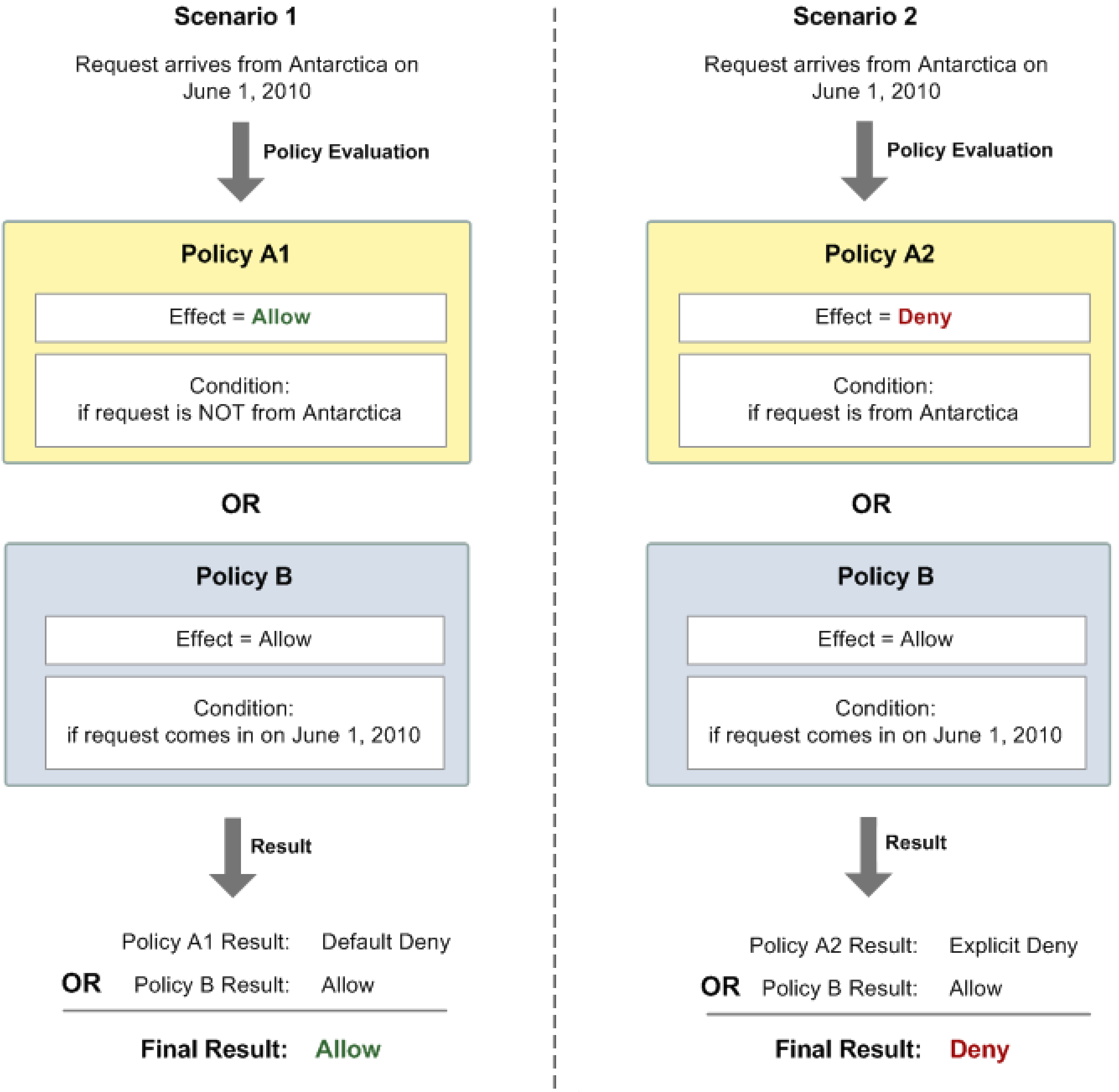
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

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.


Scenario 1
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A1
Effect = Allow
Condition: if request is NOT from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A1 Result: Default Deny
OR Policy B Result: Allow
Final Result: Allow
Scenario 2
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A2
Effect = Deny
Condition: if request is from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A2 Result: Explicit Deny
OR Policy B Result: Allow
Final Result: Deny

In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy A2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

Example cases for Amazon SNS access control

This section describes a few examples of typical use cases for access control.

Grant AWS account access to a topic

Let's say you have a topic in Amazon SNS, and you want to allow one or more AWS accounts to perform a specific action on that topic, such as publishing messages. You can accomplish this by using the Amazon SNS API action `AddPermission`.

The `AddPermission` action allows you to specify a topic, a list of AWS account IDs, a list of actions, and a label. Amazon SNS then automatically generates and adds a new policy statement to the topic's access control policy. You don’t need to write the policy statement yourself—Amazon SNS handles this for you. If you need to remove the policy later, you can do so by calling `RemovePermission` and providing the label you used when adding the permission.

For example, if you call `AddPermission` on the topic arn:aws:sns:us-east-2:444455556666:MyTopic, specify AWS account ID 1111-2222-3333, the `Publish` action, and the label `grant-1234-publish`, Amazon SNS will generate and insert the following policy statement into the topic’s access control policy:

```
{
  "Statement": [{
    "Sid": "grant-1234-publish",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Publish"],
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic"
  }]
}
```

After this statement is added, the AWS account 1111-2222-3333 will have permission to publish messages to the topic.

Additional information:

- **Custom policy management:** While `AddPermission` is convenient for granting permissions, it's often useful to manually manage the topic's access control policy for more complex scenarios, such as adding conditions or granting permissions to specific IAM roles or services. You can do this by using the `SetTopicAttributes` API to update the policy attribute directly.

- **Security best practices:** Be cautious when granting permissions to ensure that only trusted AWS accounts or entities have access to your Amazon SNS topics. Regularly review and audit the policies attached to your topics to maintain security.
- **Policy limits:** Keep in mind that there are limits to the size and complexity of Amazon SNS policies. If you need to add many permissions or complex conditions, ensure that your policy stays within these limits.

Limit subscriptions to HTTPS

To restrict the notification delivery protocol for your Amazon SNS topic to HTTPS, you must create a custom policy. The `AddPermission` action in Amazon SNS does not allow you to specify protocol restrictions when granting access to your topic. Therefore, you need to manually write a policy that enforces this restriction and then use the `SetTopicAttributes` action to apply the policy to your topic.

Here's how you can create a policy that limits subscriptions to HTTPS:

1. **Write the Policy.** The policy must specify the AWS account ID that you want to grant access to and enforce the condition that only HTTPS subscriptions are allowed. Below is an example policy that grants the AWS account ID 1111-2222-3333 permission to subscribe to the topic, but only if the protocol used is HTTPS.

```
{
  "Statement": [{
    "Sid": "Statement1",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Subscribe"],
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "StringEquals": {
        "sns:Protocol": "https"
      }
    }
  }]
}
```

2. **Apply the Policy.** Use the `SetTopicAttributes` action in the Amazon SNS API to apply this policy to your topic. Set the `Policy` attribute of the topic to the JSON policy you created.

```
snsClient.setTopicAttributes(SetTopicAttributesRequest.builder()
        .topicArn("arn:aws:sns:us-east-2:444455556666:MyTopic")
        .attributeName("Policy")
        .attributeValue(jsonPolicyString)  // The JSON policy as a string
        .build());
```

Additional information:

- **Customizing access control.** This approach allows you to enforce more granular access controls, such as restricting subscription protocols, which is not possible through the `AddPermission` action alone. Custom policies provide flexibility for scenarios requiring specific conditions, such as protocol enforcement or IP address restrictions.
- **Security best practices.** Limiting subscriptions to HTTPS enhances the security of your notifications by ensuring that data in transit is encrypted. Regularly review your topic policies to ensure they meet your security and compliance requirements.
- **Policy testing.** Before applying the policy in a production environment, test it in a development environment to ensure it behaves as expected. This helps prevent accidental access issues or unintended restrictions.

Publish messages to an Amazon SQS queue

To publish messages from your Amazon SNS topic to an Amazon SQS queue, you need to configure the correct permissions on the Amazon SQS queue. While both Amazon SNS and Amazon SQS use AWS's access control policy language, you must explicitly set a policy on the Amazon SQS queue to allow messages to be sent from the Amazon SNS topic.

You can achieve this by using the `SetQueueAttributes` action to apply a custom policy to the Amazon SQS queue. Unlike Amazon SNS, Amazon SQS does not support the `AddPermission` action for creating policy statements with conditions. Therefore, you must write the policy manually.

The following is an example of an Amazon SQS policy that grants Amazon SNS permission to send messages to your queue. Note that this policy is associated with the Amazon SQS queue, not the Amazon SNS topic. The actions specified are Amazon SQS actions, and the resource is the Amazon Resource Name (ARN) of the queue. You can retrieve the queue's ARN by using the `GetQueueAttributes` action.

```
{
  "Statement": [{
    "Sid": "Allow-SNS-SendMessage",
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": ["sqs:SendMessage"],
    "Resource": "arn:aws:sqs:us-east-2:444455556666:MyQueue",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-2:444455556666:MyTopic"
      }
    }
  }]
}
```

This policy uses the `aws:SourceArn` condition to restrict access to the SQS queue based on the source of the messages being sent. This ensures that only messages originating from the specified SNS topic (in this case, arn:aws:sns:us-east-2:444455556666:MyTopic) are allowed to be delivered to the queue.

Additional information:

- **Queue ARN.** Ensure you retrieve the correct ARN of your Amazon SQS queue using the `GetQueueAttributes` action. This ARN is essential for setting the correct permissions.
- **Security best practices.** When setting up policies, always follow the principle of least privilege. Grant only the necessary permissions to the Amazon SNS topic to interact with the Amazon SQS queue, and regularly review your policies to ensure they are up-to-date and secure
- **Default policies in Amazon SNS.** Amazon SNS doesn't automatically grant a default policy that allows other AWS services or accounts to access newly created topics. By default, Amazon SNS topics are created with no permissions, meaning they are private and only accessible to the account that created them. To enable access for other AWS services, accounts, or principals, you must explicitly define and attach an access policy to the topic. This aligns with the principle of least privilege, ensuring that no unintended access is granted by default.

- **Testing and validation.** After setting the policy, test the integration by publishing messages to the Amazon SNS topic and verifying that they are successfully delivered to the Amazon SQS queue. This helps confirm that the policy is correctly configured.

Allow Amazon S3 event notifications to publish to a topic

To allow an Amazon S3 bucket from another AWS account to publish event notifications to your Amazon SNS topic, you need to configure the topic's access policy accordingly. This involves writing a custom policy that grants permission to the Amazon S3 service from the specific AWS account and then applying this policy to your Amazon SNS topic.

Here’s how you can set it up:

1. **Write the policy.** The policy should grant the Amazon S3 service (s3.amazonaws.com) the necessary permissions to publish to your Amazon SNS topic. You will use the `SourceAccount` condition to ensure that only the specified AWS account, which owns the Amazon S3 bucket, can publish notifications to your topic.

   The following is an example policy:

```
{
  "Statement": [{
    "Effect": "Allow",
     "Principal": {
      "Service": "s3.amazonaws.com"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:111122223333:MyTopic",
    "Condition": {
      "StringEquals": {
        "AWS:SourceAccount": "444455556666"
      }
    }
  }]
}
```

   - **Topic owner** – 111122223333 is the AWS account ID that owns the Amazon SNS topic.
   - **Amazon S3 bucket owner** – 444455556666 is the AWS account ID that owns the Amazon S3 bucket sending notifications.

2. **Apply the Policy.** Use the `SetTopicAttributes` action to set this policy on your Amazon SNS topic. This will update the topic’s access control to include the permissions specified in your custom policy.

