AWS Lambda triggered by S3 to process 'Word Count' and notify via SNS

JULEANNY NAVAS
AWS Cloud Computing

Introduction

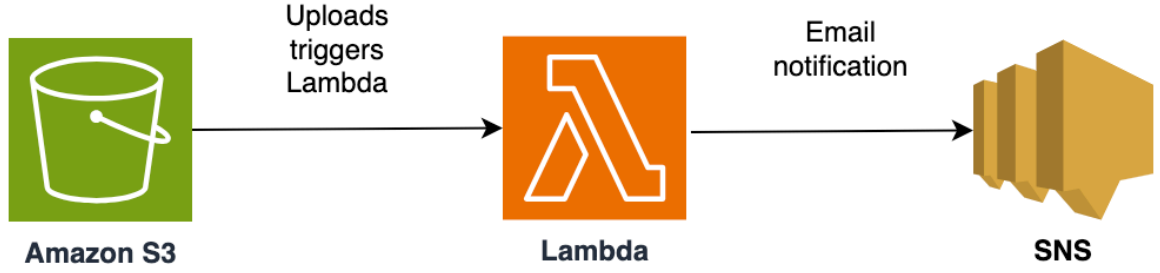
This project showcases the power of AWS **Serverless Computing** by automating word count processing using **AWS Lambda**. Each time a text file is uploaded to an **Amazon S3** bucket, an AWS Lambda function is triggered to count the number of words in the file. The result is then sent via email using **Amazon Simple Notification Service (SNS)**.

This event-driven architecture demonstrates efficient **cloud automation**, eliminating the need for traditional server management while ensuring seamless data processing and notification delivery.

Technologies used

- **AWS Lambda** (Python-based function execution)
- **Amazon S3** (Storage for text files triggering Lambda execution)
- **Amazon SNS** (Notification service for email alerts)
- https://app.diagrams.net (Architecture diagram)

Architecture Overview



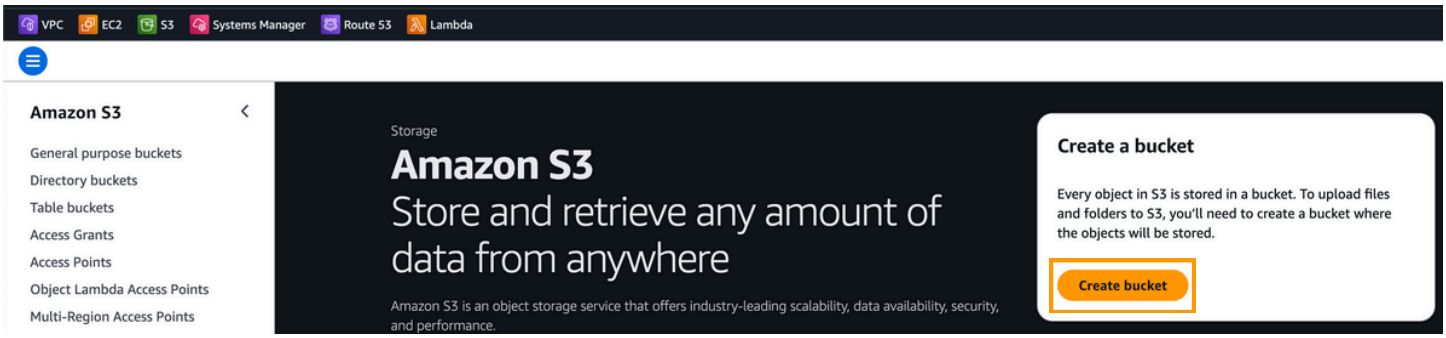
Step-by-Step Implementation

1 . Setting up an S3 bucket:

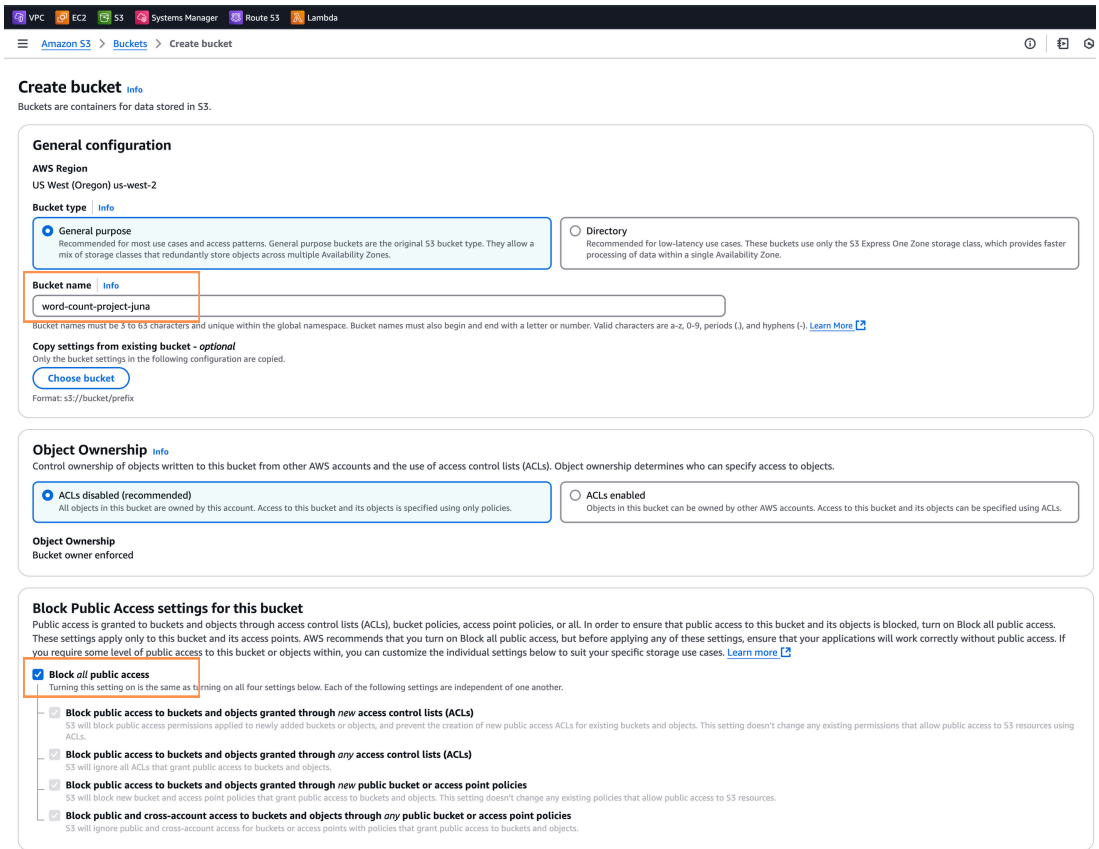
Amazon Simple Storage Service (**Amazon S3**) is a scalable object storage service that allows users to store and retrieve data with high availability and security. In this project, I used an S3 bucket to store text files, which will trigger the Lambda function upon upload.

Steps:

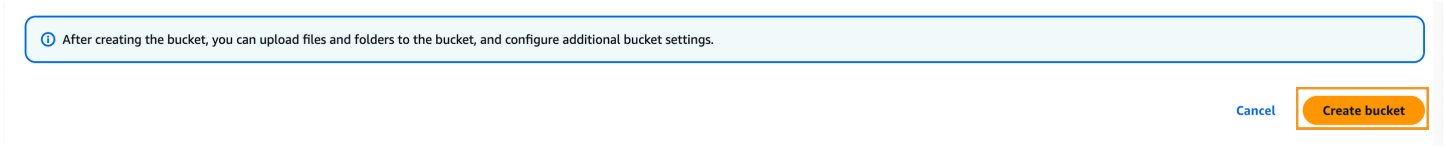
- Navigate to **S3** in the **AWS Management Console:**
- Click **"Create bucket"**.

