**SNS(SIMPLE NOTIFICATION SERVICE) AND SUBSCRIPTION**

AWS Simple Notification Service (SNS) is a way to send notifications (like text messages, emails, or push notifications) to multiple users at once, acting like a cloud-based bulletin board. It's great for alerting users about events, sending updates, or managing messages within your applications.

How it works:

**1.Publishing a Message:**

You send a message to an SNS "topic," which is like a category for notifications.

**2. Subscribing:**

Users or applications "subscribe" to a topic to receive messages related to it.

**3. Delivery:**

SNS automatically delivers the message to all subscribed users or applications.

**Examples:**

**Alerts about low battery**: Your smartphone app can send you a notification if your phone's battery is low, using SNS.

**Lab Steps**

Task 1: Sign in to AWS Management Console

1. Click on the Open Console button, and you will get redirected to AWS Console in a new browser tab.

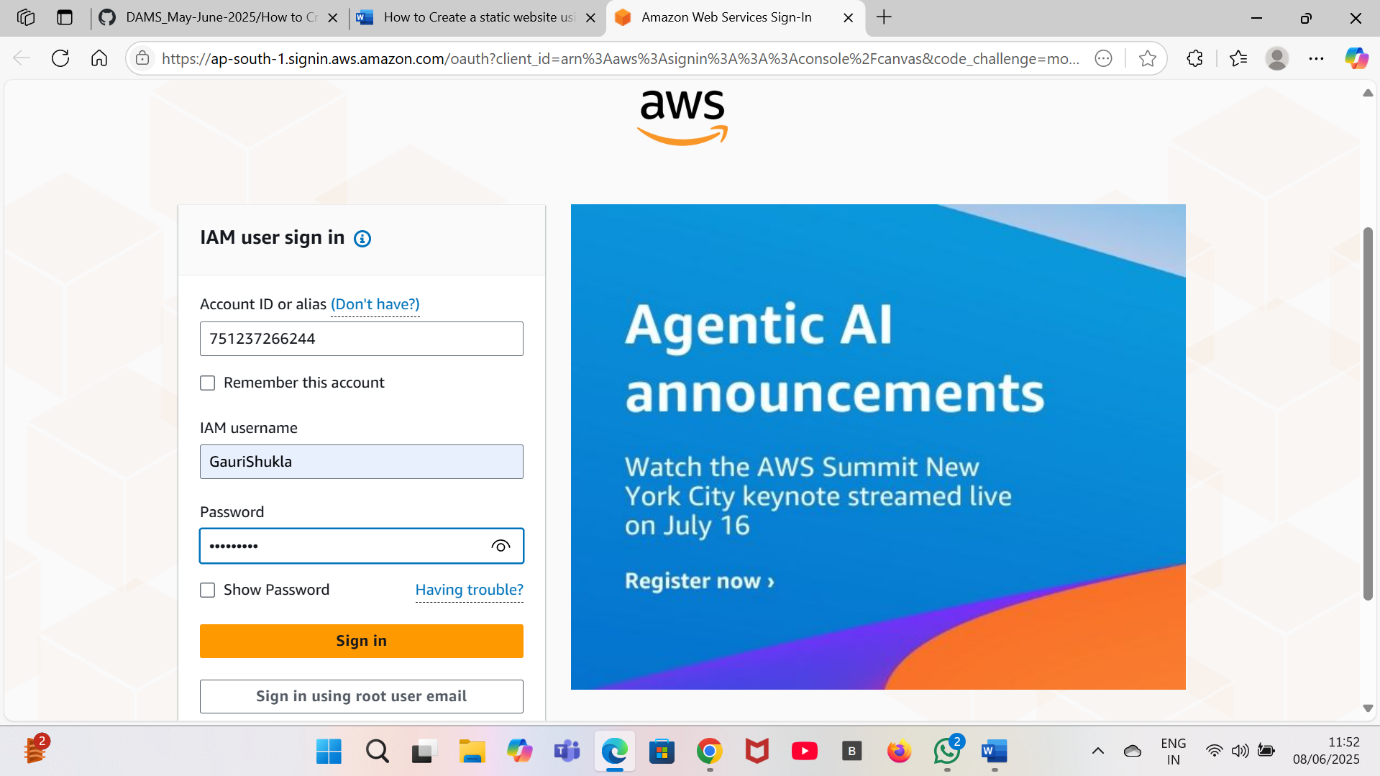
2. On the AWS sign-in page,

· Leave the Account ID as default

·Now enter your username and password.

3. click on sign-in.

4.After signing in select US East (N. Virginia) us-east- as AWS region.



**Task 2: Create SNS Topic**

1.Make sure the region is US East (N. Virginia) us-east-1.



2. Click on the search bar . Enter SNS. Select SNS.

3.Click on the topic present on the left side then click on create topic.