   ```
   snsClient.setTopicAttributes(SetTopicAttributesRequest.builder()
           .topicArn("arn:aws:sns:us-east-2:111122223333:MyTopic")
           .attributeName("Policy")
           .attributeValue(jsonPolicyString)  // The JSON policy as a string
           .build());
   ```

Additional information:

- **Using `SourceAccount` condition.** The `SourceAccount` condition ensures that only events originating from the specified AWS account (444455556666 in this case) can trigger the Amazon SNS topic. This is a security measure to prevent unauthorized accounts from sending notifications to your topic.
- **Other services supporting `SourceAccount`.** The `SourceAccount` condition is supported by the following services. It’s crucial to use this condition when you want to restrict access to your Amazon SNS topic based on the originating account.
  - Amazon API Gateway
  - Amazon CloudWatch
  - Amazon DevOps Guru
  - Amazon EventBridge
  - Amazon GameLift Servers
  - Amazon Pinpoint SMS and Voice API
  - Amazon RDS
  - Amazon Redshift
  - Amazon S3 Glacier
  - Amazon SES
  - Amazon Simple Storage Service
  - AWS CodeCommit
  - AWS Directory Service
  - AWS Lambda
  - AWS Systems Manager Incident Manager

- **Testing and validation.** After applying the policy, test the setup by triggering an event in the Amazon S3 bucket and confirming that it successfully publishes to your Amazon SNS topic. This will help ensure that your policy is correctly configured.
- **Security best practices.** Regularly review and audit your Amazon SNS topic policies to ensure they comply with your security requirements. Limiting access to only trusted accounts and services is essential for maintaining secure operations.

Allow Amazon SES to publish to a topic that is owned by another account

You can allow another AWS service to publish to a topic that is owned by another AWS account. Suppose that you signed into the 111122223333 account, opened Amazon SES, and created an email. To publish notifications about this email to a Amazon SNS topic that the 444455556666 account owns, you'd create a policy like the following. To do so, you need to provide information about the principal (the other service) and each resource's ownership. The `Resource` statement provides the topic ARN, which includes the account ID of the topic owner, 444455556666. The `"aws:SourceOwner": "111122223333"` statement specifies that your account owns the email.

JSON

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "ses.amazonaws.com"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
      "Condition": {
        "StringEquals": {
          "aws:SourceOwner": "111122223333"
        }
      }
    }
  ]
}
```

When publishing events to Amazon SNS, the following services support `aws:SourceOwner`:

- Amazon API Gateway
- Amazon CloudWatch
- Amazon DevOps Guru
- Amazon GameLift Servers
- Amazon Pinpoint SMS and Voice API
- Amazon RDS
- Amazon Redshift
- Amazon SES
- AWS CodeCommit
- AWS Directory Service
- AWS Lambda
- AWS Systems Manager Incident Manager

`aws:SourceAccount` versus `aws:SourceOwner`

> ⚠ **Important**
>
> `aws:SourceOwner` is deprecated and new services can integrate with Amazon SNS only through `aws:SourceArn` and `aws:SourceAccount`. Amazon SNS still maintains backward compatibility for existing services that are currently supporting `aws:SourceOwner`.

The `aws:SourceAccount` and `aws:SourceOwner` condition keys are each set by some AWS services when they publish to an Amazon SNS topic. When supported, the value will be the 12-digit AWS account ID on whose behalf the service is publishing data. Some services support one, and some support the other.

- See Allow Amazon S3 event notifications to publish to a topic for how Amazon S3 notifications use `aws:SourceAccount` and a list of AWS services that support that condition.
- See Allow Amazon SES to publish to a topic that is owned by another account for how Amazon SES uses `aws:SourceOwner` and a list of AWS services that support that condition.

Allow accounts in an organization in AWS Organizations to publish to a topic in a different account

The AWS Organizations service helps you to centrally manage billing, control access and security, and share resources across your AWS accounts.

You can find your organization ID in the Organizations console. For more information, see Viewing details of an organization from the management account.

In this example, any AWS account in organization `myOrgId` can publish to Amazon SNS topic `MyTopic` in account `444455556666`. The policy checks the organization ID value using the `aws:PrincipalOrgID` global condition key.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "myOrgId"
                }
            }
        }
    ]
}
```

Allow any CloudWatch alarm to publish to a topic in a different account

Use the following steps to invoke an Amazon SNS topic with a CloudWatch alarm across different AWS accounts. This example uses two accounts:

- **Account A** is used to create the CloudWatch alarm.
- **Account B** is used to create an SNS topic.

Create an SNS topic in account B

1. Sign in to the Amazon SNS console.
2. In the navigation pane, choose **Topics**, and then choose **Create topic**.
3. Choose **Standard** for the topic type, and then create a name for the topic.
4. Choose **Create topic**, and then copy the **ARN** of the topic.
5. In the navigation pane, choose **Subscriptions**, and then choose **Create subscription**.
6. Add the topic's ARN in the **Topic ARN** section, choose **Email** as the protocol, and then **enter an email address**.
7. Choose **Create subscription**, and then check your email to **confirm the subscription**.

Create a CloudWatch alarm in account A

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **Create alarms**.
3. If you haven't already created an alarm, create one now. Otherwise, select your **metric**, and then provide details for the threshold and comparison parameters.
4. From **Configure Actions**, under **Notifications**, choose **Use topic ARN to notify other accounts**, and then enter the **topic ARN** from Account B.
5. Create a name for the alarm, and then choose **Create alarm**.

Update the access policy of the SNS topic in account B

1. Sign in to the Amazon SNS console.
2. In the navigation pane, choose **Topics**, and then select the topic.
3. Choose **Edit**, and then add the following to the policy:

> **Note**
>
> Replace the example values in the policy below with your own.

JSON

```
{
    "Version": "2008-10-17",
    "Id": "__default_policy_ID",
    "Statement": [
        {
            "Sid": "__default_statement_ID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "SNS:GetTopicAttributes",
                "SNS:SetTopicAttributes",
                "SNS:AddPermission",
                "SNS:RemovePermission",
                "SNS:DeleteTopic",
                "SNS:Subscribe",
                "SNS:ListSubscriptionsByTopic",
                "SNS:Publish"
            ],
            "Resource": "arn:aws:cloudwatch:us-west-1:111122223333:alarm:",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cloudwatch:us-
west-1:111122223333:alarm:"
                }
            }
        }
    ]
}
```

Test the alarm

To test the alarm, either change the alarm threshold based on the metric data points, or manually change the alarm state. When you change the alarm threshold or alarm state, you receive an email notification.

Workaround for using a local Amazon SNS topic and forwarding messages

Use the following steps to enable cross-account Amazon SNS notifications for CloudWatch Alarms:

1. Create an **Amazon SNS topic** in the same account as the **CloudWatch alarm** (111122223333).
2. Subscribe a **Lambda function** or an Amazon EventBridge rule to that Amazon SNS topic.

3. The Lambda function or EventBridge rule can then publish the message to the Amazon SNS topic in the target account (444455556666).

Restrict publication to an Amazon SNS topic only from a specific VPC endpoint

In this case, the topic in account 444455556666 is allowed to publish only from the VPC endpoint with the ID `vpce-1ab2c34d`.

```
{
  "Statement": [{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpce": "vpce-1ab2c34d"
      }
    }
  }]
}
```

How Amazon SNS works with IAM

Before you use IAM to manage access to Amazon SNS, learn what IAM features are available to use with Amazon SNS.

IAM features you can use with Amazon Simple Notification Service

IAM feature	Amazon SNS support
Identity-based policies	Yes
Resource-based policies	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes

IAM feature	Amazon SNS support
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how Amazon SNS and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

AWS managed policies for Amazon Simple Notification Service

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AmazonSNSFullAccess

`AmazonSNSFullAccess` provides full access to Amazon SNS using the AWS Management Console. This policy also includes the following read and write actions for AWS End User Messaging SMS when called using Amazon SNS. You can attach this policy to your users, groups, or roles.

Permissions details

The following permissions apply only when using the Amazon SNS APIs:

- `sns:*` – Allows full permissions to perform any action related to Amazon SNS. This wildcard (*) means that the user can execute all possible Amazon SNS actions.
- `sms-voice:DescribeVerifiedDestinationNumbers` – Allows you to retrieve a list of phone numbers that have been verified for sending SMS messages within the AWS account.
- `sms-voice:CreateVerifiedDestinationNumber` – Allows you to verify a new phone number for use with SMS messaging services within AWS.
- `sms-voice:SendDestinationNumberVerificationCode` – Allows you to send a verification code to a phone number that is in the process of being verified for SMS messaging within AWS.
- `sms-voice:SendTextMessage` – Allows you to create a new text message and send it to a recipient's phone number. `SendTextMessage` only sends an SMS message to one recipient each time it's invoked.
- `sms-voice:DeleteVerifiedDestinationNumber` – Allows you to remove a phone number from the list of verified numbers within the AWS account
- `sms-voice:VerifyDestinationNumber` – Allows you to initiate and complete the verification process for a phone number to be used for SMS messaging services within AWS.
- `sms-voice:DescribeAccountAttributes` – Allows you to retrieve detailed information about the account-level attributes related to SMS messaging services within AWS.
- `sms-voice:DescribeSpendLimits` – Allows you to retrieve information about the spending limits associated with SMS messaging services within the AWS account
- `sms-voice:DescribePhoneNumbers` – Allows you to retrieve detailed information about the phone numbers associated with SMS messaging services within the AWS account
- `sms-voice:SetTextMessageSpendLimitOverride` – Allows you to set or override the spending limit for SMS text messaging within the AWS account
- `sms-voice:DescribeOptedOutNumbers` – Allows you to retrieve a list of phone numbers that have opted out of receiving SMS messages from your AWS account.

- `sms-voice:DeleteOptedOutNumber` – Allows you to remove a phone number from the list of opted-out numbers within the AWS account

