

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 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

AWS SDK for .NET

 **Note**

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

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>
    ///<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 { 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_snstopiclimitexcde.
```

```
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*.

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 Data Firehose](#)
- [Amazon SQS](#)

 **Note**

To subscribe to an [SNS FIFO topic](#), choose this option.

- [AWS Lambda](#)
 - [Platform application endpoint](#)
 - [SMS](#)
- For **Endpoint**, enter the endpoint value, such as an email address or the ARN of an Amazon SQS queue.
 - 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](#).
 - (Optional) For Firehose, Amazon SQS, HTTP/S endpoints, you can also enable raw message delivery. For more information, see [Amazon SNS raw message delivery](#).
 - (Optional) To configure a filter policy, expand the **Subscription filter policy** section. For more information, see [Amazon SNS subscription filter policies](#).
 - (Optional) To enable payload-based filtering, configure Filter Policy Scope to MessageBody. For more information, see [Amazon SNS subscription filter policy scope](#).
 - (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](#).
 - 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**. Messages in the same message group are delivered in the order that they are published.
 - c. For a FIFO topic, enter a **Message deduplication ID**. This ID is optional if you enabled the **Content-based message deduplication** setting for the topic.
 - d. (Optional) For [mobile push notifications](#), 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

AWS SDK for .NET

Note

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

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.SetString(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 { 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

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='String'  
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*.

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*.

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.amazonaws.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.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 to store large messages.
- **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** – The Boto3 Amazon S3 resource object used to store objects in Amazon S3. Use this if you want to control the Amazon S3 resource (for example, a custom Amazon S3 config or credentials). If not previously set on first use, the default is `boto3.resource("s3")`.

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.

- 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.
- 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
import sns_extended_client
from json import loads

s3_extended_payload_bucket = "extended-client-bucket-store"
TOPIC_NAME = "TOPIC-NAME"
QUEUE_NAME = "QUEUE-NAME"

# Create an helper to fetch message from S3
def get_msg_from_s3(body):
    json_msg = loads(body)
    s3_client = boto3.client("s3")
    s3_object = 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

# Create an helper to fetch and print message SQS queue and S3
def fetch_and_print_from_sqs(sqs, queue_url):
    """Handy Helper to fetch and print message from SQS queue and S3"""
    message = sqs.receive_message(
        QueueUrl=queue_url, MessageAttributeNames=["All"], MaxNumberOfMessages=1
    ).get("Messages")[0]
    message_body = message.get("Body")
    print("Published Message: {}".format(message_body))
    print("Message Stored in S3 Bucket is: {} \n".format(get_msg_from_s3(message_body)))

# Initialize the SNS client and create SNS Topic
sns_extended_client = boto3.client("sns", region_name="us-east-1")
create_topic_response = sns_extended_client.create_topic(Name=TOPIC_NAME)
demo_topic_arn = create_topic_response.get("TopicArn")
```

```
# Create and subscribe an SQS queue to the SNS client
sns = boto3.client("sns")
demo_queue_url = sns.create_queue(QueueName=QUEUE_NAME).get("QueueUrl")
demo_queue_arn = sns.get_queue_attributes(QueueUrl=demo_queue_url,
AttributeNames=["QueueArn"])[["Attributes"].get("QueueArn")]
# Set the RawMessageDelivery subscription attribute to TRUE
sns_extended_client.subscribe(TopicArn=demo_topic_arn, Protocol="sns",
Endpoint=demo_queue_arn, Attributes={"RawMessageDelivery":"true"})

sns_extended_client.large_payload_support = s3_extended_payload_bucket

# To store all messages content in S3, set always_through_s3 to True
# In the example, we set message size threshold as 32 bytes, adjust this threshold as
per your usecase
# Message will only be uploaded to S3 when its payload size exceeded threshold
sns_extended_client.message_size_threshold = 32
sns_extended_client.publish(
    TopicArn=demo_topic_arn,
    Message="This message should be published to S3 as it exceeds the
message_size_threshold limit",
)
# Print message stored in s3
fetch_and_print_from_sqs(sns, demo_queue_url)
```

Output

Published Message:

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

Message Stored in S3 Bucket is: This message should be published to S3 as it exceeds
the message_size_threshold limit

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

Push notification service	Reserved message attribute
	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
	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

Push notification service	Reserved message attribute
	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 KB).

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

How does message batching work?

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

Examples

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.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")
            .messageDuplicationId("duplicationId" + 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

AWS SDK for .NET

Note

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

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
  //   }
}
```