Advance DevOps Case Study

Case Study Topic: Automated Notifications Using SNS

Introduction

Case study overview:
The goal of my case study is to create an automated system where a Lambda function is triggered when a file is uploaded to an S3 bucket. This function will extract file details like name and size and send an email notification using Amazon SNS with those details.

The following concepts have been used to implement the task:

- AWS Lambda: A serverless compute service that runs code in response to events.
- Amazon S3: A scalable object storage service.
- Amazon SNS (Simple Notification Service): A fully managed messaging service for sending notifications.

Key feature:

The unique feature of this case study is the automation of email notifications based on real-time file uploads in an S3 bucket. This is achieved using AWS Lambda to handle events generated by S3 and SNS to send email notifications. No server management is required, making it a fully serverless solution.

Practical applications:

- Data Processing Pipelines: Automatically notifying teams when data files are uploaded to an S3 bucket for processing, ensuring that analysts or engineers can act quickly on fresh data.
- Backup Monitoring: Sending notifications when backup files are successfully uploaded, helping IT teams stay informed about the status of critical backups.
- User-Generated Content Moderation: In systems where users upload files (e.g., photos or documents), the system can notify moderators whenever a new file is uploaded for review.

Companies and organizations like Netflix, Airbnb, NASA use AWS Lambda, S3, and SNS for automated notifications.

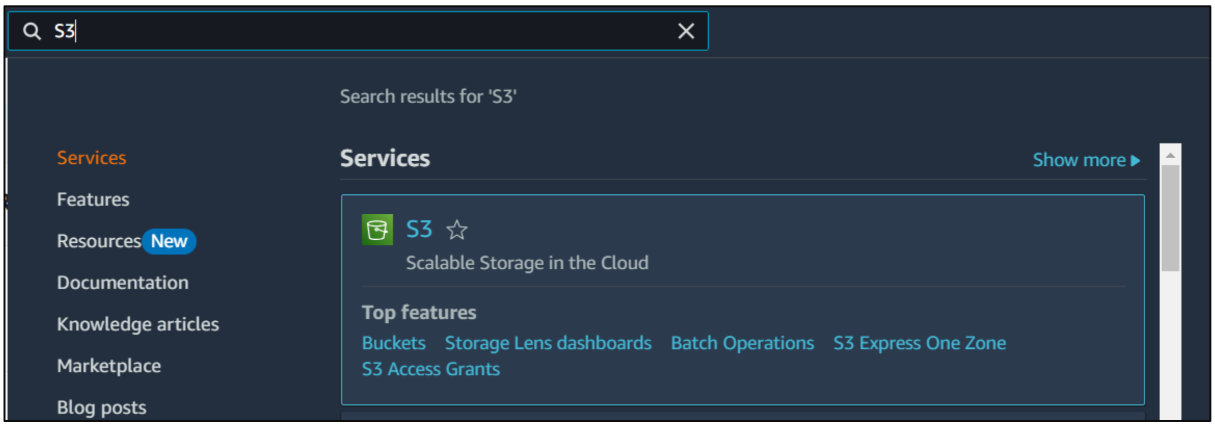
Step by step explanation

Overview of steps to be performed:

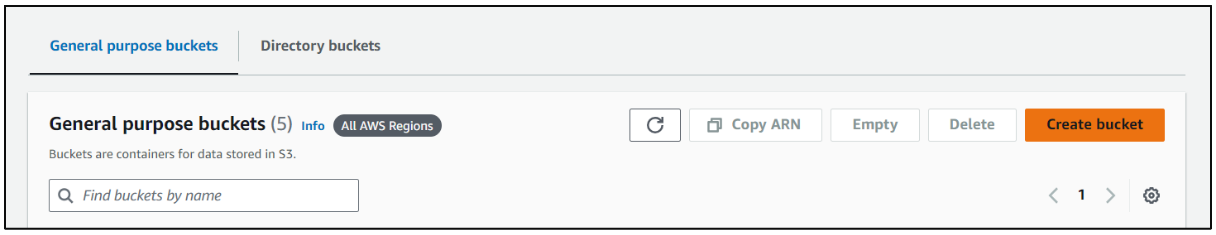
1. Create an S3 bucket
2. SNS topic and subscription configuration
3. Create lambda function
4. Create IAM role for lambda function
5. Write lambda function
6. Configuring event trigger
7. Testing the setup

Detailed steps with screenshots:

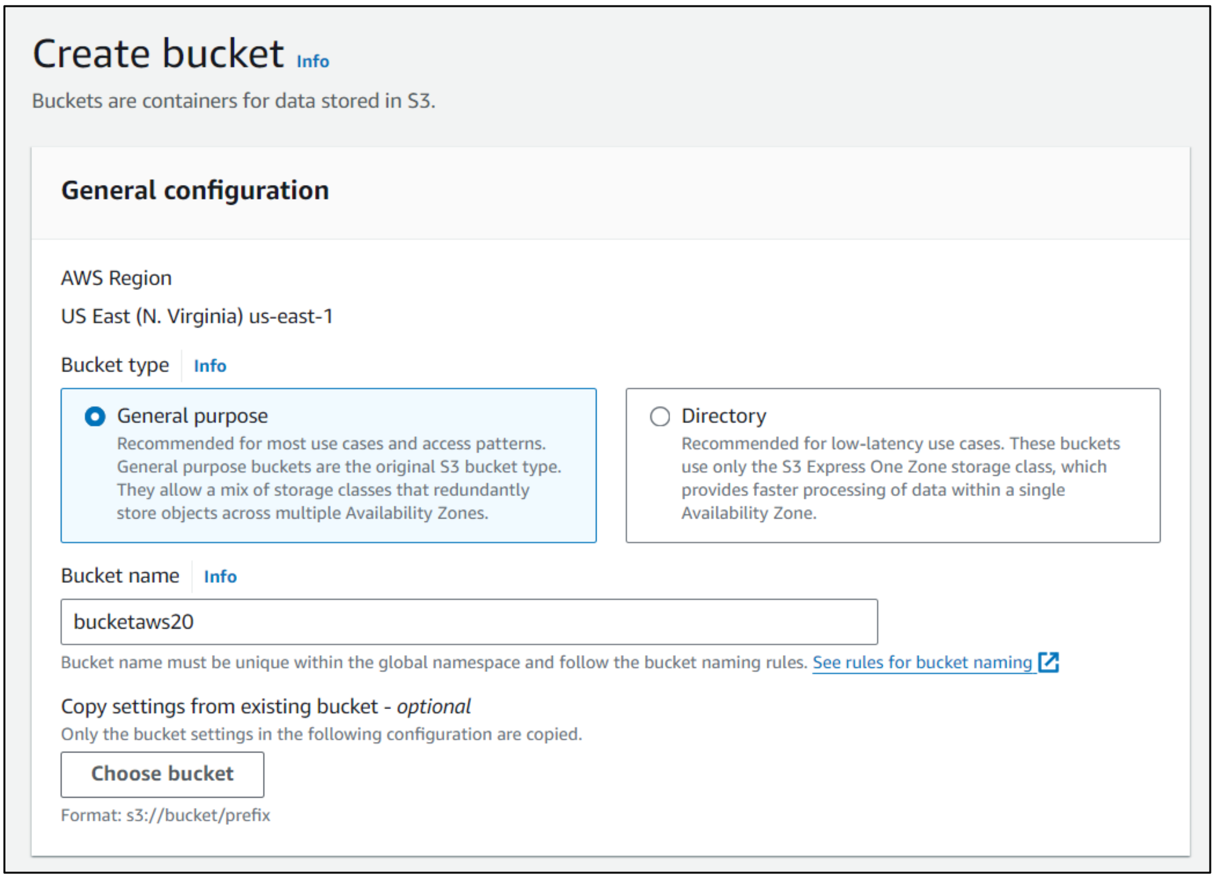
1. First of all, we have to create an S3 bucket for which sign in to your AWS account and search for S3 service and select it.