`AmazonSNSFullAccess` example policy

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SNSFullAccess",
            "Effect": "Allow",
            "Action": "sns:*",
            "Resource": "*"
        },
        {
            "Sid": "SMSAccessViaSNS",
            "Effect": "Allow",
            "Action": [
                "sms-voice:DescribeVerifiedDestinationNumbers",
                "sms-voice:CreateVerifiedDestinationNumber",
                "sms-voice:SendDestinationNumberVerificationCode",
                "sms-voice:SendTextMessage",
                "sms-voice:DeleteVerifiedDestinationNumber",
                "sms-voice:VerifyDestinationNumber",
                "sms-voice:DescribeAccountAttributes",
                "sms-voice:DescribeSpendLimits",
                "sms-voice:DescribePhoneNumbers",
                "sms-voice:SetTextMessageSpendLimitOverride",
                "sms-voice:DescribeOptedOutNumbers",
                "sms-voice:DeleteOptedOutNumber"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaLast": "sns.amazonaws.com"
                }
            }
        }
    ]
}
```

To view the permissions for this policy, see AmazonSNSFullAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AmazonSNSReadOnlyAccess

`AmazonSNSReadOnlyAccess` provides read-only access to Amazon SNS using the AWS Management Console. This policy also includes the following read-only actions for AWS End User Messaging SMS when called using Amazon SNS. You can attach this policy to your users, groups, and roles.

Permissions details

The following permissions apply only when using the Amazon SNS APIs:

- `sns:GetTopicAttributes` – Allows you to retrieve the attributes of an Amazon SNS topic. This includes information such as the topic's ARN (Amazon Resource Name), the list of subscribers, delivery policies, access control policies, and any other metadata associated with the topic.
- `sns:List*` – Allows you to perform any operation that begins with `List` for Amazon SNS resources. This includes permissions to list various elements related to Amazon SNS, such as:
  - `sns:ListTopics` – Allows you to retrieve a list of all Amazon SNS topics in the AWS account.
  - `sns:ListSubscriptions` – Allows you to retrieve a list of all subscriptions to Amazon SNS topics.
  - `sns:ListSubscriptionsByTopic` – Allows you to list all subscriptions for a specific Amazon SNS topic.
  - `sns:ListPlatformApplications` – Allows you to list all platform applications that are created for mobile push notifications.
  - `sns:ListEndpointsByPlatformApplication` – Allows you to list all endpoints associated with a platform application.
- `sns:CheckIfPhoneNumberIsOptedOut` – Allows you to check whether a specific phone number has opted out of receiving SMS messages through Amazon SNS.
- `sns:GetEndpointAttributes` – Allows you to retrieve the attributes of an endpoint associated with an Amazon SNS platform application. This could include attributes such as

the endpoint's enabled status, custom user data, and any other metadata associated with the endpoint.

- `sns:GetDataProtectionPolicy` – Allows you to retrieve the data protection policy associated with an Amazon SNS topic.
- `sns:GetPlatformApplicationAttributes` – Allows you to retrieve the attributes of an Amazon SNS platform application. Platform applications are used in Amazon SNS to send push notifications to mobile devices through services such as Apple Push Notification Service (APNS) or Firebase Cloud Messaging (FCM).
- `sns:GetSMSAttributes` – Allows you to retrieve the default SMS settings for the AWS account.
- `sns:GetSMSSandboxAccountStatus` – Allows you to retrieve the current status of the SMS sandbox for your AWS account.
- `sns:GetSubscriptionAttributes` – Allows you to retrieve the attributes of a specific subscription to an Amazon SNS topic.
- `sms-voice:DescribeVerifiedDestinationNumbers` – Allows you to view or retrieve a list of phone numbers that have been verified for sending SMS messages within the AWS account
- `sms-voice:DescribeAccountAttributes` – Allows you to view or retrieve information about the account-level attributes related to SMS messaging services within AWS.
- `sms-voice:DescribeSpendLimits` – Allows you to view or retrieve information about the spending limits associated with SMS messaging services within your AWS account
- `sms-voice:DescribePhoneNumbers` – Allows you to view or retrieve information about the phone numbers that are used for SMS messaging services within the AWS account
- `sms-voice:DescribeOptedOutNumbers` – Allows you to view or retrieve a list of phone numbers that have opted out of receiving SMS messages from your AWS account

`AmazonSNSReadOnlyAccess` example policy

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SNSReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "sns:GetTopicAttributes",
                "sns:List*",
                "sns:CheckIfPhoneNumberIsOptedOut",
                "sns:GetEndpointAttributes",
                "sns:GetDataProtectionPolicy",
                "sns:GetPlatformApplicationAttributes",
                "sns:GetSMSAttributes",
                "sns:GetSMSSandboxAccountStatus",
                "sns:GetSubscriptionAttributes"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SMSAccessViaSNS",
            "Effect": "Allow",
            "Action": [
                "sms-voice:DescribeVerifiedDestinationNumbers",
                "sms-voice:DescribeAccountAttributes",
                "sms-voice:DescribeSpendLimits",
                "sms-voice:DescribePhoneNumbers",
                "sms-voice:DescribeOptedOutNumbers"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaLast": "sns.amazonaws.com"
                }
            }
        }
    ]
}
```

To view the permissions for this policy, see AmazonSNSFullAccess in the *AWS Managed Policy Reference*.

Amazon SNS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon SNS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon SNS Document history page.

Change	Description	Date
AmazonSNSFullAccess – Update to an existing policy	Amazon SNS added new permissions to allow full access to Amazon SNS using the AWS Management Console.	09/24/2024
AmazonSNSReadOnlyAccess – Update to an existing policy	Amazon SNS added new permissions to allow read-only access to Amazon SNS using the AWS Management Console.	09/24/2024
Amazon SNS started tracking changes	Amazon SNS started tracking changes for its AWS managed policies.	08/27/2024

Policy actions for Amazon SNS

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon SNS actions, see Resources Defined by Amazon Simple Notification Service in the *Service Authorization Reference*.

Policy actions in Amazon SNS use the following prefix before the action:

```
sns
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "sns:action1",
      "sns:action2"
         ]
```

To view examples of Amazon SNS identity-based policies, see Identity-based policy examples for Amazon Simple Notification Service.

Policy resources for Amazon SNS

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon SNS resource types and their ARNs, see Actions Defined by Amazon Simple Notification Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Resources Defined by Amazon Simple Notification Service.

To view examples of Amazon SNS identity-based policies, see Identity-based policy examples for Amazon Simple Notification Service.

Policy condition keys for Amazon SNS

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon SNS condition keys, see Condition Keys for Amazon Simple Notification Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Resources Defined by Amazon Simple Notification Service.

To view examples of Amazon SNS identity-based policies, see Identity-based policy examples for Amazon Simple Notification Service.

ACLs in Amazon SNS

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon SNS

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon SNS

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon SNS

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon SNS

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

⚠ **Warning**

Changing the permissions for a service role might break Amazon SNS functionality. Edit service roles only when Amazon SNS provides guidance to do so.

Service-linked roles for Amazon SNS

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Simple Notification Service

By default, users and roles don't have permission to create or modify Amazon SNS resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon SNS, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Simple Notification Service in the *Service Authorization Reference*.

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon SNS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to

service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon SNS console

To access the Amazon Simple Notification Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon SNS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon SNS console, also attach the Amazon SNS `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs) –** RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Identity-based policies for Amazon SNS

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon SNS

To view examples of Amazon SNS identity-based policies, see Identity-based policy examples for Amazon Simple Notification Service.

Resource-based policies within Amazon SNS

Supports resource-based policies	Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Using identity-based policies with Amazon SNS

Amazon Simple Notification Service integrates with AWS Identity and Access Management (IAM) so that you can specify which Amazon SNS actions a user in your AWS account can perform with

Amazon SNS resources. You can specify a particular topic in the policy. For example, you could use variables when creating an IAM policy that grants certain users in your organization permission to use the `Publish` action with specific topics in your AWS account. For more information, see Policy Variables in the *Using IAM* guide.

> ⚠ **Important**
>
> Using Amazon SNS with IAM doesn't change how you use Amazon SNS. There are no changes to Amazon SNS actions, and no new Amazon SNS actions related to users and access control.

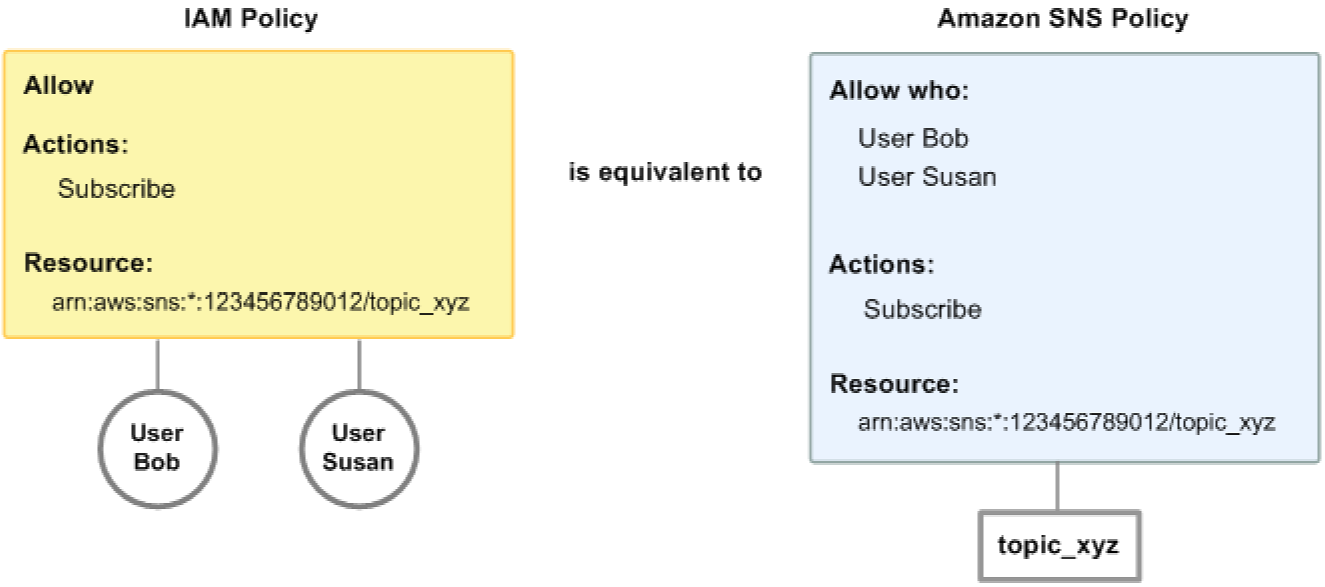
For examples of policies that cover Amazon SNS actions and resources, see Example policies for Amazon SNS.

IAM and Amazon SNS policies together

You use an IAM policy to restrict your users' access to Amazon SNS actions and topics. An IAM policy can restrict access only to users within your AWS account, not to other AWS accounts.

You use an Amazon SNS policy with a particular topic to restrict who can work with that topic (for example, who can publish messages to it, who can subscribe to it, etc.). Amazon SNS policies can grant access to other AWS accounts, or to users within your own AWS account.

To grant your users permissions for your Amazon SNS topics, you can use IAM policies, Amazon SNS policies, or both. For the most part, you can achieve the same results with either. For example, the following diagram shows an IAM policy and an Amazon SNS policy that are equivalent. The IAM policy allows the Amazon SNS `Subscribe` action for the topic called topic_xyz in your AWS account The IAM policy is attached to the users Bob and Susan (which means that Bob and Susan have the permissions stated in the policy). The Amazon SNS policy likewise grants Bob and Susan permission to access `Subscribe` for topic_xyz.



Note

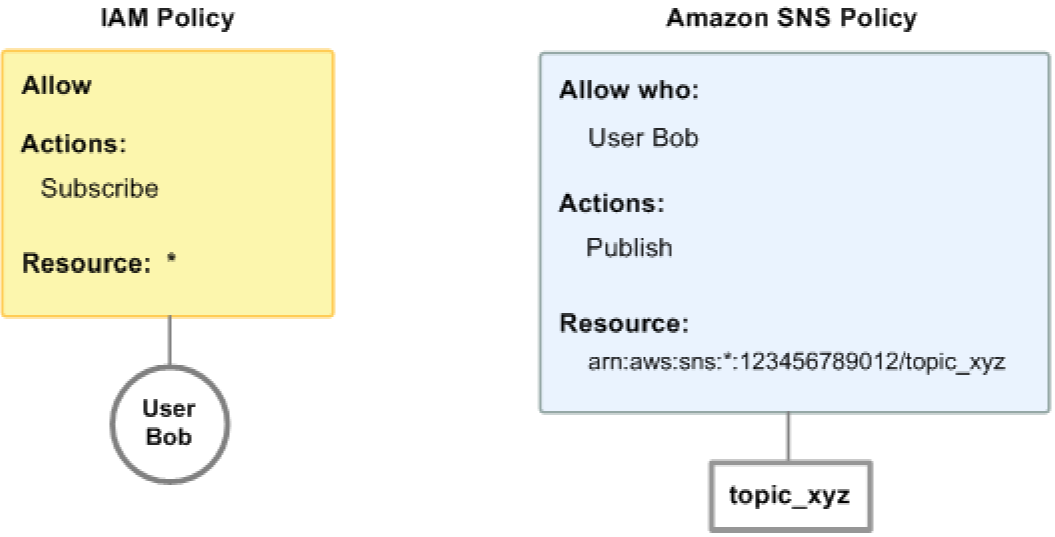
The preceding example shows simple policies with no conditions. You could specify a particular condition in either policy and get the same result.

There is one difference between AWS IAM and Amazon SNS policies: The Amazon SNS policy system lets you grant permission to other AWS accounts, whereas the IAM policy doesn't.

It's up to you how you use both of the systems together to manage your permissions, based on your needs. The following examples show how the two policy systems work together.