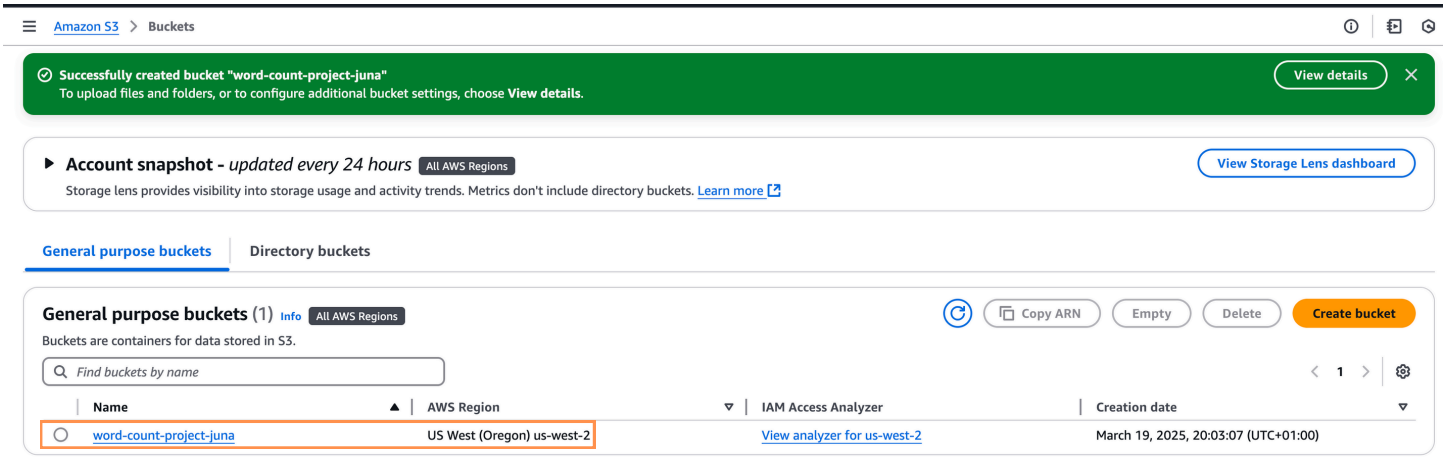


- Enter a unique bucket name (**word-count-project-juna**).
- Choose a **region** (ensure all resources are in the same region).
- Keep the **Block Public Access** settings as default (disabled for security).



- Click **"Create bucket"** to finalize.



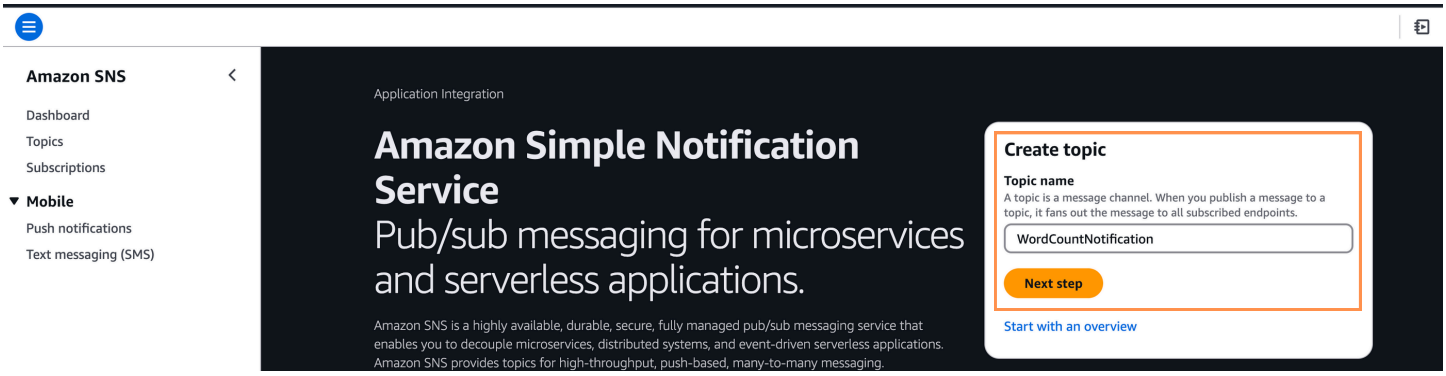


2 . Creating an SNS Topic

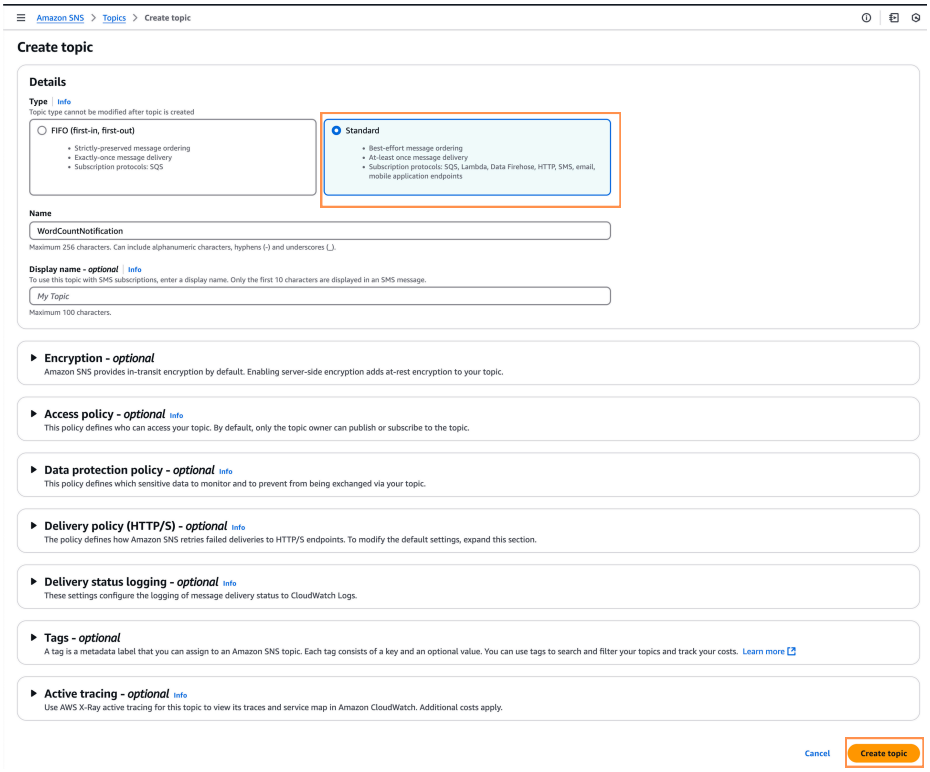
Amazon Simple Notification Service (**Amazon SNS**) is a fully managed messaging service that enables event-driven communication between distributed systems, microservices, and users. In this project, SNS is used to send an email notification with the word count result.