2. To create the bucket, select the “Create Bucket” button.



3. Give the bucket a name and select create bucket



Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type Info

[✓ General purpose] Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

bucketaws20

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

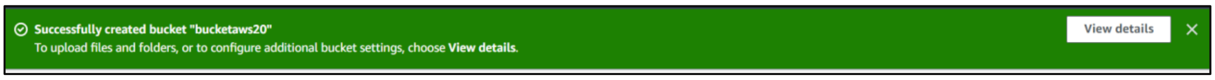
Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

4. The bucket will successfully be created.

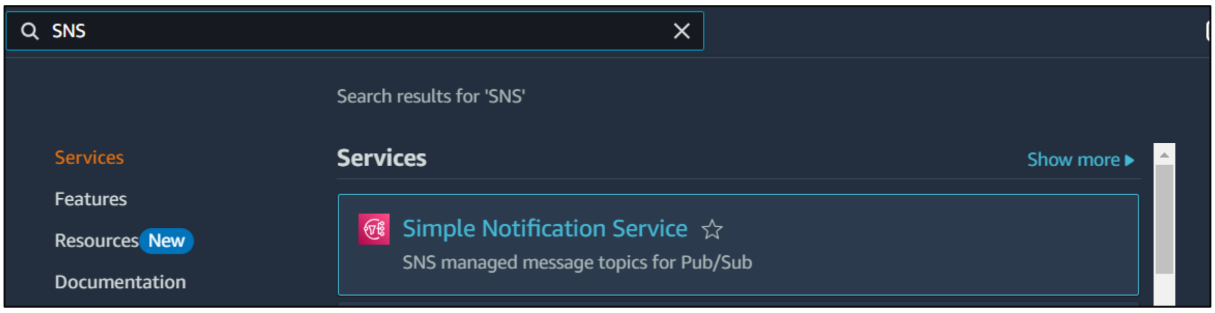


Successfully created bucket "bucketaws20"

To upload files and folders, or to configure additional bucket settings, choose **View details**.

View details

5. Now to create SNS topic search for SNS in services.



SNS

Search results for 'SNS'

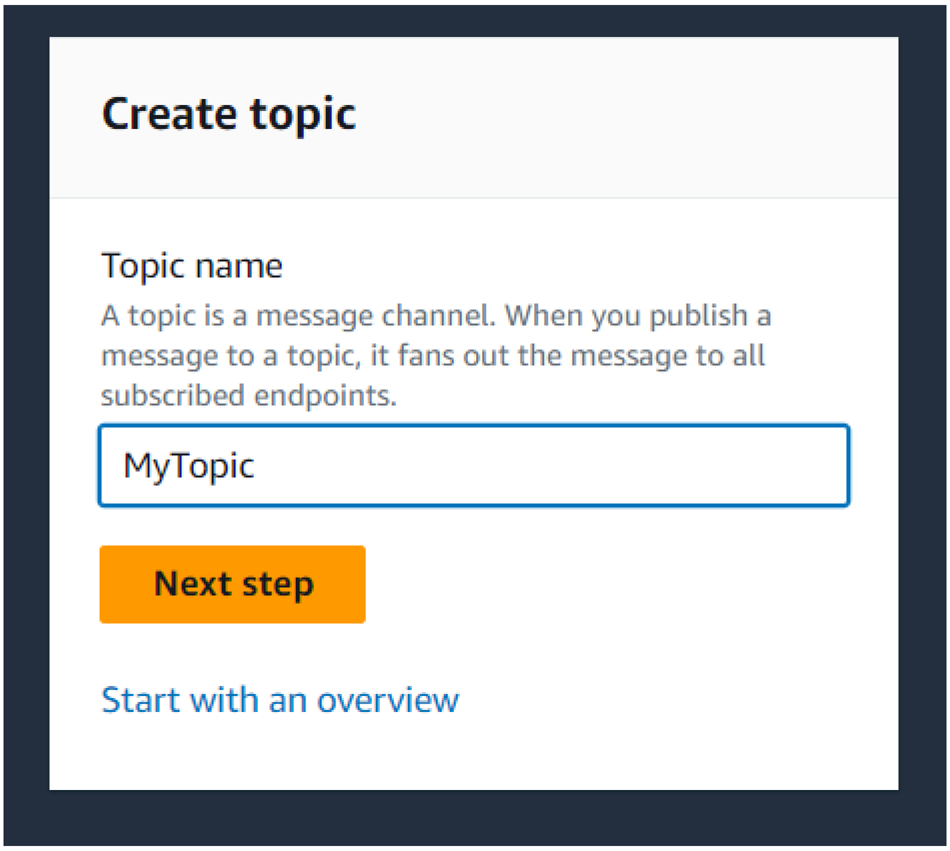
Services

Show more

Simple Notification Service

SNS managed message topics for Pub/Sub

6. Give your topic a name, here I have given the name "MyTopic"



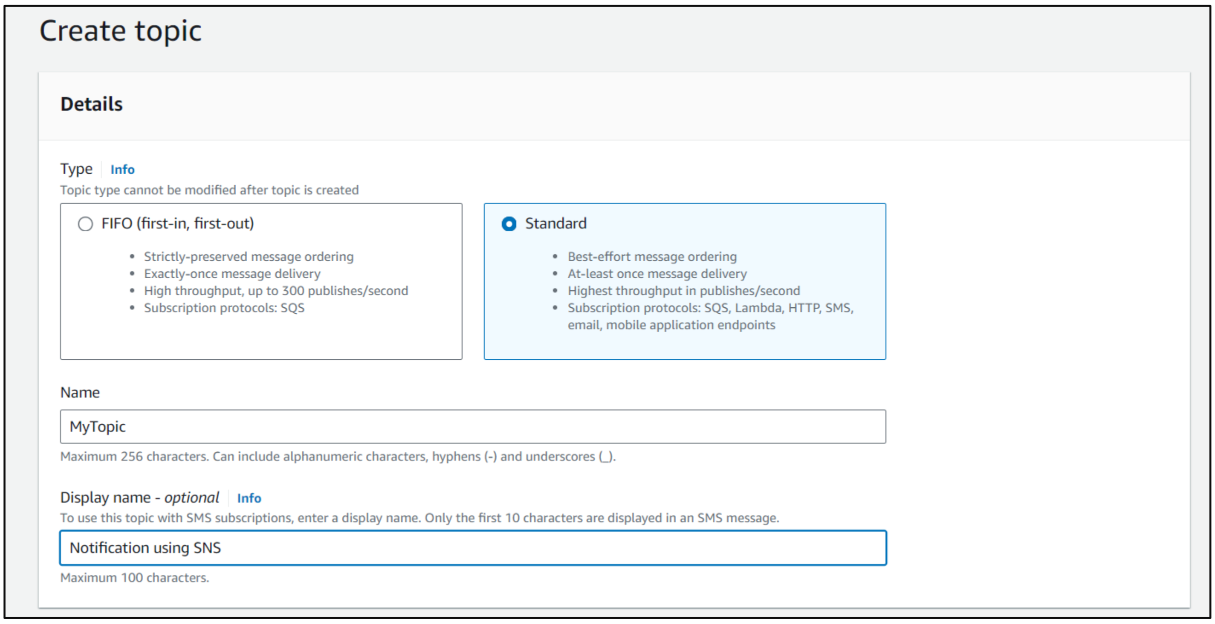
Create topic

Topic name

A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

MyTopic

7. Now select the type as Standard and the name field will automatically have your topic name.



Create topic

Details

Type Info
Topic type cannot be modified after topic is created

( ) FIFO (first-in, first-out)
- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