Example 1

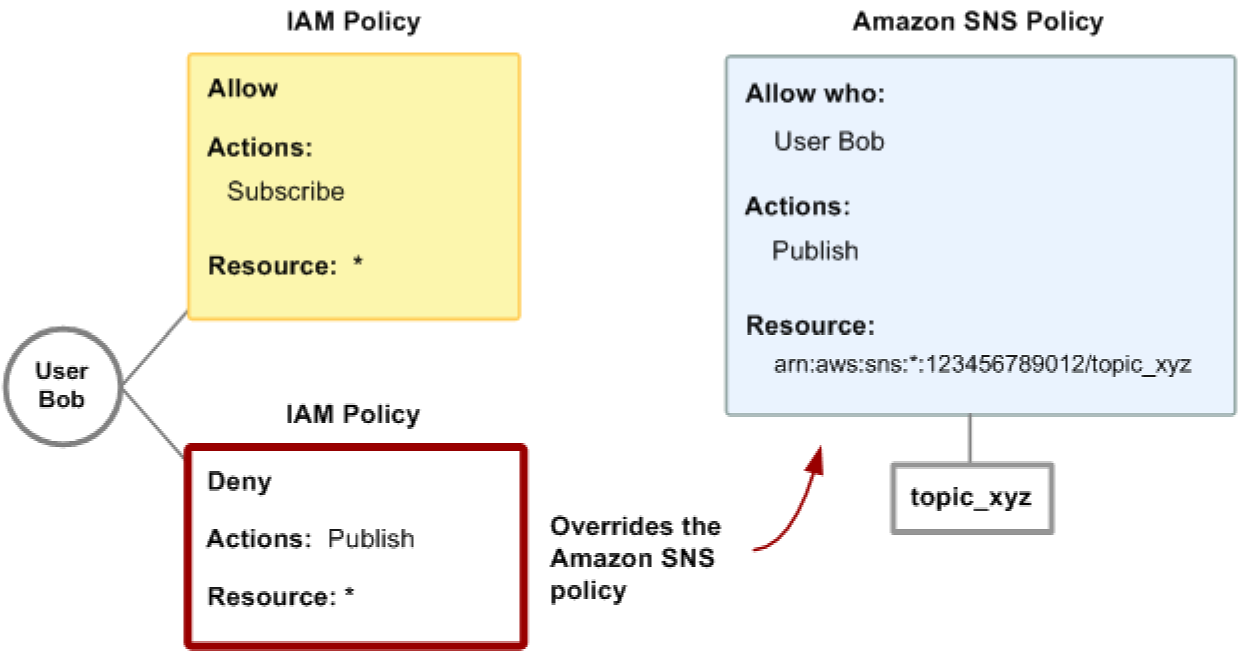
In this example, both an IAM policy and an Amazon SNS policy apply to Bob. The IAM policy grants him permission for `Subscribe` on any of the AWS account's topics, whereas the Amazon SNS policy grants him permission to use `Publish` on a specific topic (topic_xyz). The following diagram illustrates the concept.



If Bob were to send a request to subscribe to any topic in the AWS account, the IAM policy would allow the action. If Bob were to send a request to publish a message to topic_xyz, the Amazon SNS policy would allow the action.

Example 2

In this example, we build on example 1 (where Bob has two policies that apply to him). Let's say that Bob publishes messages to topic_xyz that he shouldn't have, so you want to entirely remove his ability to publish to topics. The easiest thing to do is to add an IAM policy that denies him access to the `Publish` action on all topics. This third policy overrides the Amazon SNS policy that originally gave him permission to publish to topic_xyz, because an explicit deny always overrides an allow (for more information about policy evaluation logic, see Evaluation logic). The following diagram illustrates the concept.



For examples of policies that cover Amazon SNS actions and resources, see Example policies for Amazon SNS.

Amazon SNS resource ARN format

For Amazon SNS, topics are the only resource type you can specify in a policy. The following is the Amazon Resource Name (ARN) format for topics.

```
arn:aws:sns:region:account_ID:topic_name
```

For more information about ARNs, go to ARNs in *IAM User Guide*.

Example

The following is an ARN for a topic named MyTopic in the us-east-2 region, belonging to AWS account 123456789012.

```
arn:aws:sns:us-east-2:123456789012:MyTopic
```

Example

If you had a topic named MyTopic in each of the different Regions that Amazon SNS supports, you could specify the topics with the following ARN.

```
arn:aws:sns:*:123456789012:MyTopic
```

You can use * and ? wildcards in the topic name. For example, the following could refer to all the topics created by Bob that he has prefixed with bob_.

```
arn:aws:sns:*:123456789012:bob_*
```

As a convenience to you, when you create a topic, Amazon SNS returns the topic's ARN in the response.

Amazon SNS API actions

In an IAM policy, you can specify any actions that Amazon SNS offers. However, the `ConfirmSubscription` and `Unsubscribe` actions do not require authentication, which means that even if you specify those actions in a policy, IAM won't restrict users' access to those actions.

Each action you specify in a policy must be prefixed with the lowercase string `sns:`. To specify all Amazon SNS actions, for example, you would use `sns:*`. For a list of the actions, go to the Amazon Simple Notification Service API Reference.

Amazon SNS policy keys

Amazon SNS implements the following AWS wide policy keys, plus some service-specific keys.

For a list of condition keys supported by each AWS service, see Actions, resources, and condition keys for AWS services in the *IAM User Guide*. For a list of condition keys that can be used in multiple AWS services, see AWS global condition context keys in the *IAM User Guide*.

Amazon SNS uses the following service-specific keys. Use these keys in policies that restrict access to `Subscribe` requests.

- **sns:endpoint—**The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see Example policies for Amazon SNS) to restrict access to specific endpoints (for example, *@yourcompany.com).
- **sns:protocol—**The `protocol` value from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see Example policies for Amazon SNS) to restrict publication to specific delivery protocols (for example, https).

Example policies for Amazon SNS

This section shows several simple policies for controlling user access to Amazon SNS.

> **Note**
>
> In the future, Amazon SNS might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

Example 1: Allow a group to create and manage topics

In this example, we create a policy that grants access to `CreateTopic`, `ListTopics`, `SetTopicAttributes`, and `DeleteTopic`.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:CreateTopic", "sns:ListTopics", "sns:SetTopicAttributes",
 "sns:DeleteTopic"],
    "Resource": "*"
  }]
}
```

Example 2: Allow the IT group to publish messages to a particular topic

In this example, we create a group for IT, and assign a policy that grants access to `Publish` on the specific topic of interest.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:*:123456789012:MyTopic"
  }]
}
```

Example 3: Give users in the AWS account ability to subscribe to topics

In this example, we create a policy that grants access to the `Subscribe`action, with string matching conditions for the `sns:Protocol` and `sns:Endpoint` policy keys.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:Subscribe"],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "sns:Endpoint": "*@example.com"
      },
      "StringEquals": {
        "sns:Protocol": "email"
      }
    }
  }]
}
```

Example 4: Allow a partner to publish messages to a particular topic

You can use an Amazon SNS policy or an IAM policy to allow a partner to publish to a specific topic. If your partner has an AWS account, it might be easier to use an Amazon SNS policy. However, anyone in the partner's company who possesses the AWS security credentials could publish messages to the topic. This example assumes that you want to limit access to a particular person (or application). To do this you need to treat the partner like a user within your own company, and use a IAM policy instead of an Amazon SNS policy.

For this example, we create a group called WidgetCo that represents the partner company; we create a user for the specific person (or application) at the partner company who needs access; and then we put the user in the group.

We then attach a policy that grants the group `Publish` access on the specific topic named *WidgetPartnerTopic*.

We also want to prevent the WidgetCo group from doing anything else with topics, so we add a statement that denies permission to any Amazon SNS actions other than `Publish` on any topics other than WidgetPartnerTopic. This is necessary only if there's a broad policy elsewhere in the system that grants users wide access to Amazon SNS.

```
{
  "Statement": [{
      "Effect": "Allow",
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
    },
    {
      "Effect": "Deny",
      "NotAction": "sns:Publish",
      "NotResource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
    }
  ]
}
```

Amazon SNS Policies for Email Endpoints

Amazon SNS normalizes email addresses in endpoint access policies by removing leading and trailing whitespace, converting all characters to lowercase, and normalizing special characters and escape sequences. Because of that, you must use lowercase email addresses in your policy definitions to ensure proper matching of subscription requests.

Example policy:

```
{
    "Condition": {
        "StringEquals": {
            "sns:Endpoint": "user.name@example.com"
        }
    }
}
```

The following email addresses match the above condition:

- `user.name@example.com`
- `USER.NAME@EXAMPLE.COM`
- `User.N\ame@Example.com`

Managing custom Amazon SNS IAM policies

Custom IAM policies allow you to specify permissions for individual IAM users, groups, or roles, granting or restricting access to specific AWS resources and actions. When managing Amazon SNS resources, custom IAM policies allow you to tailor access permissions according to your organization's security and operational requirements.

Use the following steps to manage custom IAM policies for Amazon SNS:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. From the navigation pane, choose **Policies**.
3. To create a new custom IAM policy, choose **Create policy** and choose **SNS**. To edit an existing policy, select the policy from the list and choose **Edit policy**.
4. In the policy editor, define the **permissions** for accessing Amazon SNS resources. You can specify **actions**, **resources**, and **conditions** based on your specific requirements.
5. To grant permissions for Amazon SNS actions, include relevant Amazon SNS actions such as `sns:Publish`, `sns:Subscribe`, and `sns:DeleteTopic` in your IAM policy. Define the ARN (Amazon Resource Name) of the Amazon SNS topics to which the permissions apply.
6. Specify the IAM **users**, **groups**, or **roles** to which the policy should be attached. You can attach the policy directly to IAM users or groups, or associate it with IAM roles used by AWS services or applications.
7. Review the IAM policy configuration to ensure it aligns with your access control requirements. Once verified, **save** the policy changes.
8. Attach the **custom IAM policy** to the relevant IAM users, groups, or roles within your AWS account. This grants them the permissions defined in the policy for managing Amazon SNS resources.

Using temporary security credentials with Amazon SNS

AWS Identity and Access Management (IAM) allows you to grant temporary security credentials to users and applications that need access to your AWS resources. These temporary security credentials are primarily used for IAM roles and federated access via industry-standard protocols such as SAML and OpenID Connect (OIDC).

To effectively manage access to AWS resources, it's essential to understand the following key concepts:

- **IAM Roles –** Roles are used to delegate access to AWS resources. Roles can be assumed by entities such as Amazon EC2 instances, Lambda functions, or users from other AWS accounts.
- **Federated Users –** These are users authenticated via external identity providers (IdPs) using SAML or OIDC. Federated access is recommended for human users, while IAM roles should be used for software applications.

- **Roles Anywhere** – For external applications requiring AWS access, you can use IAM Roles Anywhere to securely manage access without creating long-term credentials.

You can use temporary security credentials to make requests to Amazon SNS. The SDKs and API libraries compute the necessary signature using these credentials to authenticate your requests. Requests with expired credentials will be denied by Amazon SNS.

For more information on temporary security credentials, refer to Using IAM roles and Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.

Example HTTPS request example

The following example demonstrates how to authenticate an Amazon SNS request using temporary security credentials obtained from AWS Security Token Service (STS).