Steps:

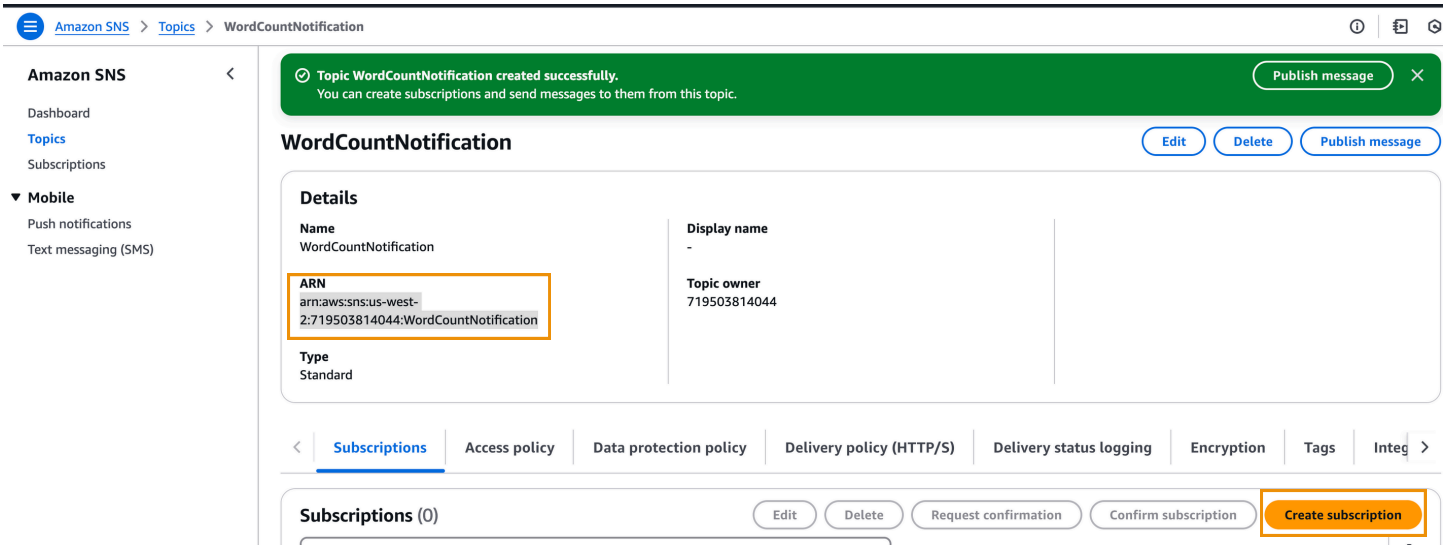
- Navigate to the **SNS** service in the **AWS Management Console**.
- Click "**Create topic**". Enter **Topic name** (**WordCountNotification**). Click **Next step.**



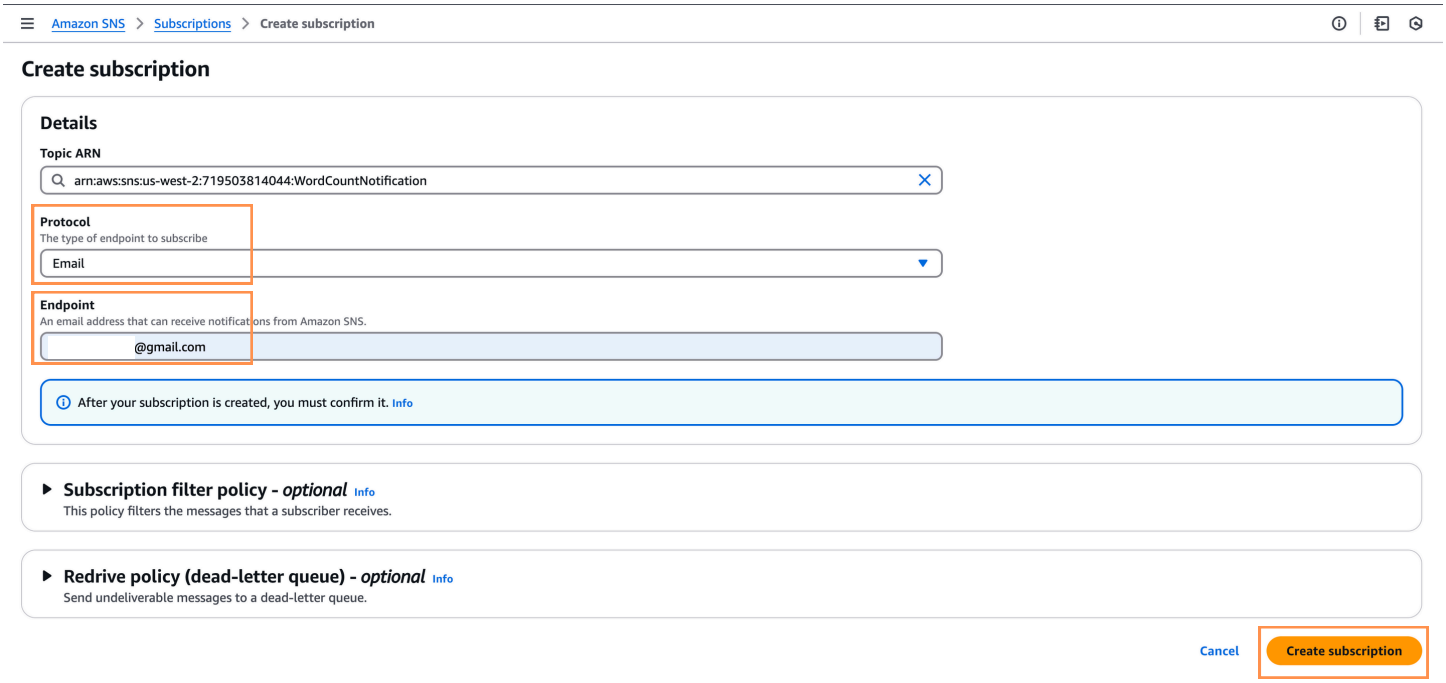
- Select "**Standard**" as the topic type. Click "**Create topic**".



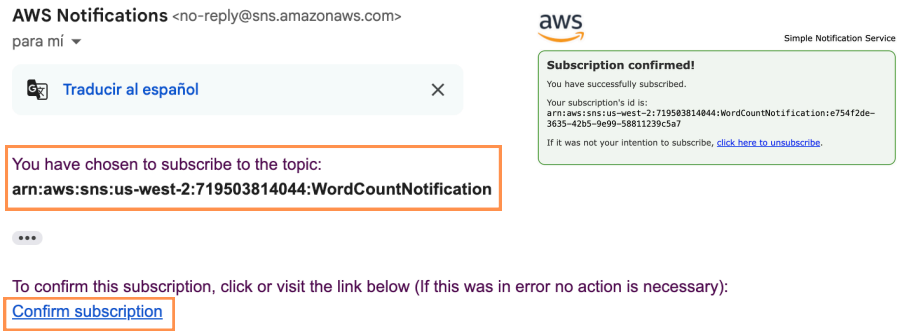
- Copy the SNS topic **ARN** (for later use).
- Click "**Create subscription**" to add an email recipient:



- **Protocol**: Email
- **Endpoint**: **My email address**.
- Click "**Create subscription**".



- I confirmed the **subscription email** by clicking the link in the verification email.