(● Standard)
- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

MyTopic

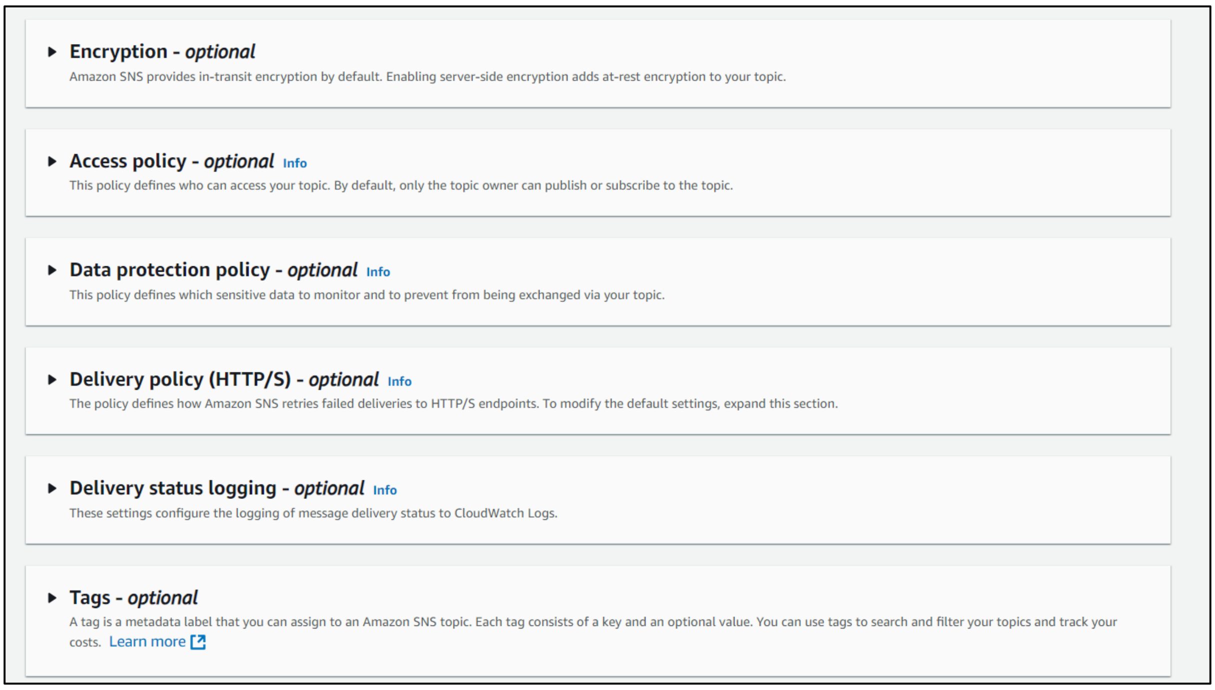
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional* Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

Notification using SNS

Maximum 100 characters.

8. You can also configure other things like encryption, data protection policy, access policy, etc. These are optional.



Encryption - *optional*
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

Access policy - *optional* Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

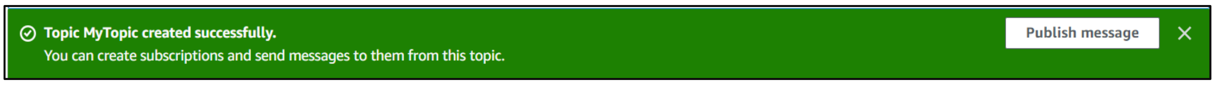
Data protection policy - *optional* Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

Delivery policy (HTTP/S) - *optional* Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

Delivery status logging - *optional* Info
These settings configure the logging of message delivery status to CloudWatch Logs.

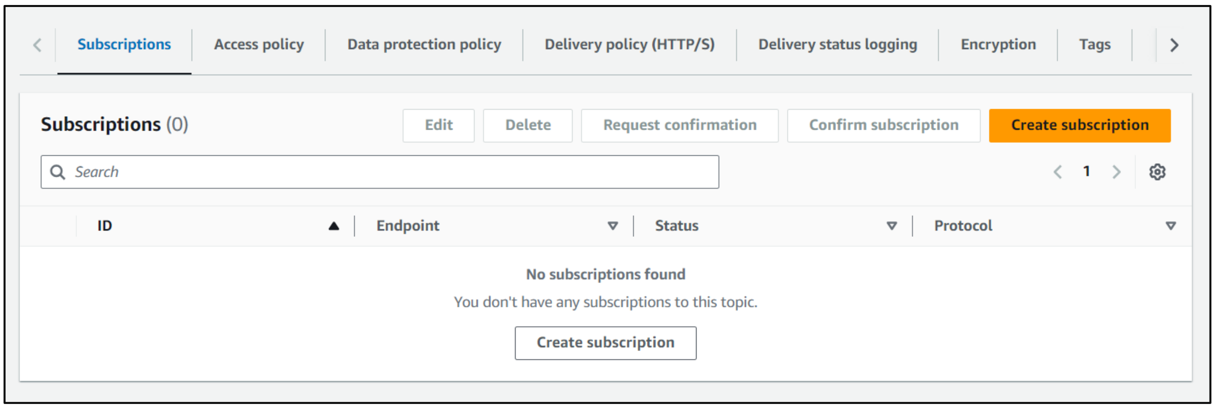
Tags - *optional*
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more

9. After all the configurations select create topic and you will see successfully created message displayed.



Topic MyTopic created successfully.
You can create subscriptions and send messages to them from this topic.

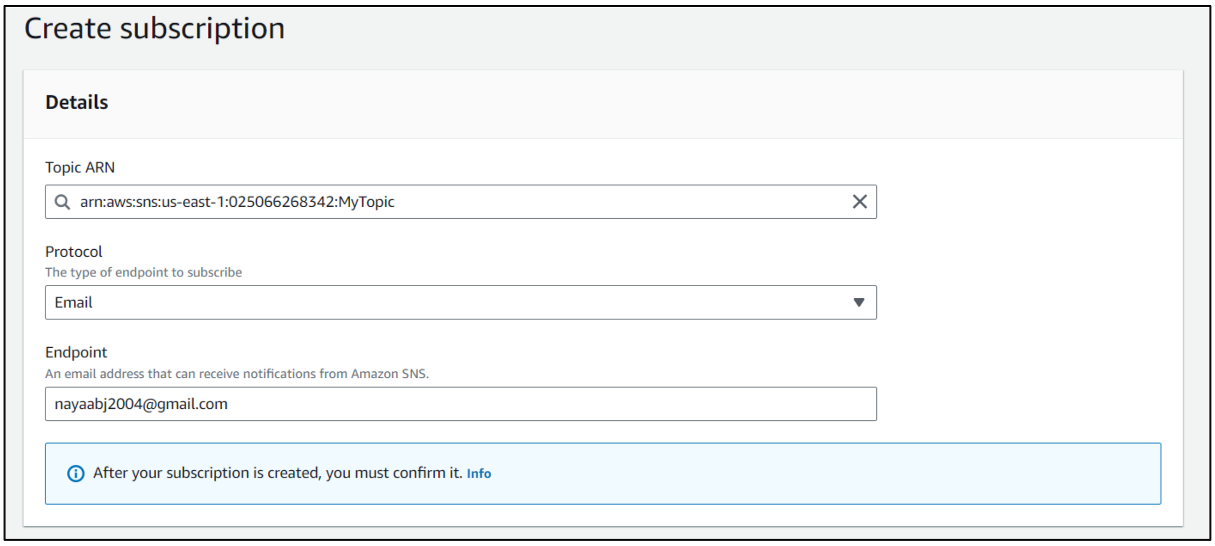
10. Scroll down in MyTopic to the subscriptions section in order to create a subscription. A subscription will define who will receive the messages published to the topic.



ID	Endpoint	Status	Protocol

No subscriptions found
You don't have any subscriptions to this topic.

11. Here you will see ARN which is Amazon Resource Name that is a unique identifier for each SNS topic (Keep a note of this ARN as we will need it afterwards). Below that select Email for protocol field and provide an email address where you want to receive the notification.



Create subscription

Details

Topic ARN

arn:aws:sns:us-east-1:025066268342:MyTopic

Protocol
The type of endpoint to subscribe

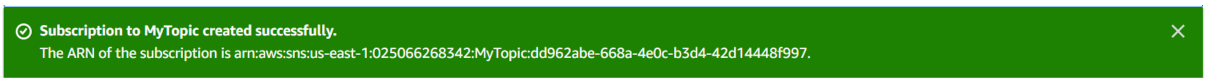
Email

Endpoint
An email address that can receive notifications from Amazon SNS.