```
https://sns.us-east-2.amazonaws.com/
?Action=CreateTopic
&Name=My-Topic
&SignatureVersion=4
&SignatureMethod=AWS4-HMAC-SHA256
&Timestamp=2023-07-05T12:00:00Z
&X-Amz-Security-Token=SecurityTokenValue
&X-Amz-Date=20230705T120000Z
&X-Amz-Credential=<your-access-key-id>/20230705/us-east-2/sns/aws4_request
&X-Amz-SignedHeaders=host
&X-Amz-Signature=<signature-value>
```

Steps to authenticate the request

1. **Obtain Temporary Security Credentials** – Use AWS STS to assume a role or get federated user credentials. This will provide you with an access key ID, secret access key, and security token.
2. **Construct the Request** – Include the required parameters for your Amazon SNS action (for example, CreateTopic), and ensure you use HTTPS for secure communication.
3. **Sign the Request** – Use the AWS Signature Version 4 process to sign your request. This involves creating a canonical request, string-to-sign, and then calculating the signature. For more on AWS Signature Version 4, see Use Signature Version 4 signing in the *Amazon EBS User Guide*.
4. **Send the Request** – Include the X-Amz-Security-Token in your request header to pass the temporary security credentials to Amazon SNS.

Amazon SNS API permissions: Actions and resources reference

The following list grants information specific to the Amazon SNS implementation of access control:

- Each policy must cover only a single topic (when writing a policy, don't include statements that cover different topics)
- Each policy must have a unique policy `Id`
- Each statement in a policy must have a unique statement `sid`

Policy quotas

The following table lists the maximum quotas for a policy statement.

Name	Maximum quota
Bytes	30 kb
Statements	100
Principals	1 to 200 (0 is invalid.)
Resource	1 (0 is invalid. The value must match the ARN of the policy's topic.)

Valid Amazon SNS policy actions

Amazon SNS supports the actions shown in the following table.

Action	Description
sns:AddPermission	Grants permission to add permissions to the topic policy.
sns:DeleteTopic	Grants permission to delete a topic.
sns:GetDataProtectionPolicy	Grants permission to retrieve a topic's data protection policy.
sns:GetTopicAttributes	Grants permission to receive all of the topic attributes.

Action	Description
sns:ListSubscriptionsByTopic	Grants permission to retrieve all the subscriptions to a specific topic.
sns:ListTagsForResource	Grants permission to list all tags added to a specific topic.
sns:Publish	Grants permission to both publish and publish batch to a topic or endpoint. For more information, see Publish and PublishBatch in the Amazon Simple Notification Service API Reference.
sns:PutDataProtectionPolicy	Grants permission to set a topic's data protection policy.
sns:RemovePermission	Grants permission to remove any permissions in the topic policy.
sns:SetTopicAttributes	Grants permission to set a topic's attributes.
sns:Subscribe	Grants permission to subscribe to a topic.

Service-specific keys

Amazon SNS uses the following service-specific keys. You can use these in policies that restrict access to `Subscribe` requests.

- **sns:endpoint—**The URL, email address, or ARN from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see Example policies for Amazon SNS) to restrict access to specific endpoints (for example, *@example.com).
- **sns:protocol—**The `protocol` value from a `Subscribe` request or a previously confirmed subscription. Use with string conditions (see Example policies for Amazon SNS) to restrict publication to specific delivery protocols (for example, https).

> ⚠ **Important**
>
> When you use a policy to control access by sns:Endpoint, be aware that DNS issues might affect the endpoint's name resolution in the future.

Troubleshooting Amazon Simple Notification Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon SNS and IAM.

I am not authorized to perform an action in Amazon SNS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `sns:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sns:GetWidget on resource: my-example-widget
```

In this case, Mateo's policy must be updated to allow him to access the *`my-example-widget`* resource using the `sns:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon SNS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon SNS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon SNS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon SNS supports these features, see How Amazon SNS works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Logging and monitoring in Amazon SNS

Amazon SNS allows you to track and monitor messaging activity by logging API calls with CloudTrail and monitoring topics with CloudWatch. These tools help you gain insights into message delivery, troubleshoot issues, and ensure the health of your messaging workflows. This topic covers the following:

- Logging AWS SNS API calls using AWS CloudTrail. This logging enables you to track the actions performed on your Amazon SNS topics, such as topic creation, subscription management, and message publishing. By analyzing CloudTrail logs, you can identify who made specific API requests and when those requests were made, helping you audit and troubleshoot your Amazon SNS usage.
- Monitoring Amazon SNS topics using CloudWatch. CloudWatch provides metrics that allow you to observe the performance and health of your Amazon SNS topics in real time. Set up alarms based on these metrics, enabling you to respond promptly to any anomalies, such as delivery

failures or high message latency. This monitoring capability ensures that you can maintain the reliability of your SNS-based messaging system by proactively addressing potential issues.

Logging AWS SNS API calls using AWS CloudTrail

AWS SNS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for SNS as events. The calls captured include calls from the SNS console and code calls to the SNS API operations. Using the information collected by CloudTrail, you can determine the request that was made to SNS, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

SNS data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, reading or writing to an Amazon S3 object). These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn’t log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the SNS resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. For more information about how to log data events, see Logging data events with the AWS Management Console and Logging data events with the AWS Command Line Interface in the *AWS CloudTrail User Guide*.

The following table lists the SNS resource types for which you can log data events. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
SNS topic	`AWS::SNS::Topic`	- Publish - PublishBatch
SNS platform endpoint	`AWS::SNS::PlatformEndpoint`	- Publish For additional details, see `AdvancedEventSelector` in the AWS CloudTrail API Reference.

> **Note**
>
> SNS resource type `AWS::SNS::PhoneNumber` is not logged by CloudTrail.

You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector in the *AWS CloudTrail API Reference*.

For information about logging data events, see Logging data events with the AWS Management Console and Logging data events with the AWS CLI in the CloudTrail User Guide.

SNS management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

AWS SNS logs the following SNS control plane operations to CloudTrail as *management events*.

- `AddPermission`
- `CheckIfPhoneNumberIsOptedOut`

- ConfirmSubscription
- CreatePlatformApplication
- CreatePlatformEndpoint
- CreateSMSSandboxPhoneNumber
- CreateTopic
- DeleteEndpoint
- DeletePlatformApplication
- DeleteSMSSandboxPhoneNumber
- DeleteTopic
- GetDataProtectionPolicy
- GetEndpointAttributes
- GetPlatformApplicationAttributes
- GetSMSAttributes
- GetSMSSandboxAccountStatus
- GetSubscriptionAttributes
- GetTopicAttributes
- ListEndpointsByPlatformApplication
- ListOriginationNumbers
- ListPhoneNumbersOptedOut
- ListPlatformApplications
- ListSMSSandboxPhoneNumbers
- ListSubscriptions
- ListSubscriptionsByTopic
- ListTagsForResource
- ListTopics
- OptInPhoneNumber
- PutDataProtectionPolicy
- RemovePermission
- SetEndpointAttributes
- SetPlatformApplicationAttributes

- SetSMSAttributes
- SetSubscriptionAttributes
- SetTopicAttributes
- Subscribe
- TagResource
- Unsubscribe
- UntagResource
- VerifySMSSandboxPhoneNumber

> **Note**
>
> When you are not logged in to Amazon Web Services (unauthenticated mode) and either the ConfirmSubscription or Unsubscribe actions are invoked, then they will not be logged to CloudTrail. Such as, when you choose the provided link in an email notification to confirm a pending subscription to a topic, the ConfirmSubscription action is invoked in unauthenticated mode. In this example, the ConfirmSubscription action would not be logged to CloudTrail.

SNS event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail event that demonstrates the **ListTopics**, **CreateTopic**, and **DeleteTopic** actions.

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
      "eventTime": "2014-09-30T00:00:00Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "ListTopics",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "aws-sdk-java/unknown-version",
      "requestParameters": {
        "nextToken": "ABCDEF1234567890EXAMPLE=="
      },
      "responseElements": null,
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
      "eventTime": "2014-09-30T00:00:00Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "CreateTopic",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "aws-sdk-java/unknown-version",
      "requestParameters": {
        "name": "hello"
      },
      "responseElements": {
        "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
      },
      "requestID": "example7-5cd3-5323-8a00-f1889011fee9",
      "eventID": "examplec-4f2f-4625-8378-130ac89660b1",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
      "eventTime": "2014-09-30T00:00:00Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "DeleteTopic",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "aws-sdk-java/unknown-version",
      "requestParameters": {
        "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
      },
      "responseElements": null,
      "requestID": "example5-4faa-51d5-aab2-803a8294388d",
      "eventID": "example8-6443-4b4d-abfd-1b867280d964",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```

The following example shows a CloudTrail event that demonstrates the **`Publish`** action.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "ExampleUser"
      },
      "attributes": {
        "creationDate": "2023-08-21T16:44:05Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-08-21T16:48:37Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "Publish",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic",
    "message": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "subject": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageStructure": "json",
    "messageAttributes": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "responseElements": {
    "messageId": "0787cd1e-d92b-521c-a8b4-90434e8ef840"
  },
  "requestID": "0a8ab208-11bf-5e01-bd2d-ef55861b545d",
  "eventID": "bb3496d4-5252-4660-9c28-3c6aebdb21c0",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SNS::Topic",
      "ARN": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sns.us-east-1.amazonaws.com"
  }
}
```

The following example shows a CloudTrail event that demonstrates the **`PublishBatch`** action.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "ExampleUser"
      },
      "attributes": {
        "creationDate": "2023-08-21T19:20:49Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-08-21T19:22:01Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "PublishBatch",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic",
    "publishBatchRequestEntries": [
      {
        "id": "1",
        "message": "HIDDEN_DUE_TO_SECURITY_REASONS"
      },
      {
        "id": "2",
        "message": "HIDDEN_DUE_TO_SECURITY_REASONS"
      }
    ]
  },
  "responseElements": {
    "successful": [
      {
        "id": "1",
        "messageId": "30d68101-a64a-5573-9e10-dc5c1dd3af2f"
      },
      {
        "id": "2",
        "messageId": "c0aa0c5c-561d-5455-b6c4-5101ed84de09"
      }
    ],
    "failed": []
  },
  "requestID": "e2cdf7f3-1b35-58ad-ac9e-aaaea0ace2f1",
  "eventID": "10da9a14-0154-4ab6-b3a5-1825b229a7ed",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SNS::Topic",
      "ARN": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sns.us-east-1.amazonaws.com"
  }
}
```

For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Monitoring Amazon SNS topics using CloudWatch

Amazon SNS and Amazon CloudWatch are integrated so you can collect, view, and analyze metrics for every active Amazon SNS notification. Once you have configured CloudWatch for Amazon SNS, you can gain better insight into the performance of your Amazon SNS topics, push notifications, and SMS deliveries. For example, you can set an alarm to send you an email notification if a specified threshold is met for an Amazon SNS metric, such as `NumberOfNotificationsFailed`. For a list of all the metrics that Amazon SNS sends to CloudWatch, see Amazon SNS metrics. For more information about Amazon SNS push notifications, see Sending mobile push notifications with Amazon SNS.

> **Note**
>
> The metrics you configure with CloudWatch for your Amazon SNS topics are automatically collected and pushed to CloudWatch at *1-minute* intervals. These metrics are gathered on all topics that meet the CloudWatch guidelines for being active. A topic is considered active by CloudWatch for up to six hours from the last activity (that is, any API call) on the topic. There is no charge for the Amazon SNS metrics reported in CloudWatch; they are provided as part of the Amazon SNS service.

View CloudWatch metrics for Amazon SNS

You can monitor metrics for Amazon SNS using the CloudWatch console, CloudWatch's own command line interface (CLI), or programmatically using the CloudWatch API. The following procedures show you how to access the metrics using the AWS Management Console.

To view metrics using the CloudWatch console

1. Sign in to the CloudWatch console.
2. On the navigation panel, choose **Metrics**.
3. On the **All metrics** tab, choose **SNS**, and then choose one of the following dimensions:

   - **Country, SMS Type**
   - **PhoneNumber**

- **Topic Metrics**
- **Metrics with no dimensions**

4. To view more detail, choose a specific item. For example, if you choose **Topic Metrics** and then choose **NumberOfMessagesPublished**, the average number of published Amazon SNS messages for a 1-minute period throughout the time range of 6 hours is displayed.
5. To view Amazon SNS usage metrics, on the **All metrics** tab, choose **Usage**, and select the **target Amazon SNS usage metric** (for example, `NumberOfMessagesPublishedPerAccount`).

Set CloudWatch alarms for Amazon SNS metrics

CloudWatch also allows you to set alarms when a threshold is met for a metric. For example, you could set an alarm for the metric, **NumberOfNotificationsFailed**, so that when your specified threshold number is met within the sampling period, then an email notification would be sent to inform you of the event.