4.Now enter details:

a.Type:Standard

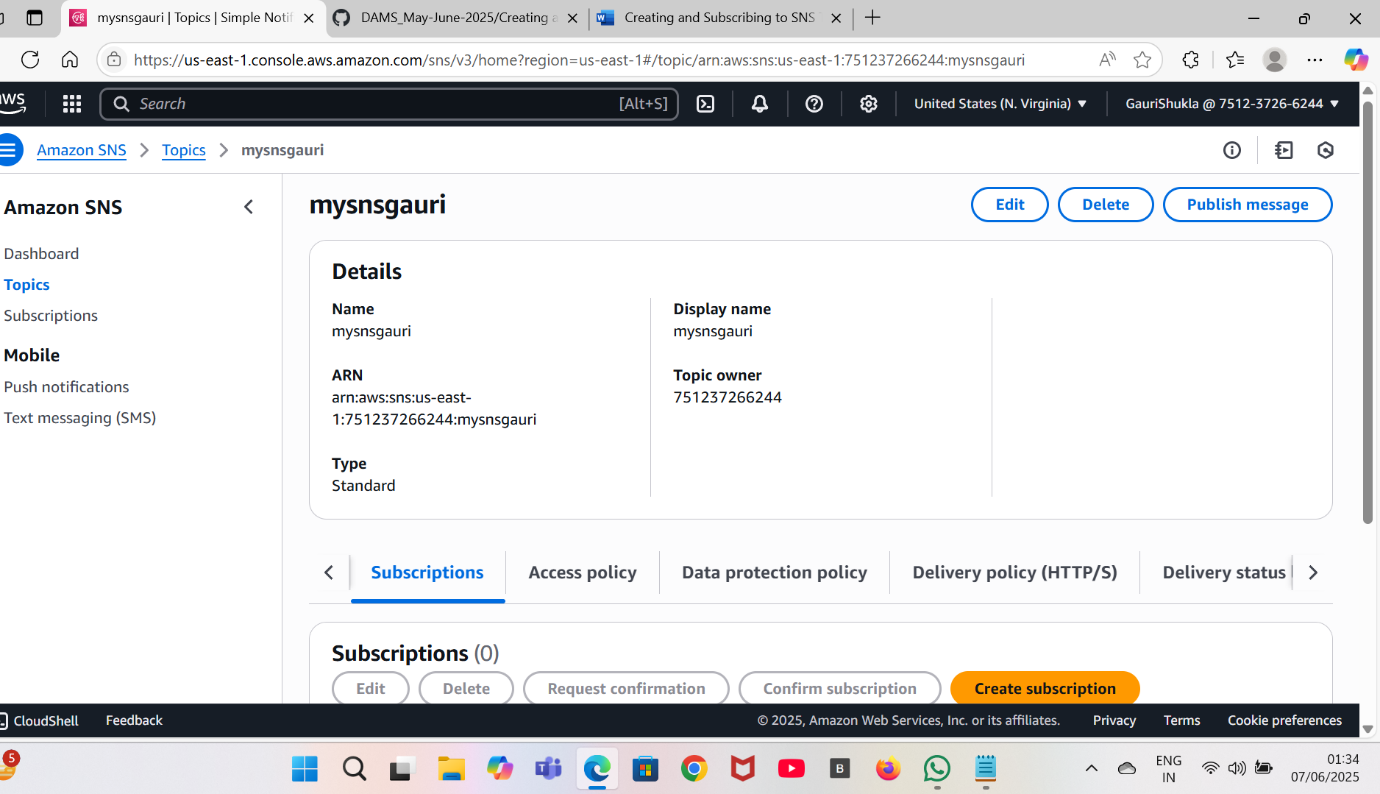
b.Name:Enter topic name(Mysnsgauri).

c.Display Name:Enter display name(mysnsgauri).

4. Leave other options as default and click on create topic at the end.

5.Your SNS topic is created.

6.Copy the ARN and save it for later use.

****

Task 3: Subscribe to SNS Topic

Now we are going to subscribe an e-mail to the SNS created earlier.