nayaabj2004@gmail.com

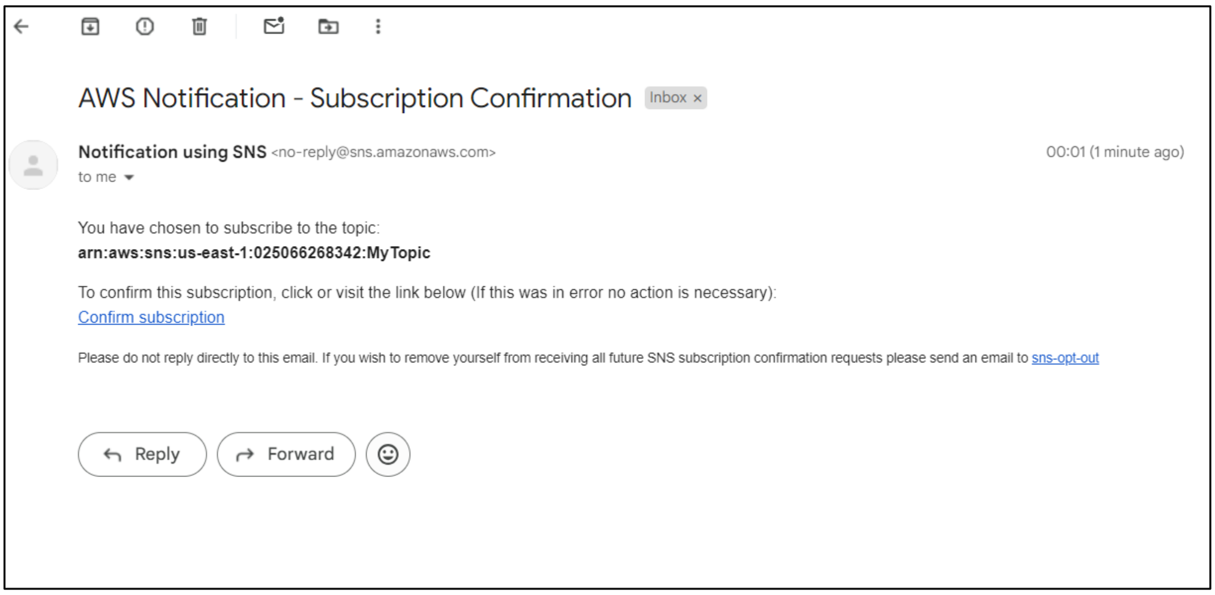
After your subscription is created, you must confirm it. Info

12. Finally click on create subscription.



Subscription to MyTopic created successfully.
The ARN of the subscription is arn:aws:sns:us-east-1:025066268342:MyTopic:dd962abe-668a-4e0c-b3d4-42d14448f997.

13. In order to confirm your subscription, go to the email address you have entered while creating the subscription. You will have received a mail like the one shown in the snippet below:



AWS Notification - Subscription Confirmation Inbox ×

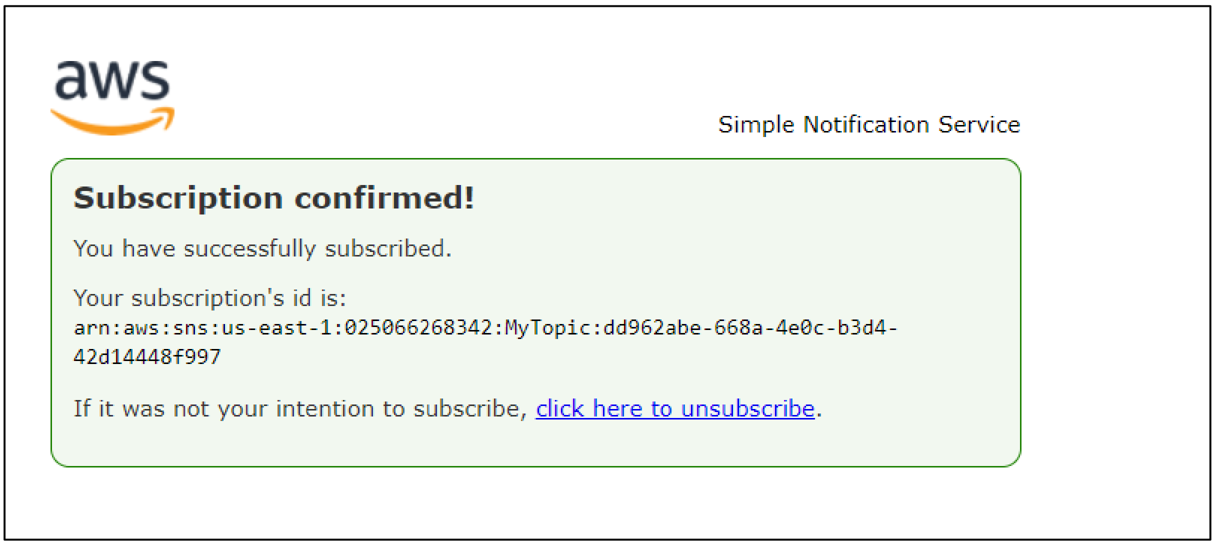
Notification using SNS <no-reply@sns.amazonaws.com> 00:01 (1 minute ago)
to me

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:025066268342:MyTopic

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

14. Once you click on the "confirm subscription" link in the mail your subscription will be confirmed and you will see the confirmation message along with the subscription id.



Simple Notification Service

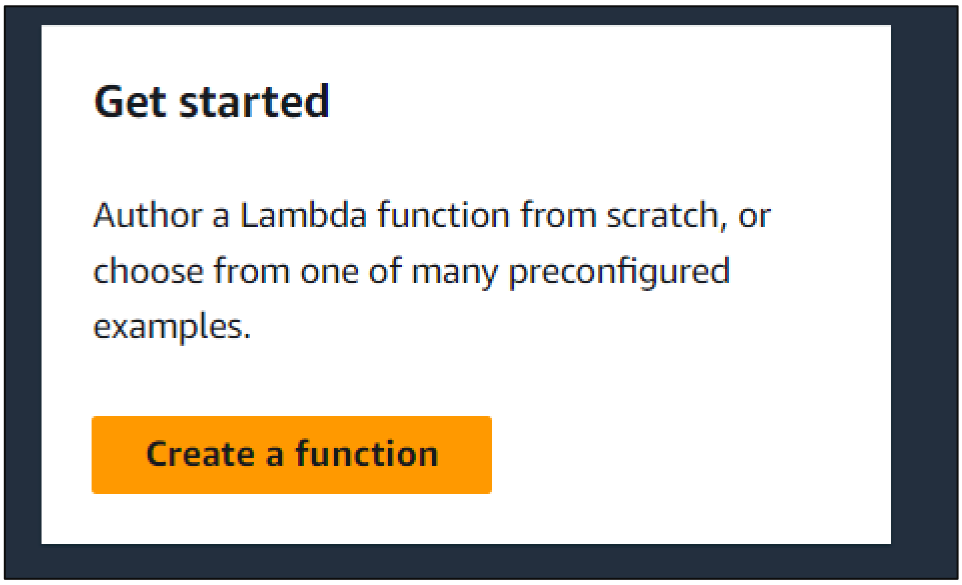
Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
```
arn:aws:sns:us-east-1:025066268342:MyTopic:dd962abe-668a-4e0c-b3d4-
42d14448f997
```

If it was not your intention to subscribe, click here to unsubscribe.