To set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Alarms**, and then choose the **Create Alarm** button. This launches the **Create Alarm** wizard.
3. Scroll through the Amazon SNS metrics to locate the metric you want to place an alarm on. Select the metric to create an alarm on and choose **Continue**.
4. Fill in the **Name**, **Description**, **Threshold**, and **Time** values for the metric, and then choose **Continue**.
5. Choose **Alarm** as the alarm state. If you want CloudWatch to send you an email when the alarm state is reached, choose either an existing Amazon SNS topic or choose **Create New Email Topic**. If you choose **Create New Email Topic**, you can set the name and email addresses for a new topic. This list will be saved and appear in the drop-down box for future alarms. Choose **Continue**.

> **Note**
>
> If you use **Create New Email Topic** to create a new Amazon SNS topic, the email addresses must be verified before they will receive notifications. Emails are sent only

when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they will not receive a notification.

6. At this point, the **Create Alarm** wizard gives you a chance to review the alarm you're about to create. If you need to make any changes, you can use the **Edit** links on the right. Once you are satisfied, choose **Create Alarm**.

For more information about using CloudWatch and alarms, see the CloudWatch Documentation.

Amazon SNS metrics

Amazon SNS sends the following metrics to CloudWatch.

Namespace	Metric	Description
AWS/SNS	`NumberOfMessagesPublished`	The number of messages published to your Amazon SNS topics. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum
AWS/SNS	`NumberOfNotificationsDelivered`	The number of messages successfully delivered from your Amazon SNS topics to subscribing endpoints. For a delivery attempt to succeed, the endpoint's subscription must accept the message. A subscription accepts a message if a.) it lacks a filter policy or b.) its filter policy includes attributes that match those assigned to the message.

Namespace	Metric	Description
		If the subscription rejects the message, the delivery attempt isn't counted for this metric. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum

Namespace	Metric	Description
AWS/SNS	`NumberOfNotificationsFailed`	The number of messages that Amazon SNS failed to deliver. For Amazon SQS, email, SMS, or mobile push endpoints, the metric increments by 1 when Amazon SNS stops attempting message deliveries. For HTTP or HTTPS endpoints, the metric includes every failed delivery attempt, including retries that follow the initial attempt. For all other endpoints, the count increases by 1 when the message fails to deliver (regardless of the number of attempts). This metric does not include messages that were rejected by subscription filter policies. You can control the number of retries for HTTP endpoints. For more information, see Amazon SNS message delivery retries. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average

Namespace	Metric	Description
AWS/SNS	`NumberOfNotificationsFilteredOut`	The number of messages that were rejected by subscription filter policies. A filter policy rejects a message when the message attributes don't match the policy attributes. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average
AWS/SNS	`NumberOfNotificationsFilteredOut-MessageAttributes`	The number of messages that were rejected by subscription filter policies for attribute-based filtering. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average

Namespace	Metric	Description
AWS/SNS	`NumberOfNotificationsFilteredOut-MessageBody`	The number of messages that were rejected by subscription filter policies for payload-based filtering . **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average
AWS/SNS	`NumberOfNotificationsFilteredOut-InvalidAttributes`	The number of messages that were rejected by subscript ion filter policies because the messages' attributes are invalid – for example, because the attribute JSON is incorrectly formatted. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average

Namespace	Metric	Description
AWS/SNS	`NumberOfNotificationsFilteredOut-NoMessageAttributes`	The number of messages that were rejected by subscription filter policies because the messages have no attributes. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average
AWS/SNS	`NumberOfNotificationsFilteredOut-InvalidMessageBody`	The number of messages that were rejected by subscription filter policies because the message body is invalid for filtering – for example, invalid JSON message body. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average

Namespace	Metric	Description
AWS/SNS	`NumberOfNotificationsRedrivenToDlq`	The number of messages that have been moved to a dead-letter queue. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average
AWS/SNS	`NumberOfNotificationsFailedToRedriveToDlq`	The number of messages that couldn't be moved to a dead-letter queue. **Units:** Count **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Sum, Average
AWS/SNS	`PublishSize`	The size of messages published. **Units:** Bytes **Valid dimensions:** Application, PhoneNumber, Platform, and TopicName **Valid statistics:** Minimum, Maximum, Average and Count

Namespace	Metric	Description
AWS/SNS	`SMSMonthToDateSpentUSD`	The charges you have accrued since the start of the current calendar month for sending SMS messages. You can set an alarm for this metric to know when your month-to-date charges are close to the monthly SMS spend quota for your account. When Amazon SNS determines that sending an SMS message would incur a cost that exceeds this quota, it stops publishing SMS messages within minutes. For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see Setting SMS messaging preferences in Amazon SNS. **Units:** USD **Valid dimensions:** None **Valid statistics:** Sum
AWS/SNS	`SMSSuccessRate`	The rate of successful SMS message deliveries. **Units:** Count **Valid dimensions:** PhoneNumber **Valid statistics:** Sum, Average, Data Samples

Dimensions for Amazon SNS metrics

Amazon Simple Notification Service sends the following dimensions to CloudWatch.

Dimension	Description
`Application`	Filters on application objects, which represent an app and device registered with one of the supported push notification services, such as APNs and FCM.
`Application,Platform`	Filters on application and platform objects, where the platform objects are for the supported push notification services, such as APNs and FCM.
`Country`	Filters on the destination country or region of an SMS message. The country or region is represented by its ISO 3166-1 alpha-2 code.
`PhoneNumber`	Filters on the phone number when you publish SMS directly to a phone number (without a topic).
`Platform`	Filters on platform objects for the push notification services, such as APNs and FCM.
`TopicName`	Filters on Amazon SNS topic names.
`SMSType`	Filters on the message type of SMS message. Can be *promotional* or *transactional*.

Amazon SNS usage metrics

Amazon Simple Notification Service sends the following usage metrics to CloudWatch.

Namespace	Service	Metric	Resource	Type	Description
AWS/Usage	SNS	`ResourceCount`	`NumberOfMessagesPu`	Resource	• The number of messages

Namespace	Service	Metric	Resource	Type	Description
			`blishedPerAccount`		published to your Amazon SNS topics across your AWS account. • Units: None • Valid Statistics: Sum
AWS/Usage	SNS	`ResourceCount`	`ApproximateNumberOfTopics`	Resource	• The approximate number of topics across your AWS account. • Units: None • Valid Statistics: Average, Minimum, Maximum, Sum

Namespace	Service	Metric	Resource	Type	Description
AWS/Usage	SNS	`ResourceCount`	`ApproximateNumberOfFilterPolicies`	Resource	- The approximate number of filter policies across your AWS account. - Units: None - Valid Statistics: Average, Minimum, Maximum, Sum
AWS/Usage	SNS	`ResourceCount`	`ApproximateNumberOfPendingSubscriptions`	Resource	- The approximate number of pending subscriptions across your AWS account. - Units: None - Valid Statistics: Average, Minimum, Maximum, Sum

Namespace	Service	Metric	Resource	Type	Description
AWS/Usage	SNS	`CallCount`	• `AddPermission` • `CheckIfPhoneNumberIsOptedOut` • `CreatePlatformApplication` • `CreatePlatformEndpoint` • `ConfirmSubscription` • `CreateSMSSandboxPhoneNumber` • `CreateTopic` • `DeleteEndpoint` • `DeletePlatformApplication` • `DeleteSMSSandboxPhoneNumber` • `DeleteTopic`	API	• The number of API calls for the selected Amazon SNS API across your AWS account. • Content is not allowed in trailing section. Units: None • Valid Statistics: Sum

Namespace	Service	Metric	Resource	Type	Description
			• GetEndpointAttributes • GetPlatformApplicationAttributes • GetSMSAttributes • GetSMSSandboxAccountStatus • GetSubscriptionAttributes • GetTopicAttributes • ListEndpointsByPlatformApplication • ListOriginationNumbers • ListPhoneNumbersOptedOut • ListPlatformApplications		

Namespace	Service	Metric	Resource	Type	Description
			• `ListSMSSandboxPhoneNumbers` • `ListSubscriptions` • `ListSubscriptionsByTopic` • `ListTagsForResource` • `ListTopics` • `OptInPhoneNumber` • `RemovePermission` • `SetEndpointAttributes` • `SetPlatformApplicationAttributes` • `SetSMSAttributes` • `SetSubscriptionAttributes` • `SetTopicAttributes`		

Namespace	Service	Metric	Resource	Type	Description
			• `Subscribe` • `Unsubscribe` • `UntagResource` • `VerifySMSSandboxPhoneNumber`		

Compliance validation for Amazon SNS

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of

Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon SNS

Resilience in Amazon SNS is ensured through leveraging the AWS global infrastructure, which revolves around AWS Regions and Availability Zones. AWS Regions offer physically separated and isolated Availability Zones connected by low-latency, high-throughput, and highly redundant networking. This architecture allows for seamless failover between Availability Zones without interruption, making applications and databases inherently more fault tolerant and scalable compared to traditional data center infrastructures. By using Availability Zones, Amazon SNS subscribers benefit from enhanced availability and reliability, guaranteeing message delivery despite potential disruptions. For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Additionally, subscriptions to Amazon SNS topics can be configured with delivery retries and dead-letter queues, enabling automatic handling of transient failures and ensuring messages reliably reach their intended destinations.

Amazon SNS also supports message filtering and message attributes, which lets you tailor resilience strategies to their specific use cases, enhancing the overall robustness of your applications.

Infrastructure security in Amazon SNS

As a managed service, Amazon SNS is protected by the AWS global network security procedures found in the Best Practices for Security, Identity, & Compliance documentation.

Use AWS API actions to access Amazon SNS through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

You must sign requests using both an access key ID and a secret access key associated with an IAM principal. Alternatively, you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials for signing requests.

You can call these API actions from any network location, but Amazon SNS supports resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon SNS policies to control access from specific Amazon VPC endpoints or specific VPCs. This effectively isolates network access to a given Amazon SNS topic from only the specific VPC within the AWS network. For more information, see Restrict publication to an Amazon SNS topic only from a specific VPC endpoint.

Amazon SNS security best practices

AWS provides many security features for Amazon SNS. Review these security features in the context of your own security policy.

> **Note**
>
> The guidance for these security features applies to common use cases and implementations. We recommend that you review these best practices in the context of your specific use case, architecture, and threat model.

Preventative best practices

The following are preventative security best practices for Amazon SNS.

Topics

- Ensure topics aren't publicly accessible
- Implement least-privilege access
- Use IAM roles for applications and AWS services which require Amazon SNS access
- Implement server-side encryption
- Enforce encryption of data in transit
- Consider using VPC endpoints to access Amazon SNS
- Ensure subscriptions are not configured to deliver to raw http endpoints

Ensure topics aren't publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your Amazon SNS topic, you should ensure that your topic isn't publicly accessible (accessible by everyone in the world or by any authenticated AWS user).

- Avoid creating policies with `Principal` set to `""`.
- Avoid using a wildcard (*). Instead, name a specific user or users.

Implement least-privilege access

When you grant permissions, you decide who receives them, which topics the permissions are for, and specific API actions that you want to allow for these topics. Implementing the principle of least privilege is important to reducing security risks. It also helps to reduce the negative effect of errors or malicious intent.

Follow the standard security advice of granting least privilege. That is, grant only the permissions required to perform a specific task. You can implement least privilege by using a combination of security policies pertaining to user access.

Amazon SNS uses the publisher-subscriber model, requiring three types of user account access:

- **Administrators** – Access to creating, modifying, and deleting topics. Administrators also control topic policies.
- **Publishers** – Access to sending messages to topics.
- **Subscribers** – Access to subscribing to topics.

For more information, see the following sections:

- Identity and access management in Amazon SNS
- Amazon SNS API permissions: Actions and resources reference

Use IAM roles for applications and AWS services which require Amazon SNS access

For applications or AWS services, such as Amazon EC2, to access Amazon SNS topics, they must use valid AWS credentials in their AWS API requests. Because these credentials aren't rotated automatically, you shouldn't store AWS credentials directly in the application or EC2 instance.

You should use an IAM role to manage temporary credentials for applications or services that need to access Amazon SNS. When you use a role, you don't need to distribute long-term credentials (such as a username, password, and access keys) to an EC2 instance or AWS service, such as AWS Lambda. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources.