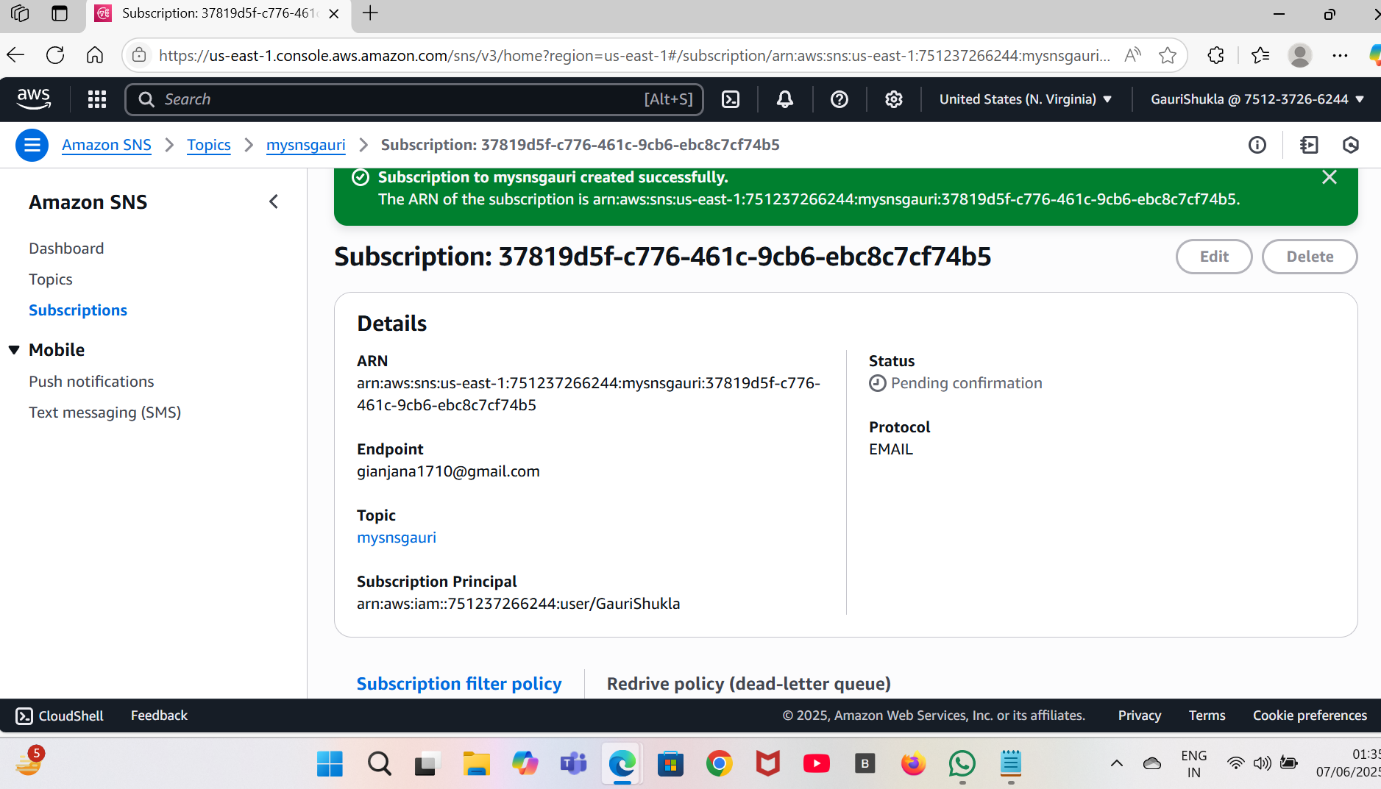
1.After SNS topic is created scroll down and click on create subscription.

2.Enter the details:

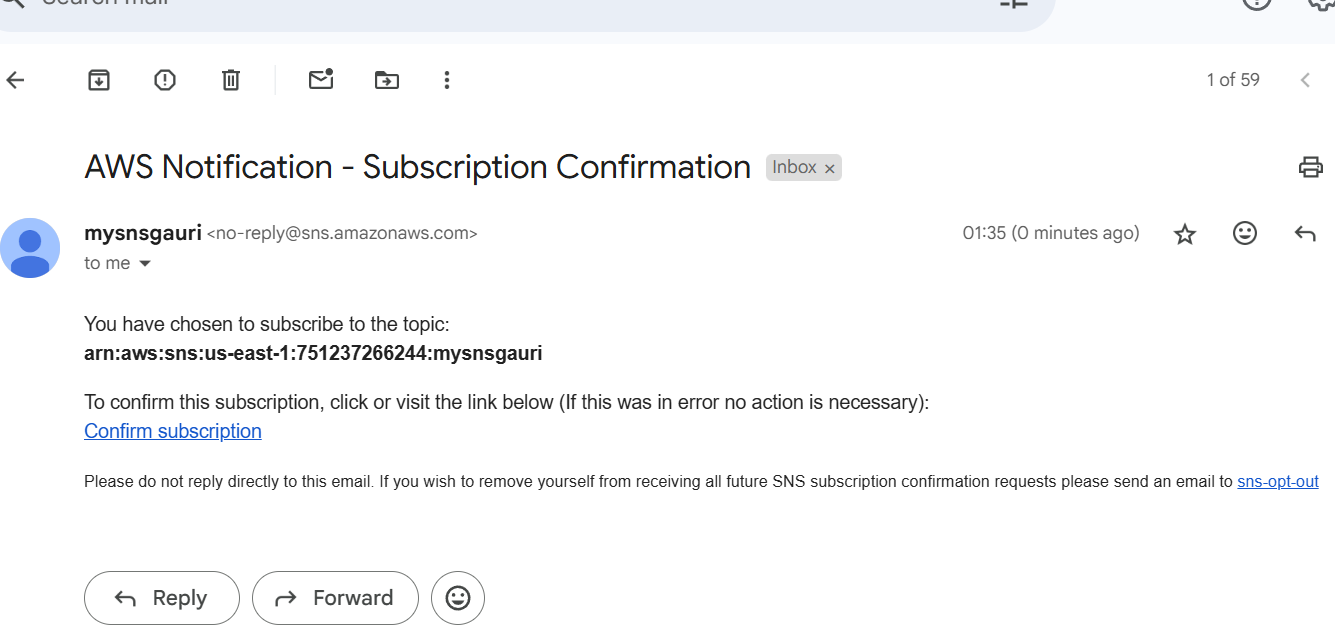
a. Protocol: Select E-mail

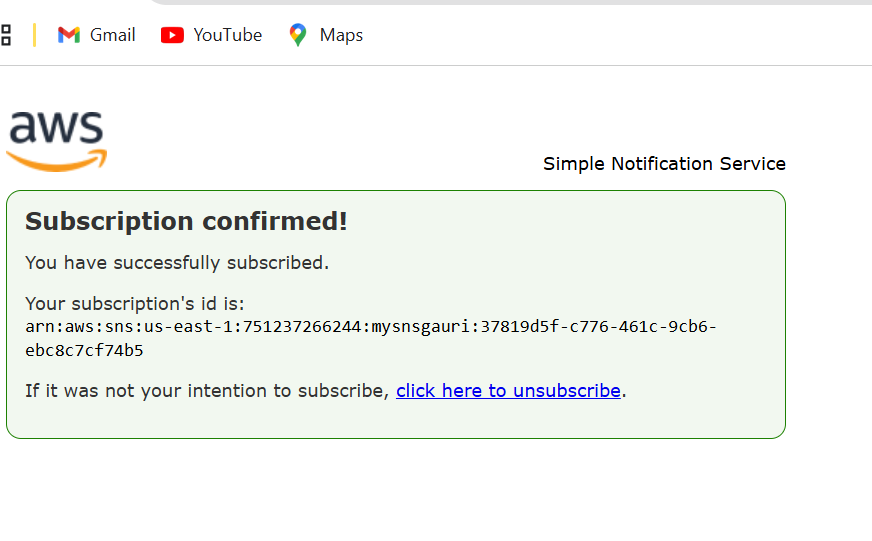
b.Endpoint: Enter your e-mail id

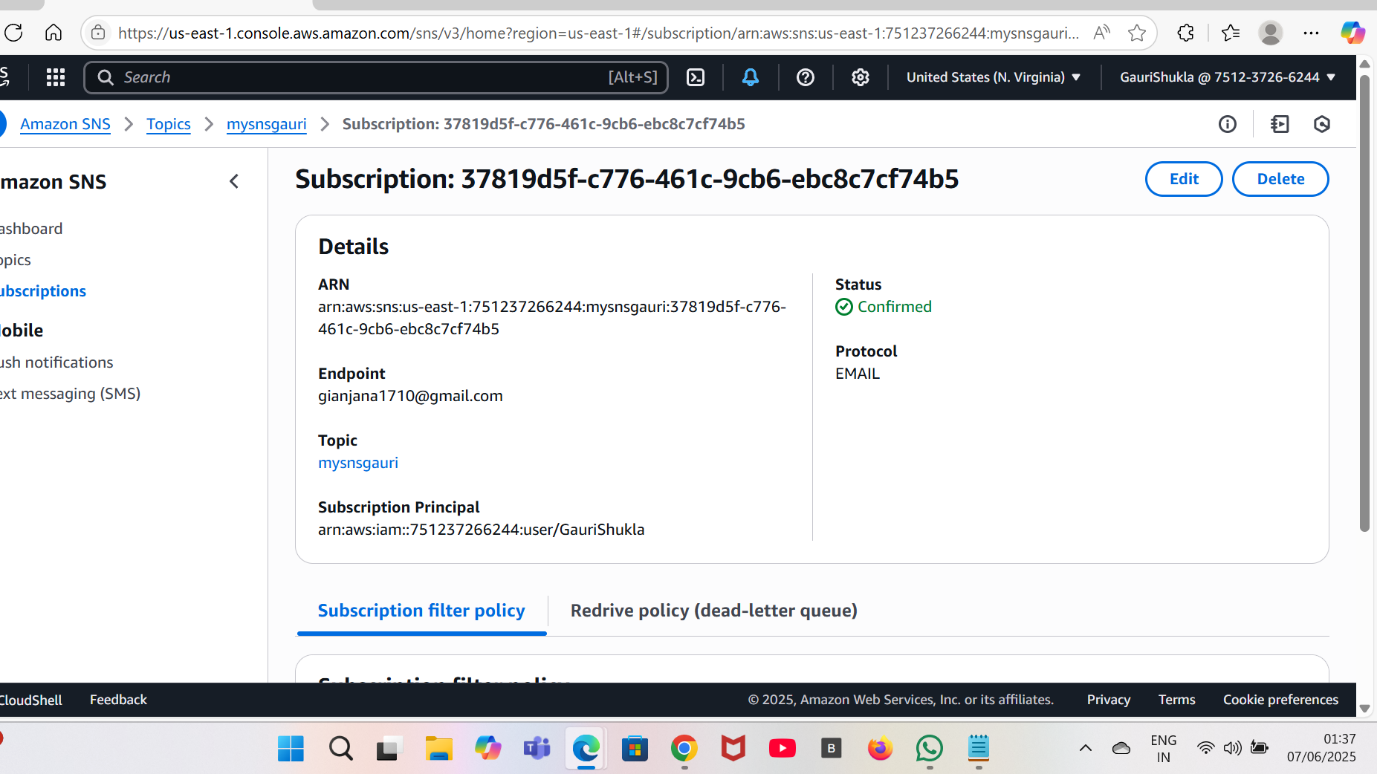
3.Leave the other settings as default and click on create subscription.