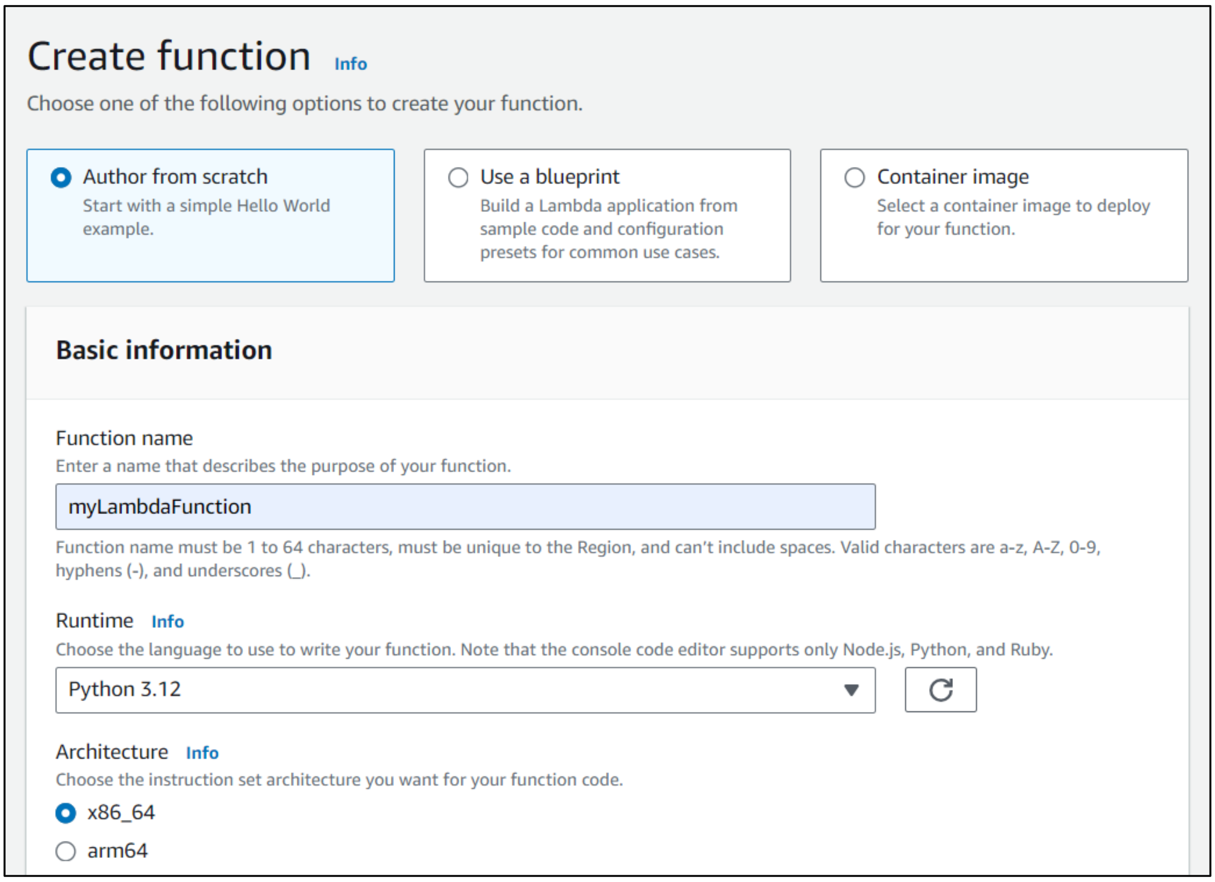
15. Now we will create a lambda function. For this search for lambda in the services, select it and click on create function



Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

16. Choose Author from scratch to create your function from scratch without using any prebuilt templates or blueprints. Also provide basic information like function name, language for writing the function (python, node js, java, ruby, etc.) and create the function.



Create function Info

Choose one of the following options to create your function.

[✓ Author from scratch — Start with a simple Hello World example.] Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases. Container image — Select a container image to deploy for your function.

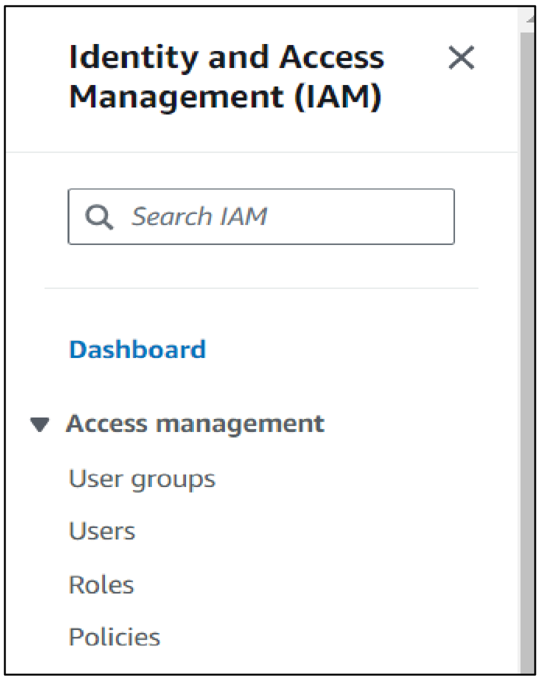
Basic information

Function name
Enter a name that describes the purpose of your function.
myLambdaFunction
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

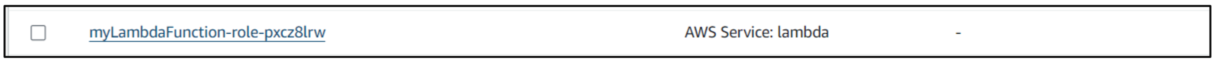
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.12

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64] arm64

17. To create IAM role for the lambda function search for IAM and in the left panel select roles option

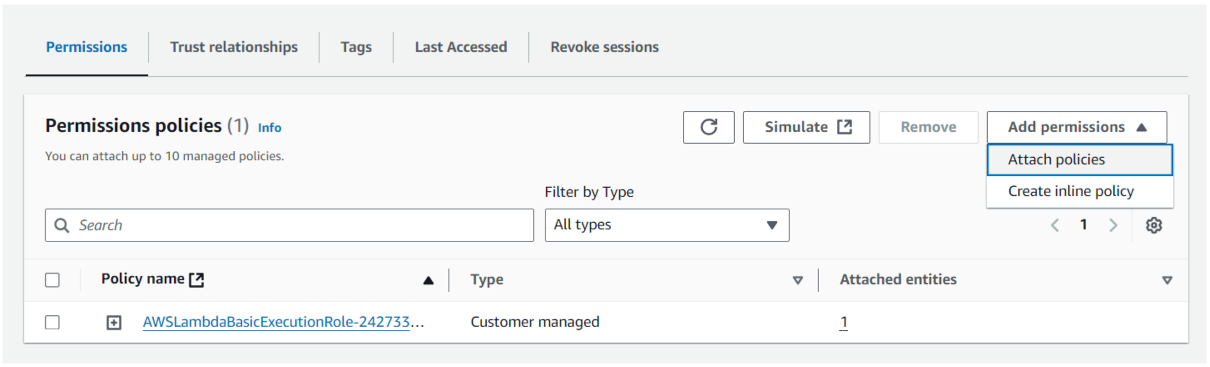


18. While creating lambda function a role is automatically created with basic permissions. Now we will provide additional permissions for the same role.



☐	myLambdaFunction-role-pxcz8lrw	AWS Service: lambda	-

19. As we can see LambdaBasicExecutionRole policy is present by default, to add other policies we will select Attach policies.



Permissions policies (1) Info

You can attach up to 10 managed policies.

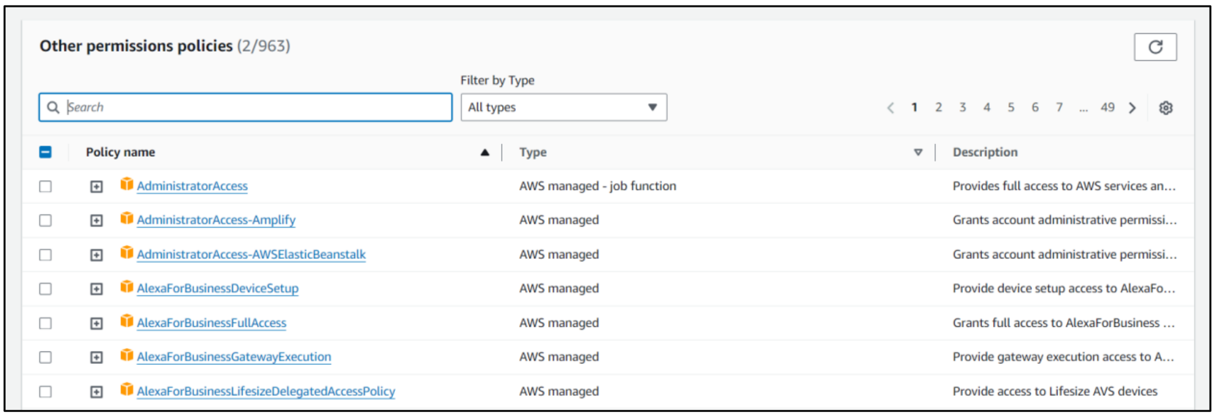
Filter by Type: All types

	Policy name	Type	Attached entities
☐	AWSLambdaBasicExecutionRole-242733…	Customer managed	1

20. Search for the following policies and select the checkbox next to them and click on attach policy: AmazonS3FullAccess, AmazonSNSFullAccess.

These are the functionalities of the policies:

<u>AmazonS3FullAccess:</u> To allow the Lambda function to access and read files from S3.
<u>AmazonSNSFullAccess:</u> To allow Lambda to publish messages to an SNS topic.