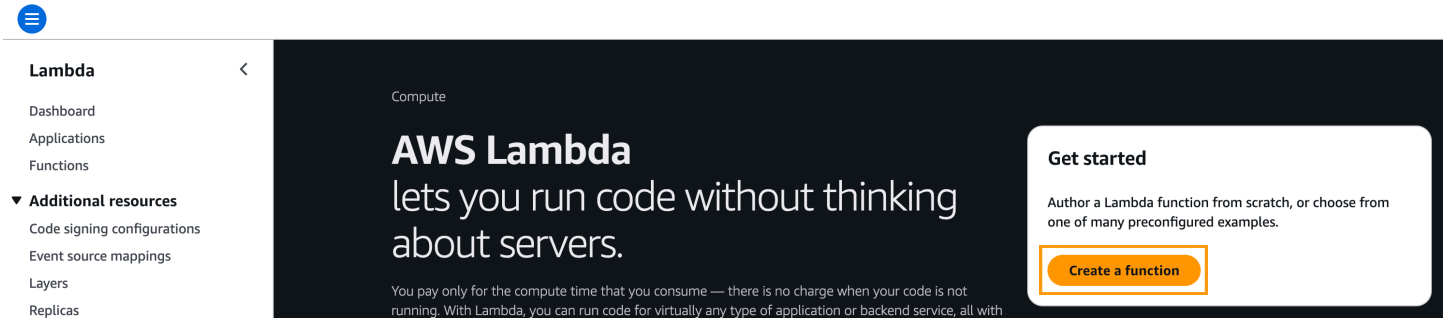


3 . Creating the Lambda Function

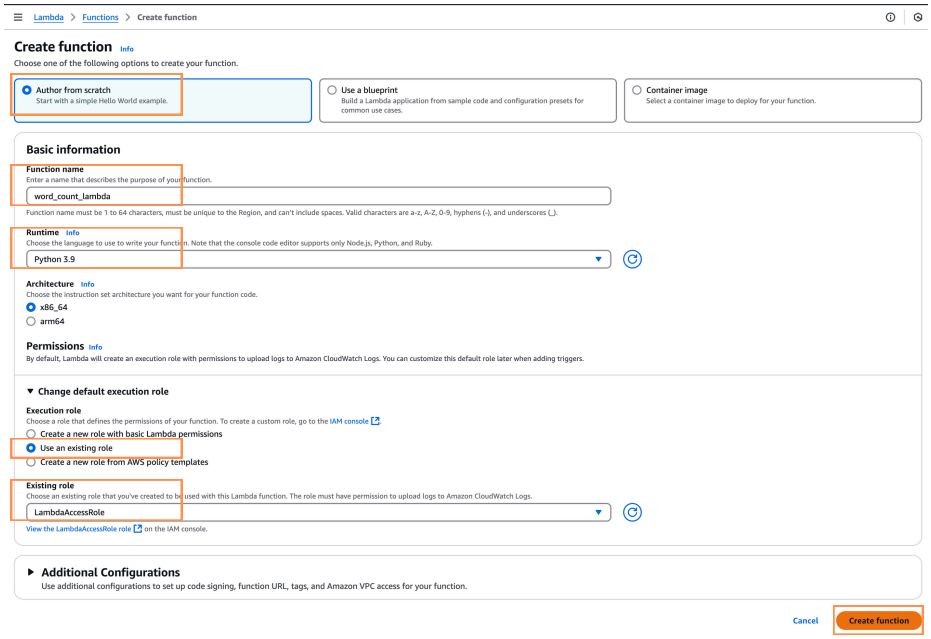
AWS **Lambda** is a serverless compute service that allows you to run code in response to events without provisioning or managing servers. You can trigger Lambda functions based on events from AWS services like S3, DynamoDB, and SNS, among others. It automatically scales by running code in response to triggers.

Steps:

- Go to **Lambda** in the AWS Management Console. Click **Create function**.

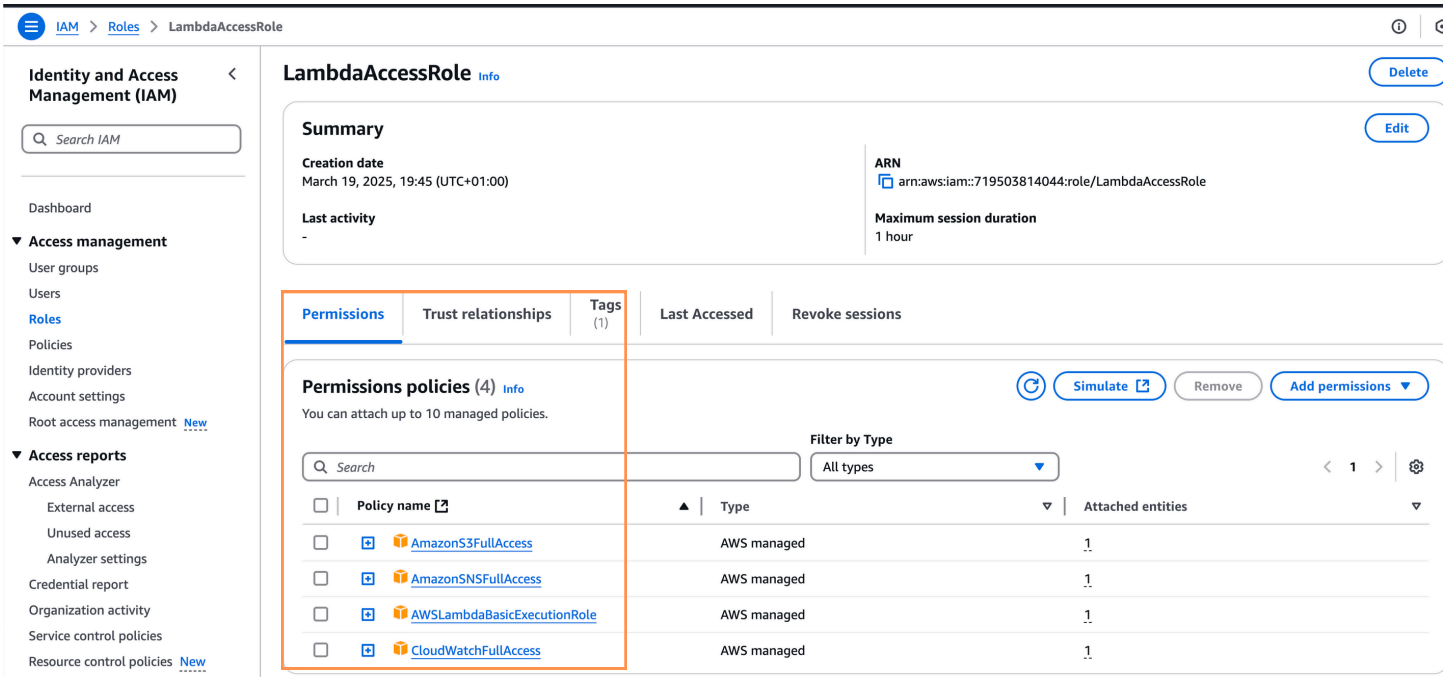


- Select **Author from scratch**.
- Enter the **Function name** (**word_count_lambda**).
- Select **Runtime**: Python 3.9 (or the latest available version).
- Under **Change default execution role**, select **Use an existing role**.
- Select the **LambdaAccessRole** (S3 and SNS access).
- Click **Create function**.



Notes: An AWS Identity and Access Management (IAM) role is required for the Lambda function to access other AWS services. For this project, I chose **LambdaAccessRole**, which provides the following permissions:

- AWSLambdaBasicExecutionRole: This is an AWS managed policy that provides write permissions to Amazon CloudWatch Logs.
- AmazonSNSFullAccess: This is an AWS managed policy that provides full access to Amazon SNS via the AWS Management Console.
- AmazonS3FullAccess: This is an AWS managed policy that provides full access to all buckets via the AWS Management Console.
- CloudWatchFullAccess



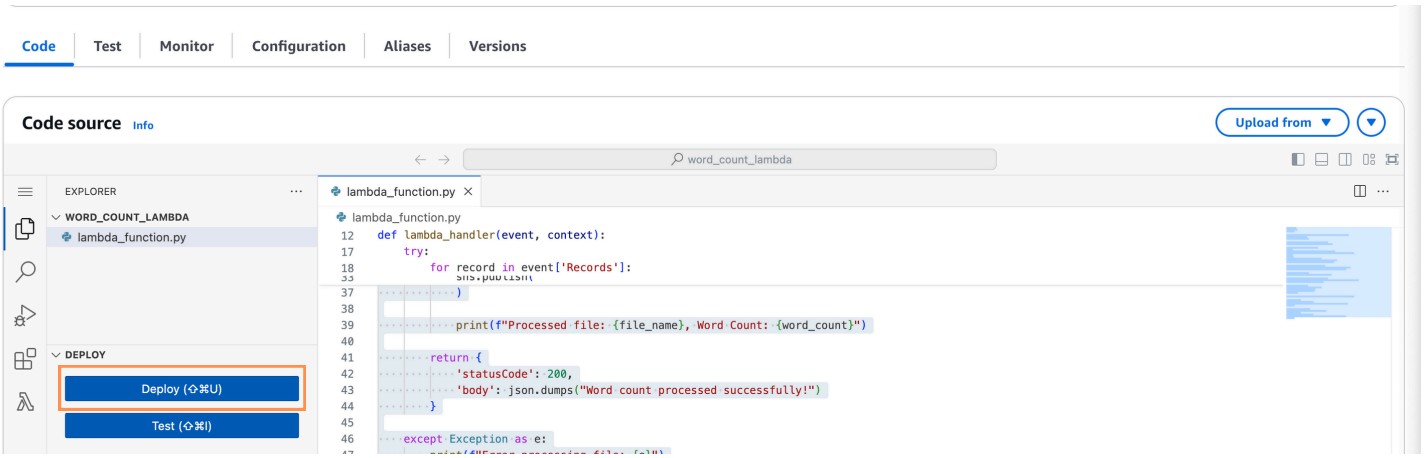
4 . Adding Code to the Lambda Function

Steps:

- Scroll down to the **Code source** section of the Lambda function configuration page.
- Click on the code and replace the default code with the following:

```python
import json
import boto3
import os

# Initialize AWS clients outside the function to optimize performance
s3 = boto3.client('s3')
sns = boto3.client('sns')

# Get SNS topic ARN from environment variables
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN', '')

def lambda_handler(event, context):
    """
    AWS Lambda function triggered by S3 file uploads.
    Counts the number of words in a text file and sends the result via SNS.
    """
    try:
        for record in event['Records']:
            bucket_name = record['s3']['bucket']['name']
            file_name = record['s3']['object']['key']

            # Retrieve file content from S3
            response = s3.get_object(Bucket=bucket_name, Key=file_name)
            file_content = response['Body'].read().decode('utf-8')

            # Count words
            word_count = len(file_content.split())

            # Create and send SNS notification
            message = f"The word count in the {file_name} file is {word_count}."
            subject = "Word Count Result"

            sns.publish(
                TopicArn=SNS_TOPIC_ARN,
                Message=message,
                Subject=subject
            )

            print(f"Processed file: {file_name}, Word Count: {word_count}")

        return {
            'statusCode': 200,
            'body': json.dumps("Word count processed successfully!")
        }

    except Exception as e:
        print(f"Error processing file: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps(f"Error processing file: {str(e)}")
        }
```

- Click **Deploy** button.



What does this Python code do?

This **AWS Lambda function** is triggered when a text file is uploaded to an **Amazon S3 bucket**. It **counts the number of words** in the file and sends the result via **Amazon SNS (Simple Notification Service)**.

- **AWS Clients Initialization (Outside the Function for Efficiency):**
  - **boto3**: AWS SDK for Python to interact with **S3** and **SNS**.
  - Initializing AWS clients **outside the function** reduces latency by reusing connections.

```
5  # Initialize AWS clients outside the function to optimize performance
6  s3 = boto3.client('s3')
7  sns = boto3.client('sns')
```

- **Fetching SNS Topic ARN from Environment Variables:**
  - **SNS topic ARN** is stored in an **environment variable**, making the function **more flexible** without hardcoding values.

```
9  # Get SNS topic ARN from environment variables
10 SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN', '')
11
```

- **Lambda Event Processing & Word Count Execution:**
  - Loops through S3 event records to process each uploaded file.
  - Extracts bucket name and file name from the event.

```
18         for record in event['Records']:
19             bucket_name = record['s3']['bucket']['name']
20             file_name = record['s3']['object']['key']
```

- **Reading File Content from S3:**
  - Retrieves the uploaded file from S3 and decodes it as text.

```
22             # Retrieve file content from S3
23             response = s3.get_object(Bucket=bucket_name, Key=file_name)
24             file_content = response['Body'].read().decode('utf-8')
```

- **Counting Words in the File:**
  - Splits the file content by spaces and counts words.

```
26             # Count words
27             word_count = len(file_content.split())
```

- **Sending Notification via SNS:**
  - Publishes an **SNS message** containing the **word count result**.

```
29             # Create and send SNS notification
30             message = f"The word count in the {file_name} file is {word_count}."
31             subject = "Word Count Result"
32
33             sns.publish(
34                 TopicArn=SNS_TOPIC_ARN,
35                 Message=message,
36                 Subject=subject
37             )
```

- **Logging & Success Response:**
  - Logs the processed file details and returns a 200 OK response.

```
            print(f"Processed file: {file_name}, Word Count: {word_count}")

        return {
            'statusCode': 200,
            'body': json.dumps("Word count processed successfully!")
        }
```

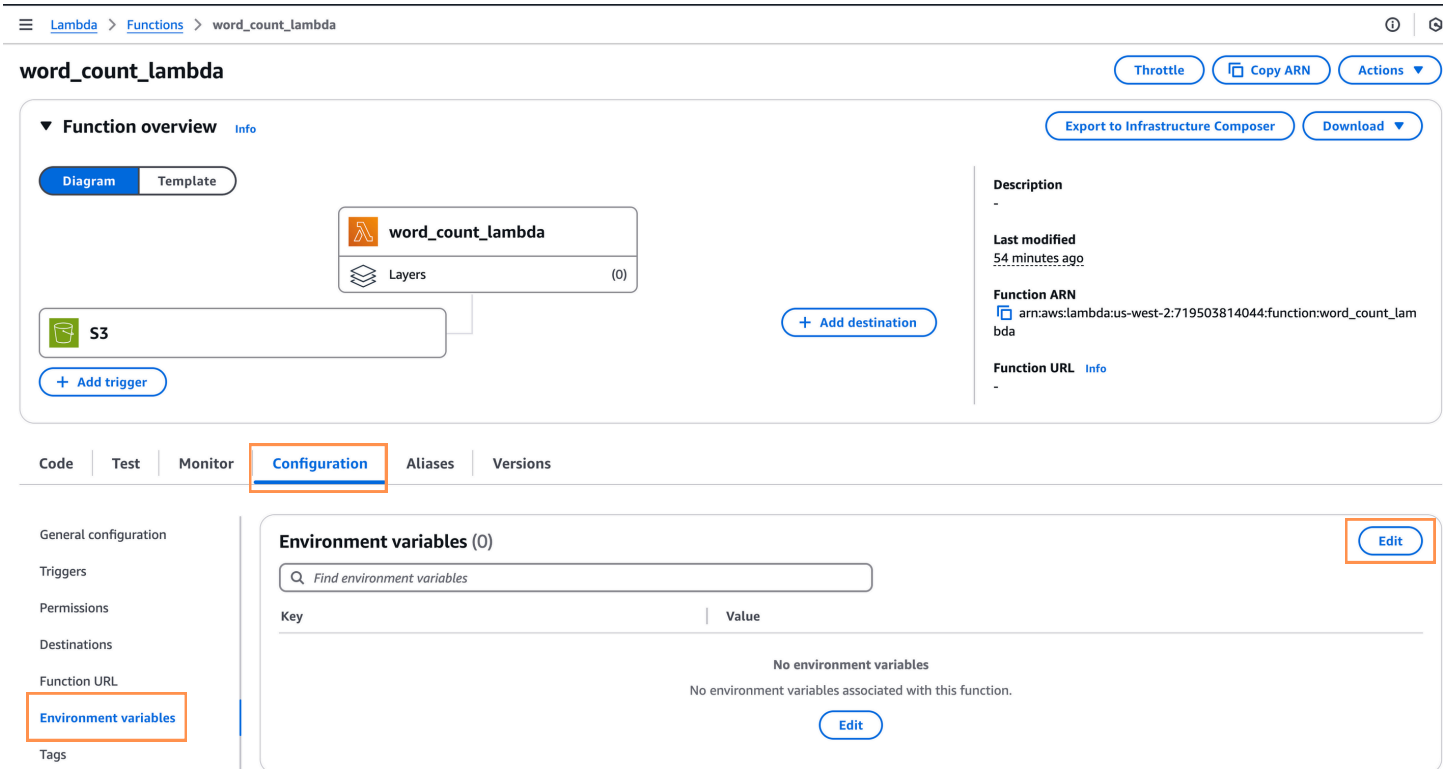
- **Error Handling:**
  - Catches errors and returns a 500 error response if the function fails.

```
    except Exception as e:
        print(f"Error processing file: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps(f"Error processing file: {str(e)}")
        }
```