4.Now the check your e-mail inbox you would’ve received an SNS notification confirming your subscription to your email.



5.Your email address is now subscribed to your SNS topic mysnsgauri.



Task 4: Create an S3 Bucket

1.Navigate to S3.

2.Click on bucket from the left side and click on create bucket.

3.Enter the details:

a.Enter bucket name:gaurisnsbucket(you can enter name but remember it should be unique.)

b.Region:US East N.Virginia -us-east-1

c.Object ownership:Select ACLs disabled option.

4.Leave other options as default.Now click on create bucket.

Your bucket is created.

5.Now click on the created bucket and copy the ARN for later use.

Task 5: Update SNS Topic Access Policy

1.Navigate back to the SNS page.

2.Click on mysnsgauri.

3.Click on edit in the top right corner then click on access policy of the SNS topic.

4.Expand Access policy by clicking on it.

5.Remove the SNS policy mentioned over there. Paste the new SNS policy given below.

6. In the SNS policy given below you have to update two things

a.SNS Topic ARN in the resource section below.

b.S3 Bucket ARN in the bucket arn below.

{

"Version": "2008-10-17",

"Id": "\_\_default\_policy\_ID",

"Statement": [

{

"Sid": "\_\_default\_statement\_ID",

"Effect": "Allow",

"Principal": {

"AWS": "\*"

},

"Action": [

"SNS:GetTopicAttributes",

"SNS:SetTopicAttributes",

"SNS:AddPermission",

"SNS:RemovePermission",

"SNS:DeleteTopic",

"SNS:Subscribe",

"SNS:ListSubscriptionsByTopic",

"SNS:Publish",

"SNS:Receive"

],

"Resource": "<Your\_SNS\_Topic\_ARN>",

"Condition": {

"ArnLike": {

"aws:SourceArn": "<Your\_Bucket\_ARN>"

}

}

}

]

}

7.Click on save changes.

8. Now, your SNS topic has access to send notification events based on S3 bucket events.

Task 6: Create S3 Event

1.Navigate back to S3 page.

2.Click on the bucket which you have created earlier.

3.Go to the properties and scroll down to event notifications.

4.Click on create event notification.