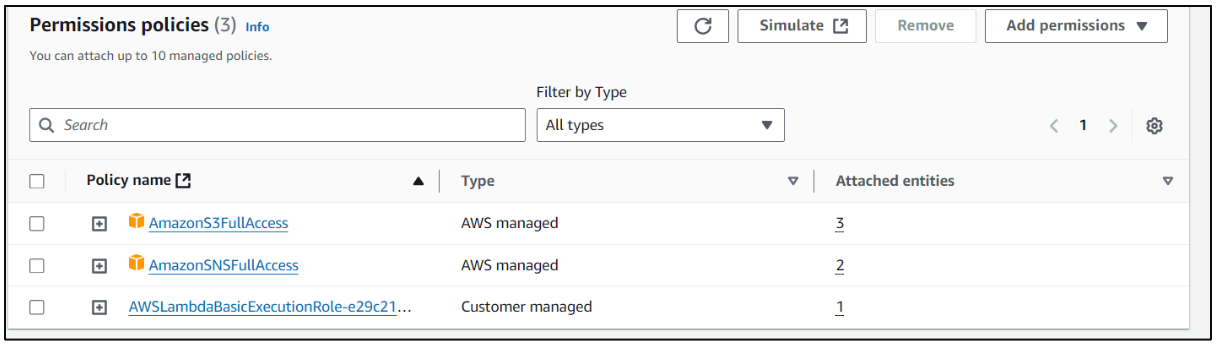


Other permissions policies (2/963)

Filter by Type: All types

	Policy name	Type	Description
☐	AdministratorAccess	AWS managed - job function	Provides full access to AWS services an…
☐	AdministratorAccess-Amplify	AWS managed	Grants account administrative permissi…
☐	AdministratorAccess-AWSElasticBeanstalk	AWS managed	Grants account administrative permissi…
☐	AlexaForBusinessDeviceSetup	AWS managed	Provide device setup access to AlexaFo…
☐	AlexaForBusinessFullAccess	AWS managed	Grants full access to AlexaForBusiness …
☐	AlexaForBusinessGatewayExecution	AWS managed	Provide gateway execution access to A…
☐	AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	Provide access to Lifesize AVS devices

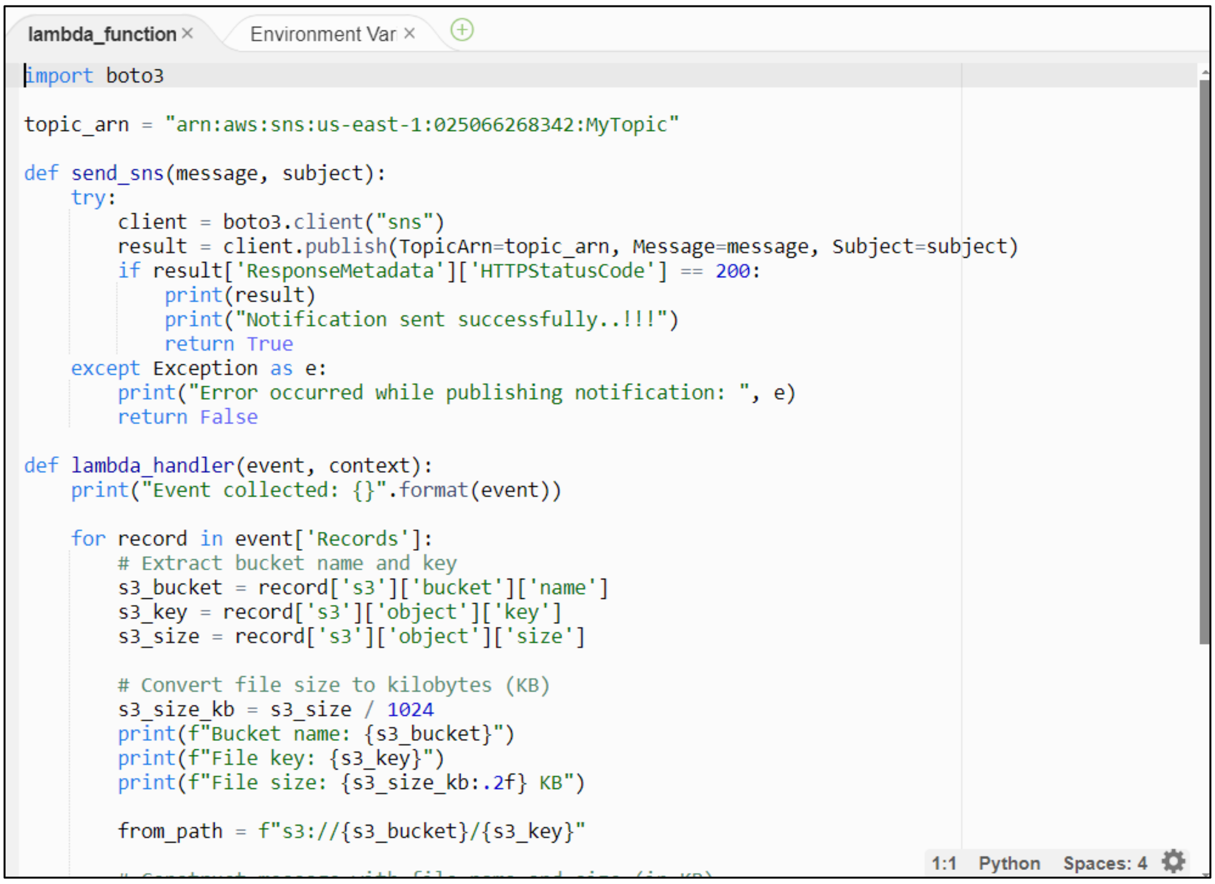
21. We have successfully added the permission policies.



You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	3
AmazonSNSFullAccess	AWS managed	2
AWSLambdaBasicExecutionRole-e29c21...	Customer managed	1

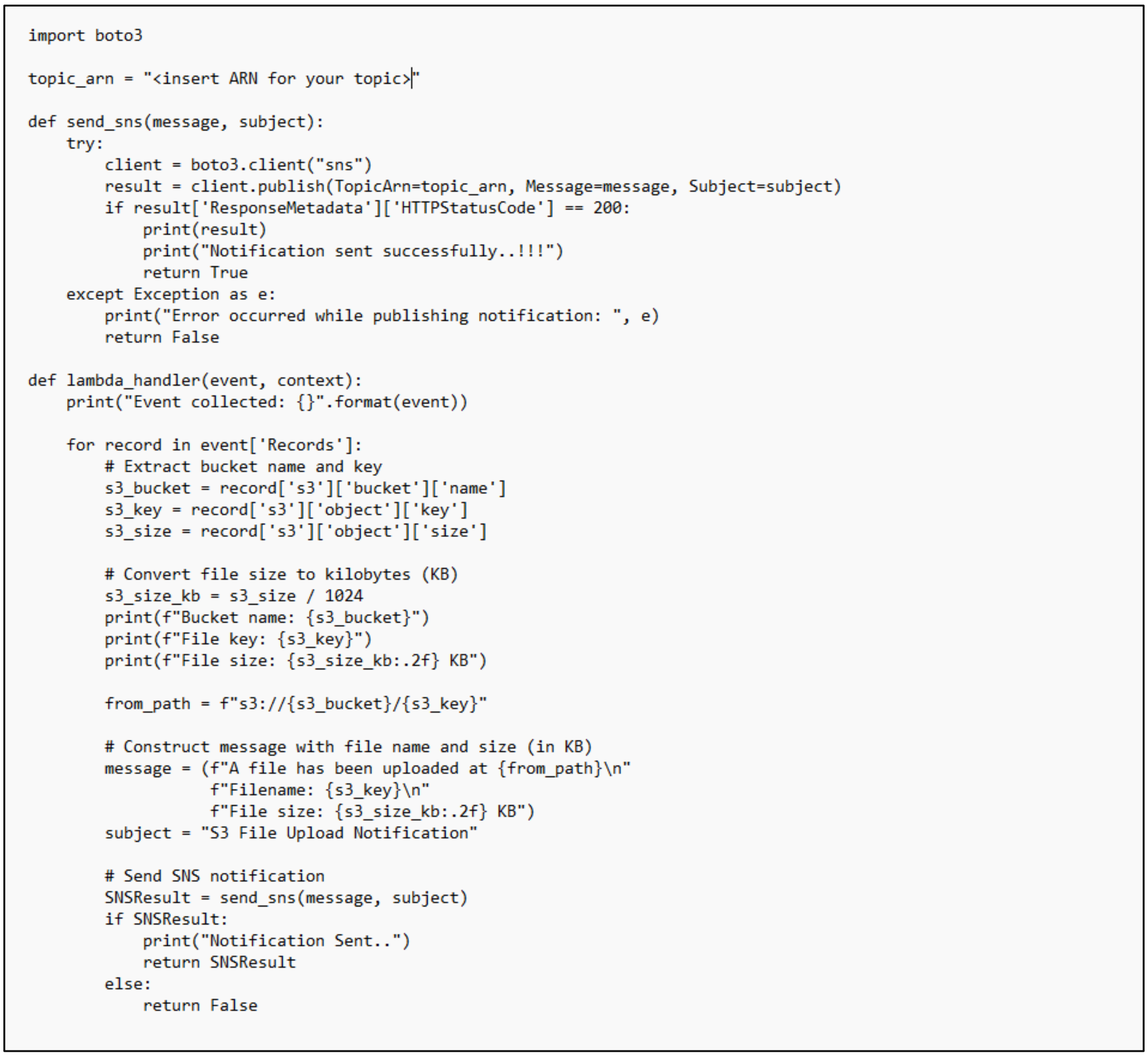
22. Now we will write the lambda function. As we selected python as the language while creating the function, we will be pasting python code in the code section of the lambda function like shown below:



```python
import boto3

topic_arn = "arn:aws:sns:us-east-1:025066268342:MyTopic"

def send_sns(message, subject):
    try:
        client = boto3.client("sns")
        result = client.publish(TopicArn=topic_arn, Message=message, Subject=subject)
        if result['ResponseMetadata']['HTTPStatusCode'] == 200:
            print(result)
            print("Notification sent successfully..!!!")
            return True
    except Exception as e:
        print("Error occurred while publishing notification: ", e)
        return False

def lambda_handler(event, context):
    print("Event collected: {}".format(event))

    for record in event['Records']:
        # Extract bucket name and key
        s3_bucket = record['s3']['bucket']['name']
        s3_key = record['s3']['object']['key']
        s3_size = record['s3']['object']['size']

        # Convert file size to kilobytes (KB)
        s3_size_kb = s3_size / 1024
        print(f"Bucket name: {s3_bucket}")
        print(f"File key: {s3_key}")
        print(f"File size: {s3_size_kb:.2f} KB")

        from_path = f"s3://{s3_bucket}/{s3_key}"
```