For more information, see IAM Roles and Common Scenarios for Roles: Users, Applications, and Services in the *IAM User Guide*.

Implement server-side encryption

To mitigate data leakage issues, use encryption at rest to encrypt your messages using a key stored in a different location from the location that stores your messages. Server-side encryption (SSE) provides data encryption at rest. Amazon SNS encrypts your data at the message level when it stores it, and decrypts the messages for you when you access them. SSE uses keys managed in AWS Key Management Service. When you authenticate your request and have access permissions, there is no difference between accessing encrypted and unencrypted topics.

For more information, see Securing Amazon SNS data with server-side encryption and Managing Amazon SNS encryption keys and costs.

Enforce encryption of data in transit

It's possible, but not recommended, to publish messages that are not encrypted during transit by using HTTP. However, when a topic is encrypted at rest using AWS KMS, it is required to use HTTPS for publishing messages to ensure encryption both at rest and in transit. While the topic does not automatically reject HTTP messages, using HTTPS is necessary to maintain the security standards.

AWS recommends that you use HTTPS instead of HTTP. When you use HTTPS, messages are automatically encrypted during transit, even if the SNS topic itself isn't encrypted. Without HTTPS, a network-based attacker can eavesdrop on network traffic or manipulate it using an attack such as man-in-the-middle.

To enforce only encrypted connections over HTTPS, add the `aws:SecureTransport` condition in the IAM policy that's attached to unencrypted SNS topics. This forces message publishers to use HTTPS instead of HTTP. You can use the following example policy as a guide:

JSON

```
{
    "Id": "ExamplePolicy",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPublishThroughSSLOnly",
            "Action": "SNS:Publish",
            "Effect": "Deny",
            "Resource": [
                "arn:aws:sns:us-east-1:111122223333:test-topic"
            ],
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            },
            "Principal": "*"
        }
    ]
}
```

Consider using VPC endpoints to access Amazon SNS

If you have topics that you must be able to interact with, but these topics must absolutely not be exposed to the internet, use VPC endpoints to limit topic access to only the hosts within a particular VPC. You can use topic policies to control access to topics from specific Amazon VPC endpoints or from specific VPCs.

Amazon SNS VPC endpoints provide two ways to control access to your messages:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint.
- You can control which VPCs or VPC endpoints have access to your topic using a topic policy.

For more information, see Creating the endpoint and Creating an Amazon VPC endpoint policy for Amazon SNS.

Ensure subscriptions are not configured to deliver to raw http endpoints

Avoid configuring subscriptions to deliver to a raw http endpoints. Always have subscriptions delivering to an endpoint domain name. For example, a subscription configured to deliver to an endpoint, `http://1.2.3.4/my-path`, should be changed to `http://my.domain.name/my-path`.

Troubleshooting Amazon SNS topics using AWS X-Ray

AWS X-Ray collects data about requests that your application serves, and lets you view and filter data to identify potential issues and opportunities for optimization. For any traced request to your application, you can see detailed information about the request, the response, and the calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

You can use X-Ray with Amazon SNS to trace and analyze the messages that travel through your application. You can use the AWS Management Console to view the map of connections between Amazon SNS and other services that your application uses. You can also use the console to view metrics such as average latency and failure rates. For more information, see Amazon SNS and AWS X-Ray in the *AWS X-Ray Developer Guide*.

Active tracing in Amazon SNS

Use AWS X-Ray to trace and analyze user requests as they pass through your Amazon SNS topics to Amazon Data Firehose, AWS Lambda, Amazon SQS, and HTTP/S endpoint subscriptions.

With X-Ray, you get an end-to-end view of each request, allowing you to:

- Identify what is calling your Amazon SNS topic and what services are downstream of its subscriptions.
- Analyze latencies, such as:
  - Time spent in the Amazon SNS topic before processing.
  - Delivery times for each subscribed endpoint.

⚠ **Important**

Amazon SNS topics with numerous subscriptions may reach a size limit and not be fully traced. For information about trace document size limits, see X-ray service quotas in AWS General Reference.

If you call an Amazon SNS API from a service that's already being traced, Amazon SNS passes the trace through, even if X-Ray tracing isn't enabled on the API.

Amazon SNS supports X-Ray tracing for both standard and FIFO topics. You can enable X-Ray for an Amazon SNS topic by using the Amazon SNS console, Amazon SNS `SetTopicAttributes` API, Amazon Simple Notification Service CLI Reference, or AWS CloudFormation.

To learn more about using Amazon SNS with X-Ray, see Amazon SNS and AWS X-Ray in the AWS X-Ray Developer Guide.

Active tracing permissions

When using the Amazon SNS console, Amazon SNS attempts to create the necessary permissions for the Amazon SNS topic to call X-Ray. The attempt can be rejected if you don't have sufficient permissions to use the Amazon SNS console. For more information, see Identity and access management in Amazon SNS and Example cases for Amazon SNS access control.

When using the CLI, you must manually configure the permissions. Those permissions are configured using resource policies. For more on using required permissions in X-Ray, see Amazon SNS and AWS X-Ray.

Enabling active tracing on an Amazon SNS topic using the AWS console

When active tracing is enabled on an Amazon SNS topic, it reads the trace ID, sends the data to the customer based on the trace ID, and propagates the trace ID to downstream services.

1. Sign in to the Amazon SNS console.
2. Choose a topic or create a new one. For more details on creating topics, see Creating an Amazon SNS topic.
3. On the **Create topic** page, in the **Details** section, choose a topic type: **FIFO** or **Standard**.

    a. Enter a **Name** for the topic.

    b. (Optional) Enter a **Display name** for the topic.
4. Expand **Active tracing**, and choose **Use active tracing**.

Once you've enabled X-Ray for your Amazon SNS topic, you can use the X-Ray service map to view the end-to-end traces and service maps for the topic.

Enabling active tracing on an Amazon SNS topic using the AWS SDK

The following code example shows how to enable active tracing on an Amazon SNS topic by using the AWS SDK for Java.

```
public static void enableActiveTracing(SnsClient snsClient, String topicArn) {

        try {

            SetTopicAttributesRequest request = SetTopicAttributesRequest.builder()
                .attributeName("TracingConfig")
                .attributeValue("Active")
                .topicArn(topicArn)
                .build();

            SetTopicAttributesResponse result = snsClient.setTopicAttributes(request);
            System.out.println("\n\nStatus was " +
 result.sdkHttpResponse().statusCode() + "\n\nTopic " + request.topicArn()
                + " updated " + request.attributeName() + " to " +
 request.attributeValue());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
```

Enabling active tracing on an Amazon SNS topic using the AWS CLI

The following code example shows how to enable active tracing on an Amazon SNS topic by using the AWS CLI.