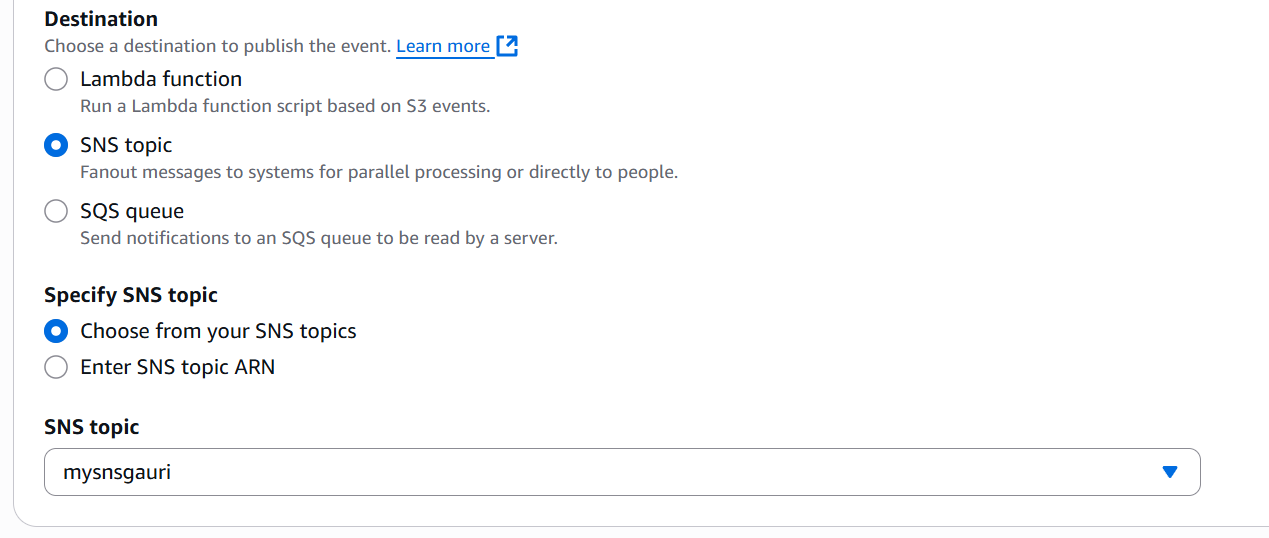
5.Enter the details: hellogauri

a. Event name:

b. Event type: Select put

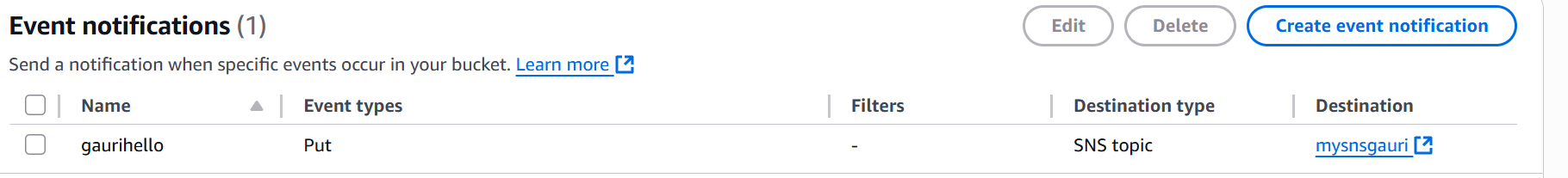
c. Under destination select SNS Topic.

d. Under SNS Topic select the topic created earlier.



6.Click on save changes.

7. Now the S3 bucket has been enabled event notifications for putting new objects through our SNS topic.



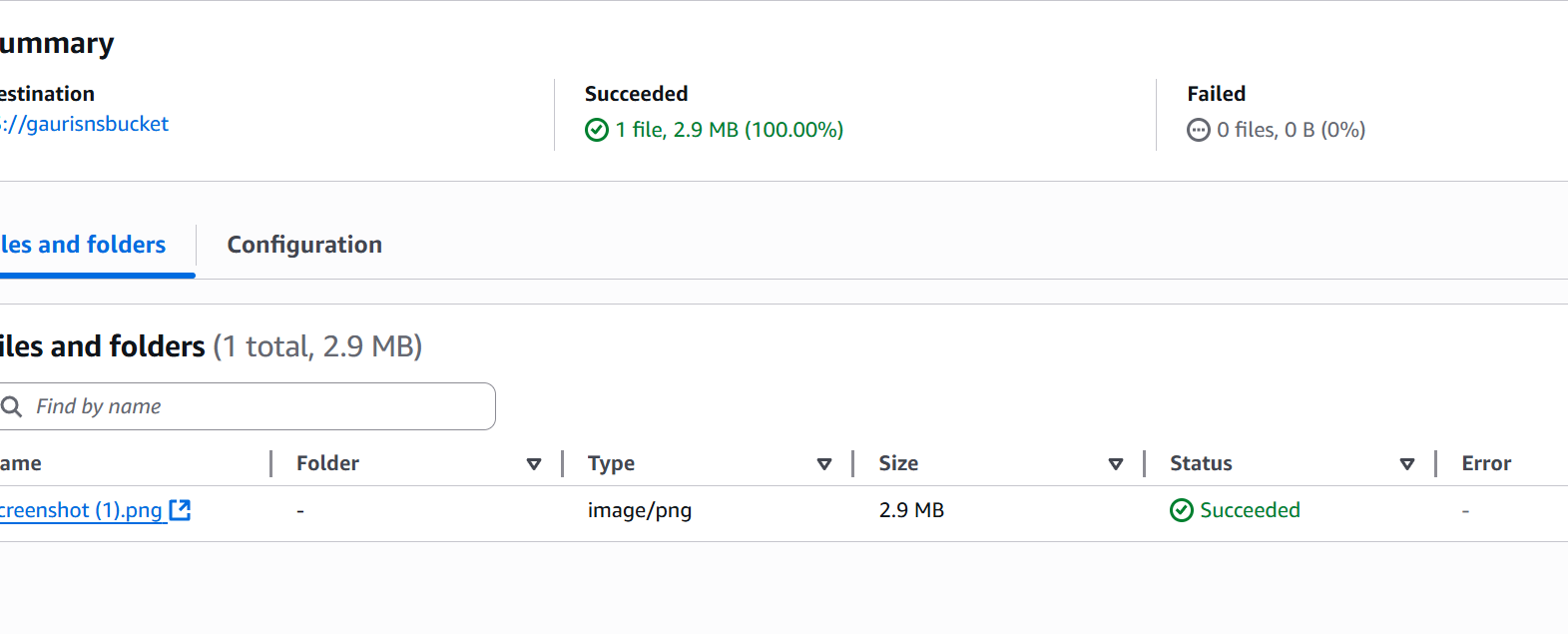
Task 7: Testing the SNS Notification

1.Open your S3 bucket gaurisnsbucket.