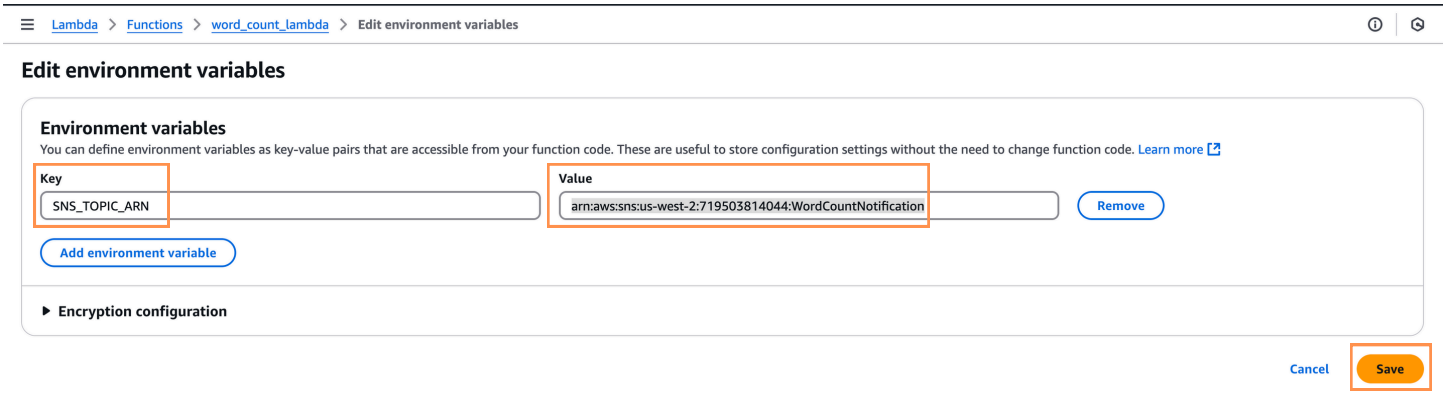
5 . Configuring Environment Variables in AWS Lambda

Configuring the **SNS topic ARN** as an **Environment variable** for the Lambda function:

- Select **Configuration** tab, in the left pane select **Environment variables**. Click the **Edit** button.



- **Add the SNS Topic ARN as an Environment Variable:**
  - **Key:** *SNS_TOPIC_ARN*
  - **Value:** *arn:aws:sns:us-west-2:719503814044:WordCountNotification* (the SNS topic ARN)
  - Click **Save** button.



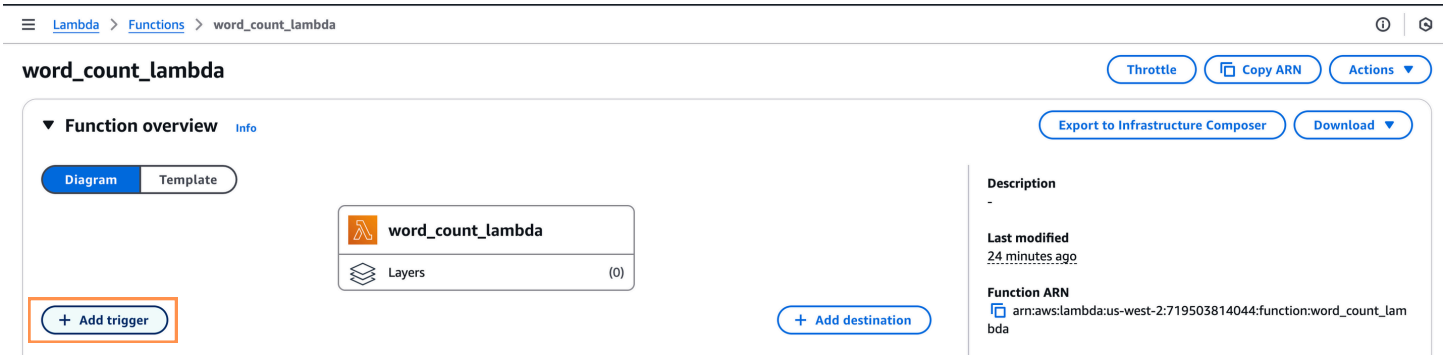
6 . Configuring the S3 Trigger

A **trigger** is an event that automatically invokes a function. In the case of AWS Lambda, a trigger can be an action performed by an AWS service, such as an S3 upload, a DynamoDB update, or an SNS notification. When the specified event occurs, it automatically executes the Lambda function to process the event.

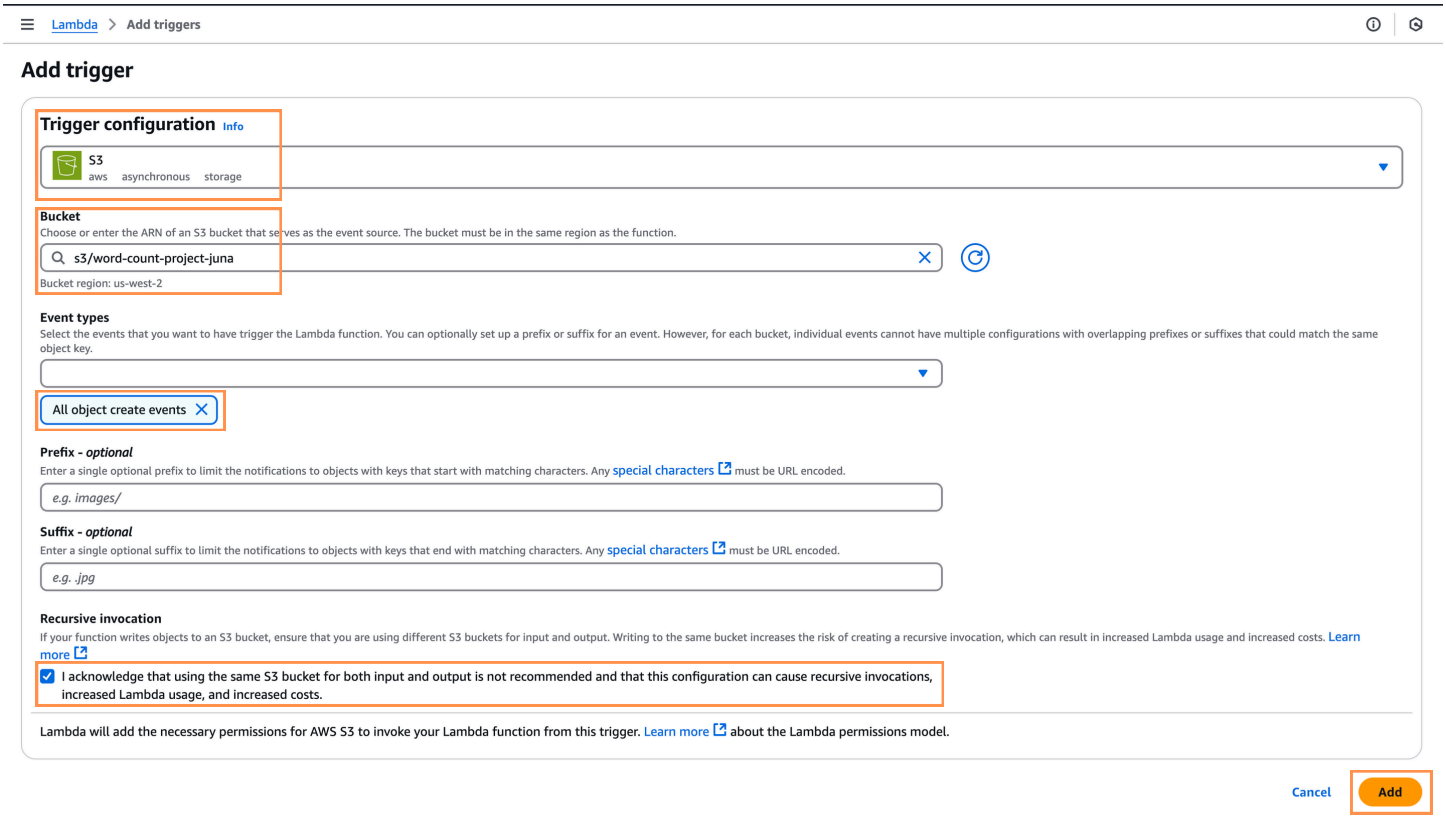
Configuring the Lambda function to be triggered when a file is uploaded to the S3 bucket:

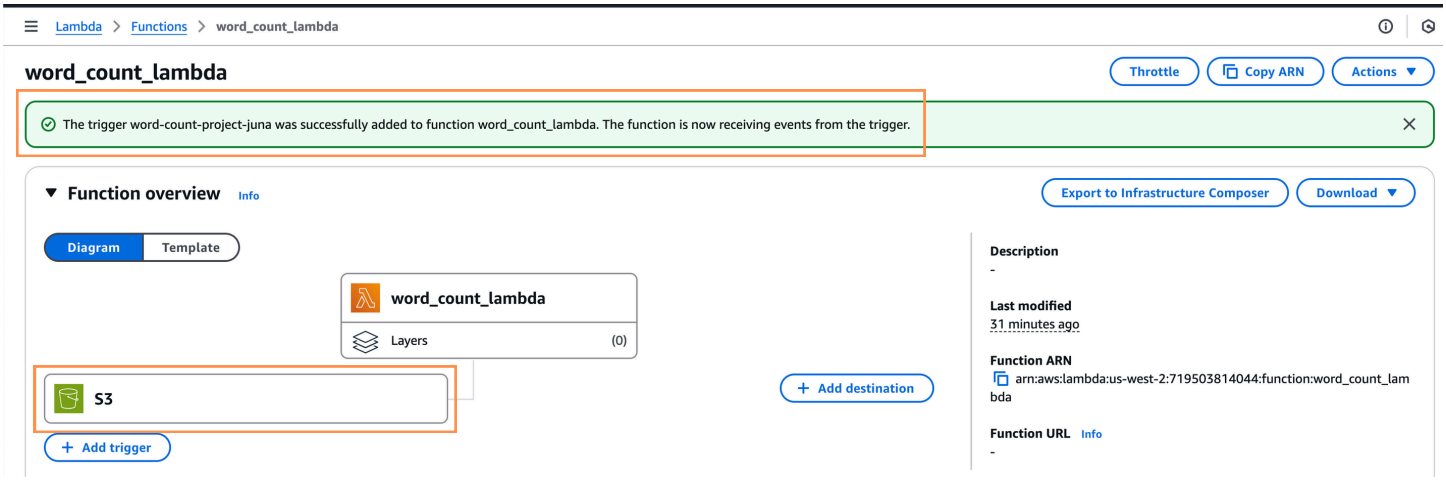
Steps:

- Go to your **Lambda**. Select word_count_lambda. Click **Add trigger**.



- Choose **S3** as the **source**.
- Bucket: Select the S3 bucket you created before.
- In **Event type**, select **PUT** (for new uploads or '**All object create events**').
- Click **Add**.



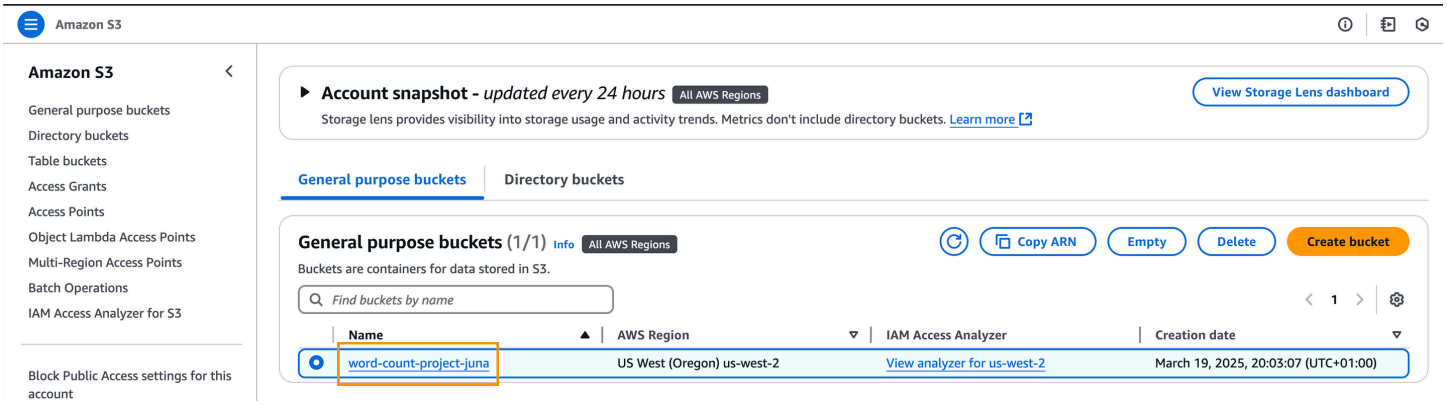

Lambda > Functions > word_count_lambda
word_count_lambda
Throttle
Copy ARN
Actions
The trigger word-count-project-juna was successfully added to function word_count_lambda. The function is now receiving events from the trigger.
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
word_count_lambda
Layers (0)
S3
Add destination
Add trigger
Description
-
Last modified
31 minutes ago
Function ARN
arn:aws:lambda:us-west-2:719503814044:function:word_count_lambda
Function URL Info
-

7 . Testing the Function

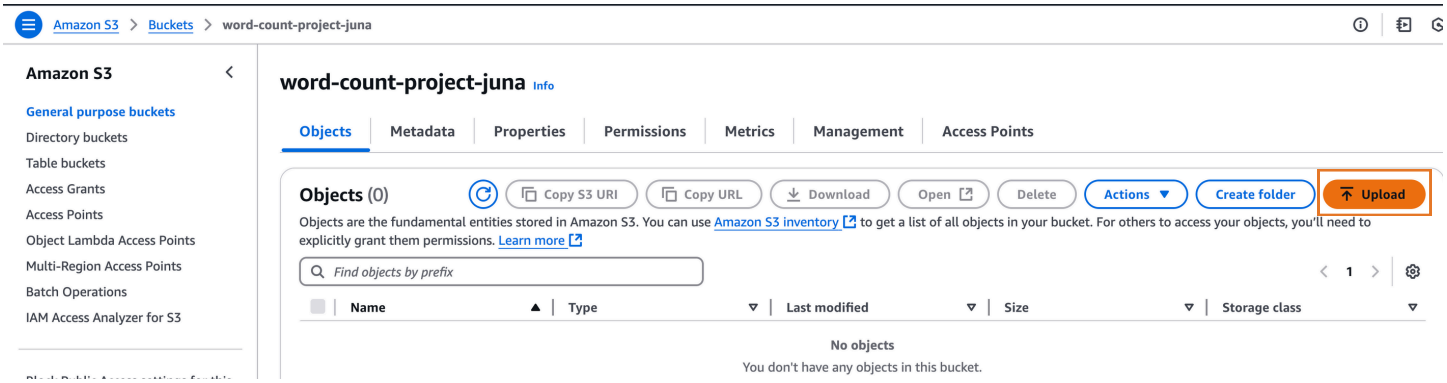
Testing ensures that the Lambda function is working as expected and correctly integrated with other AWS services, such as S3 and SNS. It helps identify issues early in the process, allowing you to troubleshoot and fix errors before the system is deployed in a production environment. Additionally, testing verifies that the function processes the input data correctly and triggers the expected notifications.