Click on deploy changes.

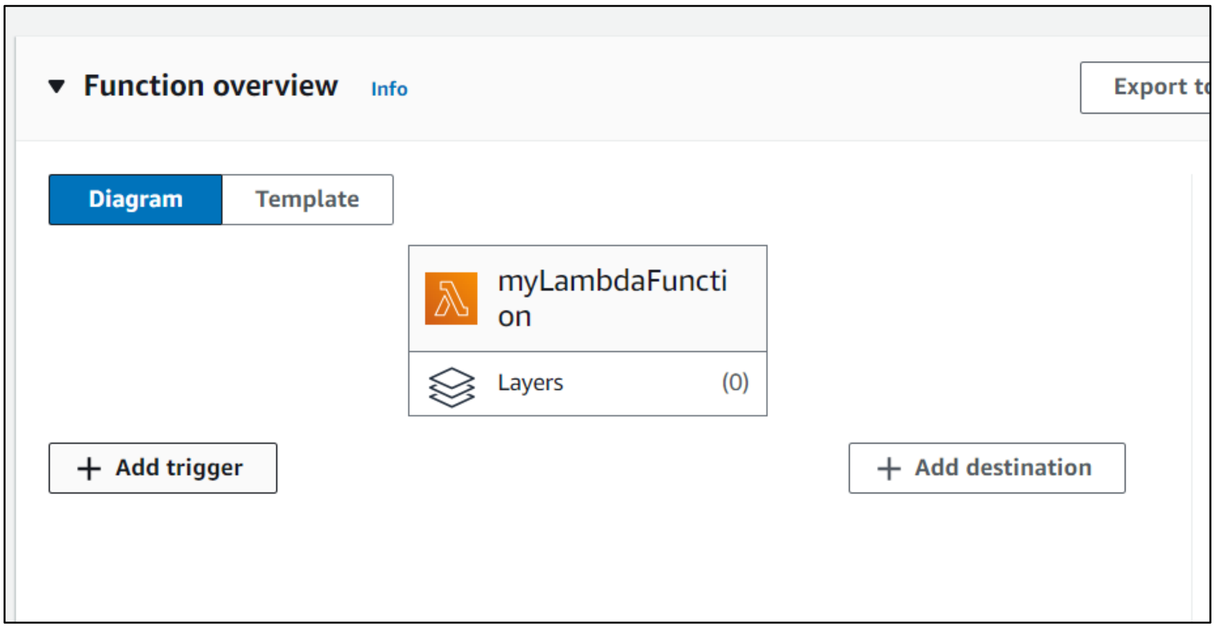
This is the code I have pasted above:



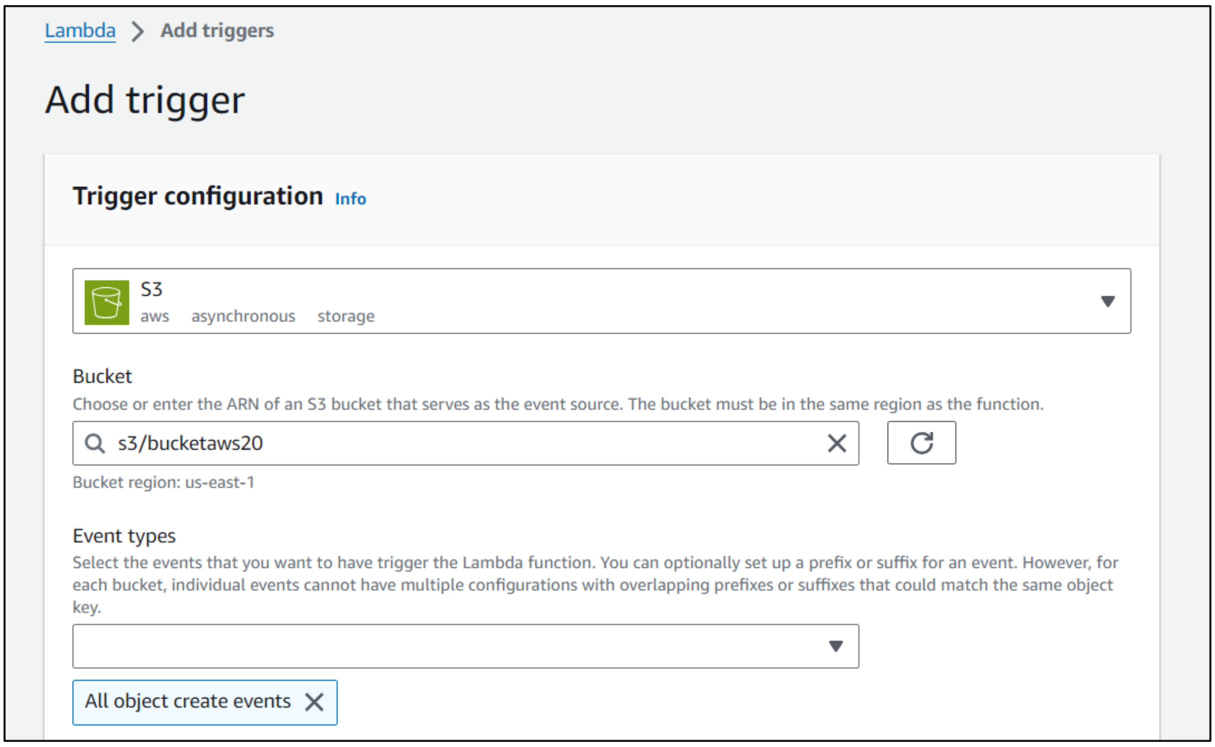
```python
import boto3

topic_arn = "<insert ARN for your topic>"

def send_sns(message, subject):
    try:
        client = boto3.client("sns")
        result = client.publish(TopicArn=topic_arn, Message=message, Subject=subject)
        if result['ResponseMetadata']['HTTPStatusCode'] == 200:
            print(result)
            print("Notification sent successfully..!!!")
            return True
    except Exception as e:
        print("Error occurred while publishing notification: ", e)
        return False

def lambda_handler(event, context):
    print("Event collected: {}".format(event))

    for record in event['Records']:
        # Extract bucket name and key
        s3_bucket = record['s3']['bucket']['name']
        s3_key = record['s3']['object']['key']
        s3_size = record['s3']['object']['size']

        # Convert file size to kilobytes (KB)
        s3_size_kb = s3_size / 1024
        print(f"Bucket name: {s3_bucket}")
        print(f"File key: {s3_key}")
        print(f"File size: {s3_size_kb:.2f} KB")

        from_path = f"s3://{s3_bucket}/{s3_key}"

        # Construct message with file name and size (in KB)
        message = (f"A file has been uploaded at {from_path}\n"
                   f"Filename: {s3_key}\n"
                   f"File size: {s3_size_kb:.2f} KB")
        subject = "S3 File Upload Notification"

        # Send SNS notification
        SNSResult = send_sns(message, subject)
        if SNSResult:
            print("Notification Sent..")
            return SNSResult
        else:
            return False
```

Just insert the ARN of your created topic for "topic_arn". This ARN is displayed when you create a subscription or you can also find it when you go to SNS->Topics.