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --attribute-name TracingConfig \
    --attribute-value Active
```

Enabling active tracing on an Amazon SNS topic using AWS CloudFormation

The following AWS CloudFormation stack shows how to enable active tracing on an Amazon SNS topic.

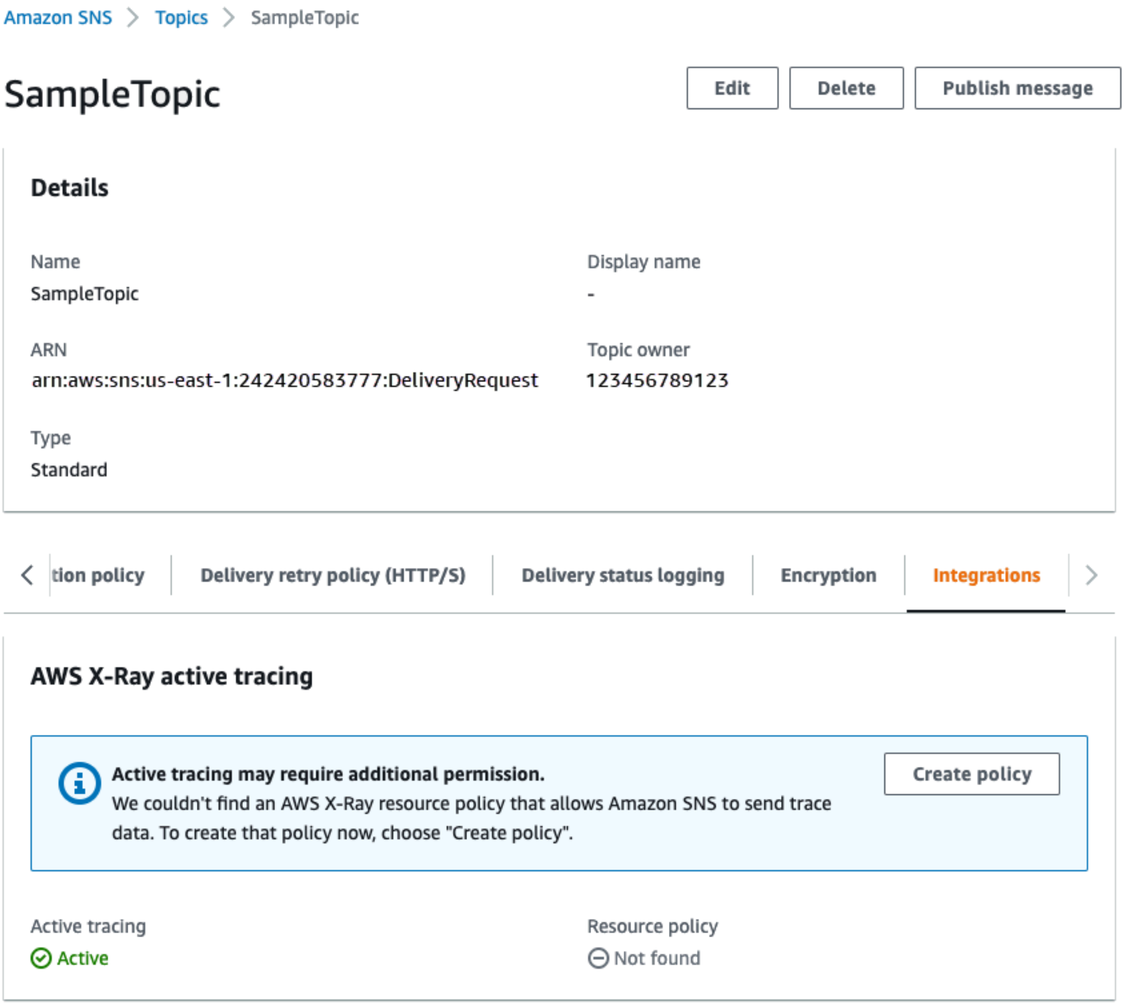
```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyTopicResource:
    Type: 'AWS::SNS::Topic'
    Properties:
      TopicName: 'MyTopic'
      TracingConfig: 'Active'
```

Verifying active tracing is enabled for your topic

You can use the Amazon SNS console to verify if active tracing is enabled for your topic, or when the resource policy has failed to be added.

1. Sign in to the Amazon SNS console.
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic.
4. Choose the **Integrations** tab.

   When active tracing is enabled, a green **Active** icon is displayed.
5. If you have enabled active tracing and you don't see that the resource policy has been added, choose **Create policy** to add the additional required permissions.



Details

Name
SampleTopic

Display name
-

ARN
arn:aws:sns:us-east-1:242420583777:DeliveryRequest

Topic owner
123456789123

Type
Standard

AWS X-Ray active tracing

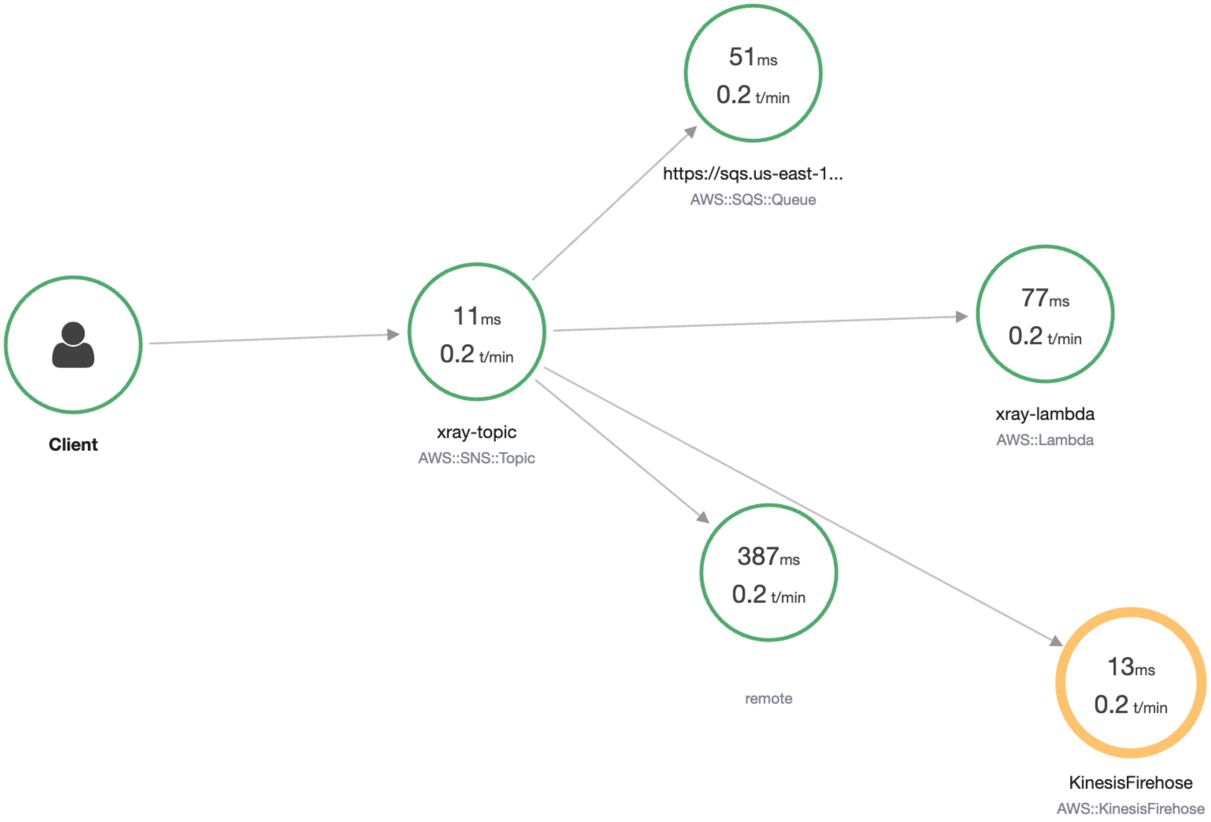
Active tracing may require additional permission.
We couldn't find an AWS X-Ray resource policy that allows Amazon SNS to send trace data. To create that policy now, choose "Create policy".

Active tracing
Active

Resource policy
Not found

Testing active tracing

1. Sign in to the Amazon SNS console.
2. Create an Amazon SNS topic. For details on how to do this, see To create a topic using the AWS Management Console.
3. Expand **Active tracing**, and choose **Use active tracing**.
4. Publish a message to the Amazon SNS topic. For details on how to do this, see To publish messages to Amazon SNS topics using the AWS Management Console.
5. Use the X-Ray service map to view the end-to-end traces and service maps for the topic.


51ms
0.2 t/min
https://sqs.us-east-1...
AWS::SQS::Queue
Client
11ms
0.2 t/min
xray-topic
AWS::SNS::Topic
77ms
0.2 t/min
xray-lambda
AWS::Lambda
387ms
0.2 t/min
remote
13ms
0.2 t/min
KinesisFirehose
AWS::KinesisFirehose

Amazon SNS documentation history

The following table describes recent changes to the *Amazon Simple Notification Service Developer Guide*.

Service features are sometimes rolled out incrementally to the AWS Regions where a service is available. We update this documentation for the first release only. We don't provide information about Region availability or announce subsequent Region rollouts. For information about Region availability of service features and to subscribe to notifications about updates, see What's New with AWS?.

Change	Description	Date
Added support for dual-stack (IPv4 and IPv6) endpoints	Amazon SNS now supports IPv6 for API requests, allowing clients to connect to public endpoints using IPv4, IPv6, or dual stack.	April 3, 2025
Amazon SNS added the `FifoThroughputScope` attribute for Amazon SNS FIFO topics	Amazon SNS support the FifoThroughputScope attribute, which specifies the throughput quota and deduplication behaviour to apply for the FIFO topic. Valid values are `Topic` or `MessageGroup` .	January 21, 2025
`AmazonSNSFullAccess` and `AmazonSNSReadOnlyAccess` managed policy updates	Amazon SNS added new permissions the AmazonSNSFullAccess and AmazonSNSReadOnlyAccess managed policies, which allows additional access to Amazon SNS via the AWS Management Console.	September 24, 2024

Amazon SNS integration with AWS End User Messaging SMS for delivery of SMS messages	Amazon SNS supports new features such as SMS resource management, two-way messaging, granular resource permissions, country block rules, and centralized billing for all AWS SMS messaging without making any changes to configurations or the global AWS SMS network used by Amazon SNS.	September 24, 2024
Canada West (Calgary) support for FIFO topics	Amazon SNS supports FIFO topic in Canada West (Calgary).	March 28, 2024
Amazon SNS SMS support in five new regions	Amazon SNS added SMS support to the following regions: Asia Pacific (Hyderabad), Asia Pacific (Melbourne), Middle East (UAE), Europe (Zurich). and Europe (Spain).	February 8, 2024
Firebase Cloud Messaging (FCM) HTTP v1 support	Amazon SNS supports FCM v1 credentials.	January 18, 2024
Amazon SNS SMS supported in Asia Pacific (Jakarta)	Amazon SNS supports SMS messaging in Asia Pacific (Jakarta).	December 14, 2023
AWS CloudFormation support for configuring `DeliveryStatusLogging` for Amazon SNS topics	AWS CloudFormation support is available for configuring `DeliveryStatusLogging` while creating or updating Amazon SNS topics.	December 7, 2023

New message filtering operators added	You can now use suffix matching, equals-ignore case, and OR operators when filtering Amazon SNS messages.	November 16, 2023
Support added for message archiving and replay	Topic owners can archive messages to a topic for up to 365 days. Topic subscribers can replay the archived messages back to a subscribed endpoint to recover messages due to a failure in a downstream application, or to replicate the state of an existing application.	October 26, 2023
Support added for subscribing a standard queue to a FIFO topic	You can subscribe an Amazon SQS FIFO queue or a standard queue to an Amazon SNS FIFO topic. Only Amazon SQS FIFO queues guarantee messages are received in order and with no duplicates.	September 14, 2023
SMS support added for Israel (Tel Aviv)	Amazon SNS SMS is now supported in the Israel (Tel Aviv) region.	August 28, 2023

Support for X-Ray active tracing added to FIFO topics	Previously only supported with Amazon SNS standard topics, AWS X-Ray now traces and analyzes user requests as they travel through your FIFO topics to your Amazon Data Firehose, AWS Lambda, Amazon SQS, and HTTP/S endpoint subscriptions.	May 31, 2023
Enhanced Content-Type header support	You can set the Content-Type header in the request policy to specify the media type of the notification.	March 23, 2023
Active tracing support added	AWS X-Ray traces and analyzes user requests as they travel through your Amazon SNS standard topics to your Amazon Data Firehose, AWS Lambda, Amazon SQS, and HTTP/S endpoint subscriptions.	February 8, 2023
Singapore Sender ID registration	Instructions added for registering Sender IDs in Singapore.	January 10, 2023
Payload-based message filtering	Payload-based filtering lets you filter messages based on the message payload and avoid the costs associated with processing unwanted data.	November 22, 2022

SHA256 hash algorithm added for Amazon SNS message signing	Support added for SHA256 hash algorithm when using Amazon SNS message signing.	September 15, 2022
Additional regions added to SMS messaging	Amazon SNS supports SMS messaging in the following regions: Africa (Cape Town), Asia Pacific (Osaka), Europe (Milan) and AWS GovCloud (US-East).	September 9, 2022
Message data protection support added	Message data protection safeguards the data that's published to your Amazon SNS topics by using data protection policies to audit and block the sensitive information that moves between applications or AWS services.	September 8, 2022
New registration process for toll-free numbers	Support added for sending for Amazon SNS messages using toll-free phone numbers (TFN) to United States recipients.	August 1, 2022

Support for Attribute-based access controls (ABAC)	Added support for attribute-based access control (ABAC) for API actions including `Publish` and `PublishBatch` . ABAC is an authorization strategy that defines access permissions based on tags which can be attached to IAM resources, such as IAM users and roles, and to AWS resources, like Amazon SNS topics, to simplify permission management.	January 10, 2022
Support for Apple token-based authentication for push notifications	You can authorize Amazon SNS to send push notifications to your iOS or macOS app by providing information that identifies you as the developer of the app.	October 28, 2021
New senders of SMS messages are placed in the SMS sandbox	The SMS sandbox exists to help prevent fraud and abuse, and to help protect your reputation as a sender. While your AWS account is in the SMS sandbox, you can send SMS messages only to verified destination phone numbers.	June 1, 2021

New senders of SMS messages are placed in the SMS sandbox	The SMS sandbox exists to help prevent fraud and abuse, and to help protect your reputation as a sender. While your AWS account is in the SMS sandbox, you can send SMS messages only to verified destination phone numbers.	June 1, 2021
New attributes for sending SMS messages to recipients in India	Two new attributes, **Entity ID** and **Template ID**, are now required for sending SMS messages to recipients in India.	April 22, 2021
Updates to message filtering operators	A new operator, `cidr`, is available for matching message source IP addresses and subnets. You can now also check for the absence of an attribute key, and use a prefix with the `anything-but` operator for attribute string matching.	April 7, 2021
Ending support for P2P long codes for US destinations	Effective June 1, 2021, US telecom providers no longer support using person-to -person (P2P) long codes for application-to-person (A2P) communications to US destinations. Instead, you can use short codes, toll-free numbers, or a new type of origination number called *10DLC*.	February 16, 2021

Support for 1-minute Amazon CloudWatch metrics	The 1-minute CloudWatch metric for Amazon SNS is now available in all AWS Regions.	January 28, 2021
Support for endpoints	You can subscribe Firehose delivery streams to SNS topics. This allows you to send notifications to archiving and analytics endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, Amazon OpenSearch Service (OpenSearch Service), and more.	January 12, 2021
Origination numbers are available	You can use origination numbers when sending text messages (SMS).	October 23, 2020
Support for Amazon SNS FIFO topics	To integrate distributed applications that require data consistency in near-real time, you can use Amazon SNS first-in, first-out (FIFO) topics with Amazon SQS FIFO queues.	October 22, 2020
The Amazon SNS Extended Client Library for Java is available	You can use this library to publish large Amazon SNS messages.	August 25, 2020
SSE is available in the China Regions	Server-side encryption (SSE) for Amazon SNS is available in the China Regions.	January 20, 2020

Support for using DLQs to capture undeliverable messages	To capture undeliverable messages, you can use an Amazon SQS dead-letter queue (DLQ) with an Amazon SNS subscription.	November 14, 2019
Support for specifying custom APNs header values	You can specify a custom APNs header value.	October 18, 2019
Support for the '`apns-push-type` ' header field for APNs	You can use the `apns-push-type` header field for mobile notifications sent through APNs.	September 10, 2019
Support for topic troubleshooting using AWS X-Ray	You can use X-Ray to troubleshoot messages passing through SNS topics.	July 24, 2019
Support for attribute key matching using the '`exists`' operator	To check whether an incoming message has an attribute whose key is listed in the filter policy, you can use the `exists` operator.	July 5, 2019
Support for anything-but matching of multiple numeric values	In addition to multiple strings, Amazon SNS allows anything-but matching of multiple numeric values.	July 5, 2019
Amazon SNS release notes are available as an RSS feed	Following the title on this page (**Documentation history**), choose **RSS**.	June 22, 2019