Steps:

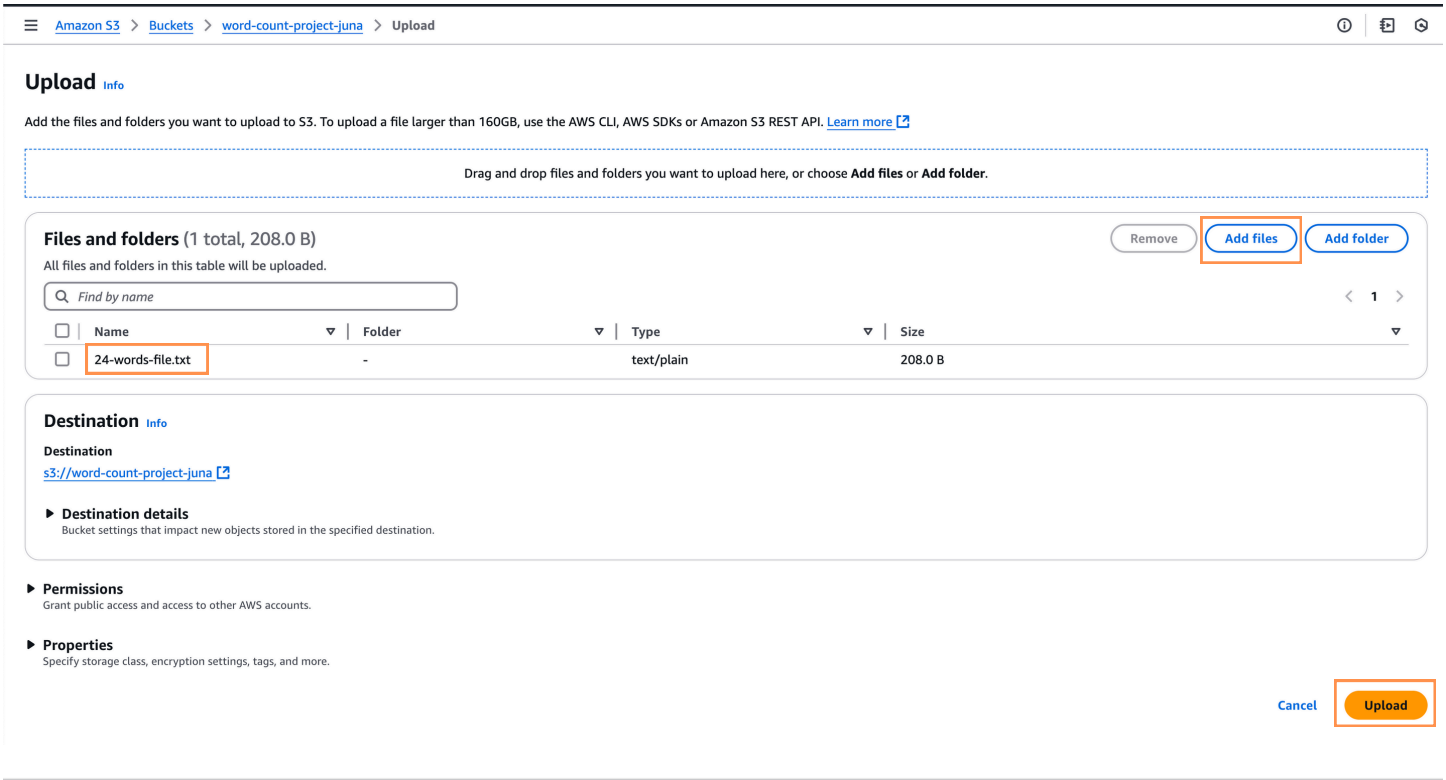
- **Upload a text file to the S3 bucket:**
  - Go to **S3** and select your bucket.

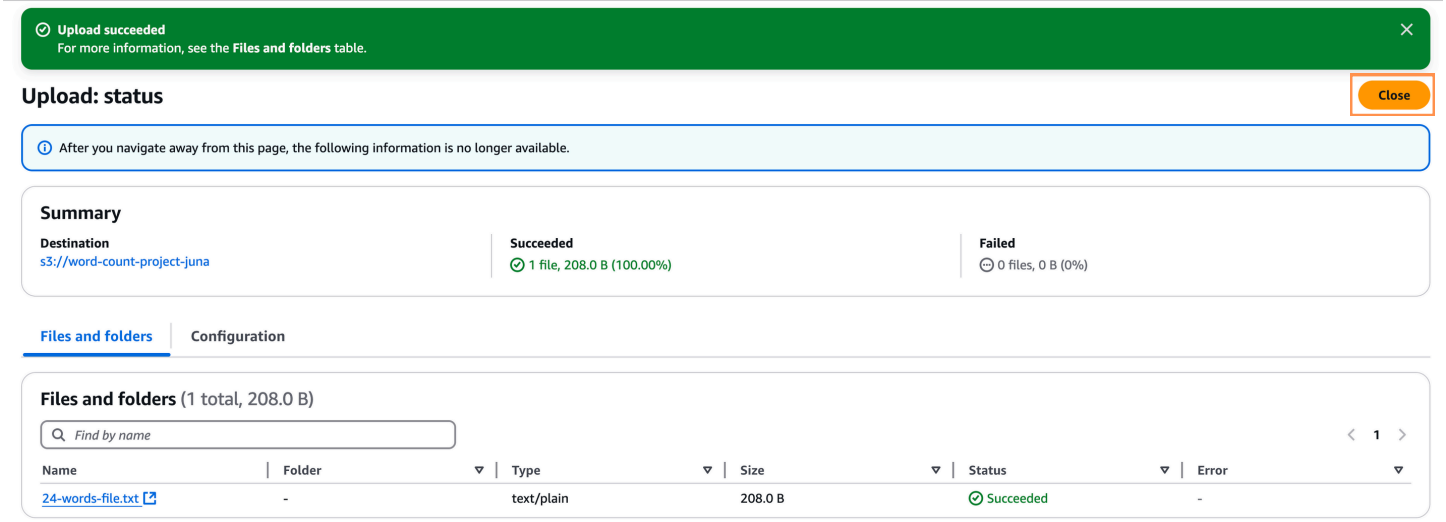


  - Click **Upload**.

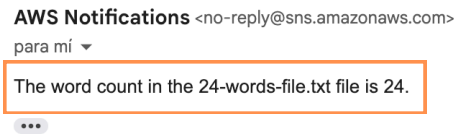


- Click **Add Files**, choose a text file and click **Upload**.





- **Check SNS Notifications:**
  - As expected, I received an email with the word count result.



Conclusions & Lessons Learned

1. Serverless Computing Simplifies Deployment

No need to manage servers, scaling, or infrastructure.

2. Event-Driven Automation is Efficient

AWS Lambda automatically processes files as soon as they are uploaded.

3. Seamless AWS Service Integration

S3, Lambda, and SNS work together for a fully automated pipeline.

4. Security & Best Practices Matter

IAM roles must be properly configured for Lambda, S3, and SNS.
SNS email subscriptions require user confirmation.

Final Thoughts

This project highlights the **efficiency and scalability** of **AWS Serverless Computing**. By leveraging **Lambda, S3, and SNS**, I successfully **automated word count processing** while eliminating the need for traditional server management.