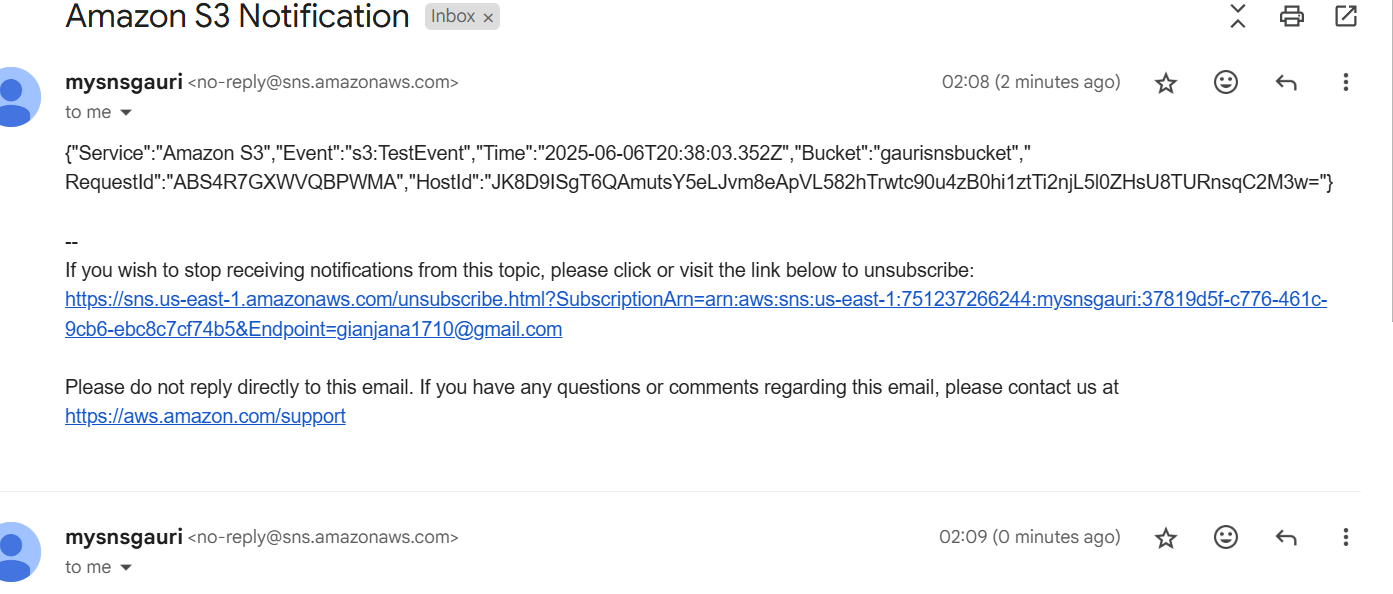
2.Under objects , click on upload.

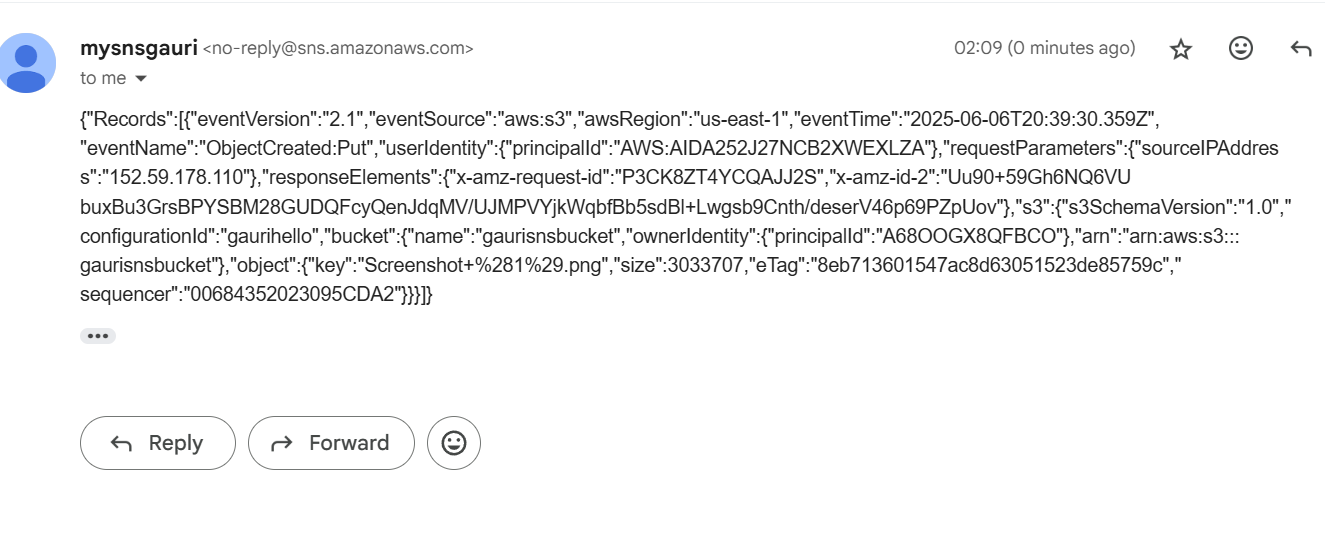
3.Now click on add files and upload an image from your pc .