23. Now we will add a trigger for the lambda function we have created. For that click Add trigger.



24. Select source as S3, choose the S3 bucket you created and for event types select the events for which you want to trigger the function, here I have selected All object create events. So, this will trigger the function whenever new objects are added to the bucket.



S3
aws asynchronous storage

Bucket

Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

s3/bucketaws20

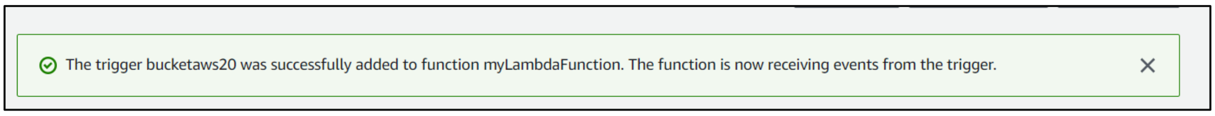
Bucket region: us-east-1

Event types

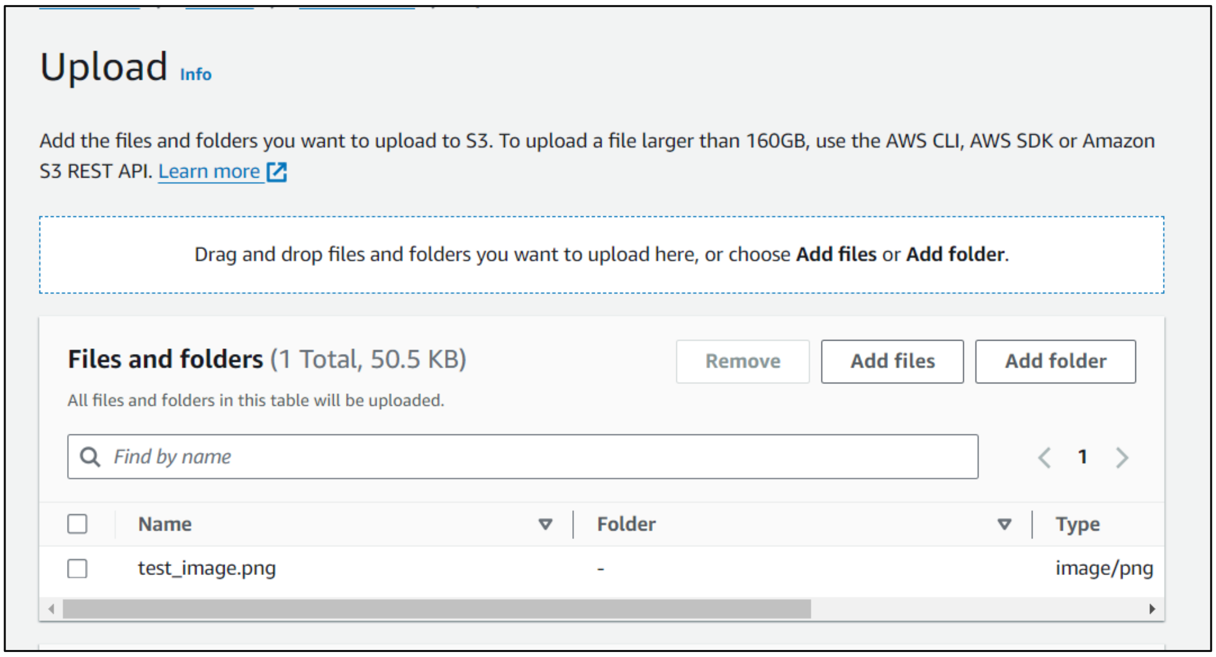
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events

Click on "add" to add trigger to the function


The trigger bucketaws20 was successfully added to function myLambdaFunction. The function is now receiving events from the trigger.

25. To test whether we get notified once we add a file to the S3 bucket we will upload an image to our bucket.


Upload Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more

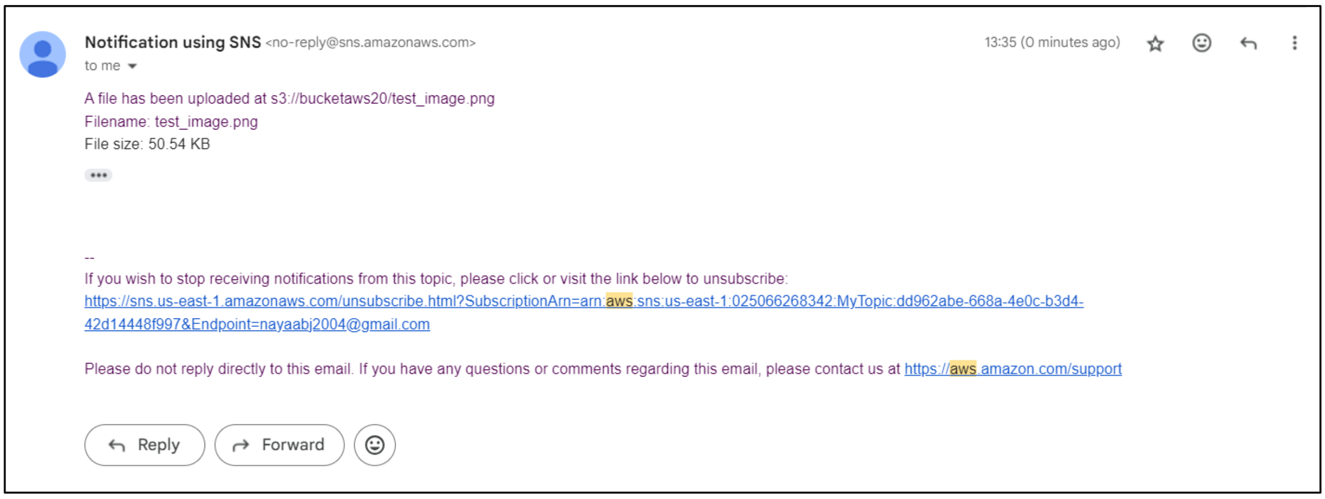
Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 50.5 KB)

All files and folders in this table will be uploaded.

Name	Folder	Type
test_image.png	-	image/png

Now check your mail.


Notification using SNS <no-reply@sns.amazonaws.com>
to me
13:35 (0 minutes ago)

A file has been uploaded at s3://bucketaws20/test_image.png
Filename: test_image.png
File size: 50.54 KB

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:025066268342:MyTopic:dd962abe-668a-4e0c-b3d4-42d14448f997&Endpoint=nayaabj2004@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Here we have received the notification with the file name and file size.

Additional Guidelines

In case you do not receive the mail these are some common ways to solve the issue:

1. Check for logs on CloudWatch. You will get to know if there were any errors during the execution of lambda function or any other permission related issues.
2. Check for the mail in the spam/junk folder as well if it is not visible in the primary inbox.
3. Verify if your topic ARN matches to the ARN in the lambda function code.
4. Make sure all the correct permissions have been attached to the IAM role.

Conclusion

This case study shows how AWS services like Lambda, S3, and SNS can be combined to automate tasks without needing a lot of infrastructure. When a file is uploaded to an S3 bucket, a Lambda function is triggered, extracting file details like file name and file size and sending a notification through SNS. This case study required the creation of an S3 bucket, lambda function, topic and subscription and also adding permissions to the IAM role.