4.Once the image is uploaded on the S3 bucket, you can see the uploaded image in the objects.



5. You have successfully received an SNS notification based on the PUT object event in S3 bucket.





**INTERVIEW QUESTIONS RELATED TO SNS(SIMPLE NOTIFICATION SERVICE)**

Scenario 1: Ensuring High Availability for Critical Alerts

Problem Statement: Your team manages a mission-critical system that must notify administrators immediately in case of failures. However, email notifications often get delayed, and SMS is expensive. What’s the best way to ensure high availability and real-time alerts?

Solution:

* Use AWS SNS with multiple subscriptions to distribute messages across various channels.
* Combine email, SMS, and AWS Lambda (to trigger an automated remediation process).
* Use Amazon CloudWatch Alarms with SNS to trigger alerts instantly.
* Enable Amazon SQS (Simple Queue Service) as a backup in case any notification delivery fails.

Scenario 2: Filtering Messages for Different Teams

Problem Statement: Your company has different teams (DevOps, Security, and Support) who need specific notifications based on event categories. How do you ensure that each team receives only relevant messages?

Solution:

* Utilize SNS message filtering.
* Create a single SNS topic but use message attributes.
* Set up different subscriptions for each team and apply filter policies based on the event type.
* Example:
* DevOps: "eventType": ["deployment", "server-down"]
* Security: "eventType": ["unauthorized-access", "data-breach"]
* Support: "eventType": ["customer-ticket"]

Scenario 3: Handling Message Fan-Out with Multiple Subscribers

Problem Statement: A retail company needs to send order confirmation messages to customers via email, push notifications, and an internal system. How do you efficiently distribute these notifications?

Solution:

* Use SNS fan-out architecture.
* Publish a message to an SNS topic and have multiple SQS queues and Lambda functions subscribed.
* Email notifications go to Amazon SES.
* Push notifications go to Amazon Pinpoint.
* Internal processing happens via an SQS queue.

Scenario 4: Preventing Duplicate Notifications

Problem Statement: Your system occasionally sends duplicate notifications due to retries. How do you prevent users from receiving the same message multiple times?

Solution:

* Use SNS Message Deduplication (for FIFO topics).
* Generate a unique message ID and store it in a cache (e.g., DynamoDB with TTL).
* Implement idempotency in subscribers to ensure repeated messages are ignored.

Scenario 5: Ensuring Secure Access to SNS Topics

Problem Statement: You need to ensure that only authorized applications and services can publish or subscribe to your SNS topics. What’s the best security approach?

Solution:

* Implement SNS access policies to restrict access based on IAM roles.
* Use AWS KMS (Key Management Service) for encryption.
* Require authentication via IAM policies for publishing messages.
* Use private VPC endpoints if SNS is used within an internal AWS network.

Scenario 6: Handling SNS Message Delivery Failures

Problem Statement: Some of your SNS messages fail to deliver. How do you troubleshoot and ensure message reliability?

Solution:

* Enable Dead Letter Queues (DLQs) in SNS to capture failed messages.
* Set up CloudWatch Logs for SNS to track message delivery statuses.
* Implement a retry mechanism for transient failures.
* Check if the subscription endpoint (Lambda, SQS, HTTP) is correctly configured and reachable.

Scenario 7: Reducing SNS Cost for High-Volume Messages

Problem Statement: You’re sending millions of notifications daily via SMS, and costs are skyrocketing. How do you optimize costs while maintaining reliability?

Solution:

* Use Amazon Pinpoint instead of SNS for bulk SMS at a lower cost.
* Prioritize email and push notifications over SMS.
* Enable batch processing where possible to reduce API calls.
* Leverage region-specific pricing to minimize expenses.