Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

Case Study

Serverless Image Processing Workflow

Name:Urvashi Changlani

D15B/08

Introduction:

This guide outlines the steps to create a serverless image processing system using AWS services. By leveraging Amazon S3, AWS Lambda, CodeBuild, and CodePipeline, the system automates the process of triggering a Lambda function whenever an image is uploaded to an S3 bucket. This setup provides an efficient, scalable, and fully automated solution for processing images in the cloud, with continuous deployment handled through AWS CodePipeline.

Problem Statement:

"Create a serverless workflow that triggers an AWS Lambda function when a new image is uploaded to an S3 bucket. Use AWS CodePipeline to automate the deployment of the Lambda function."

Tools and Concepts Used:

- **AWS Lambda**: To execute the image processing function.
- **Amazon S3**: To store the uploaded images and trigger the Lambda function.
- **AWS CodePipeline**: To automate the deployment of updates to the Lambda function.

Key Features:

- **Serverless Architecture**: No need to manage servers, as Lambda automatically scales and executes in response to S3 events.
- **Automated Deployment**: AWS CodePipeline continuously deploys any updates made to the Lambda function code.

Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

- **Event-Driven Workflow**: The system is triggered only when an image is uploaded to the S3 bucket, ensuring efficiency.

Application:

- **Image Processing**: Automatically processes or logs images when uploaded to S3.
- **Efficient Automation**: Ensures seamless integration between S3, Lambda, and CodePipeline to automate workflows without manual intervention.
- **Scalability**: Designed to handle varying image uploads, scaling automatically based on demand.

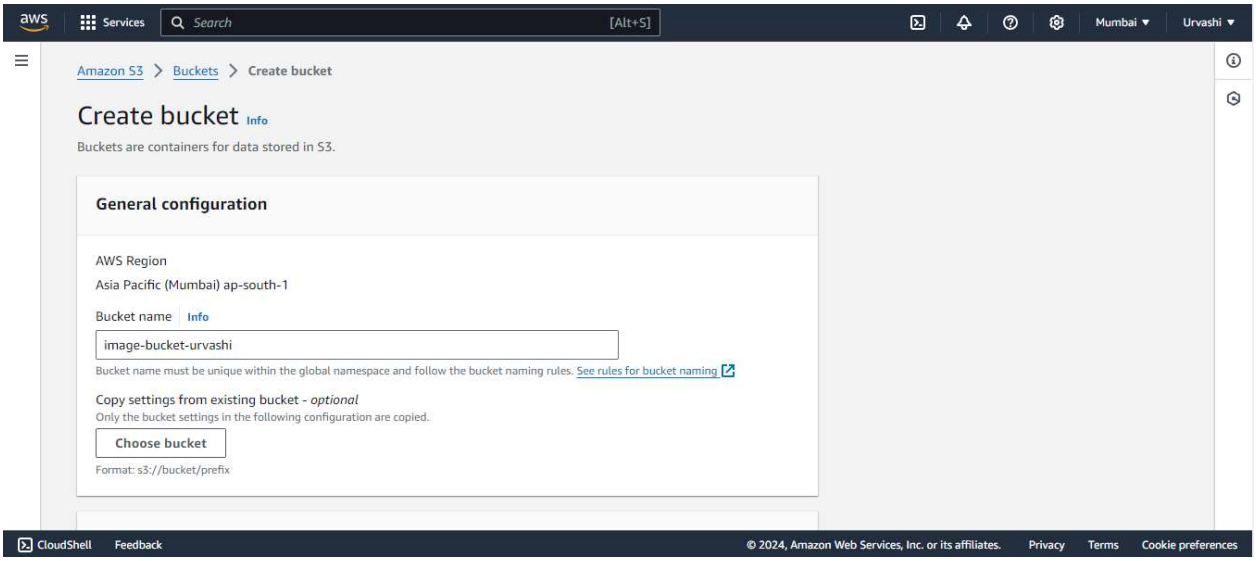
Steps:

Step 1: Set Up an S3 Bucket

1. **Log in to AWS Console**:
    a. Access your AWS account using your login credentials.
2. **Navigate to the S3 Service**:
    a. In the AWS Management Console, search for "S3" in the search bar and select it from the results.
3. **Create a New S3 Bucket**:
    a. Click the "Create Bucket" button.
    b. **Bucket Name**: Provide a unique name for your bucket (e.g., `image-bucket-urvashi`). Bucket names must be globally unique, so ensure the name hasn't been taken by another AWS user.
    c. **Region**: Select the AWS region where you want to store the data (e.g., `us-east-1`).
4. **Configure Optional Settings**:
    a. **Versioning**: You can enable versioning to keep multiple versions of objects (files) in the bucket. This helps track changes and recover earlier versions.
    b. Leave other options as default, such as encryption and object ownership settings.
5. **Create the Bucket**:
    a. After configuring, click "Create Bucket." Your S3 bucket is now ready for use.

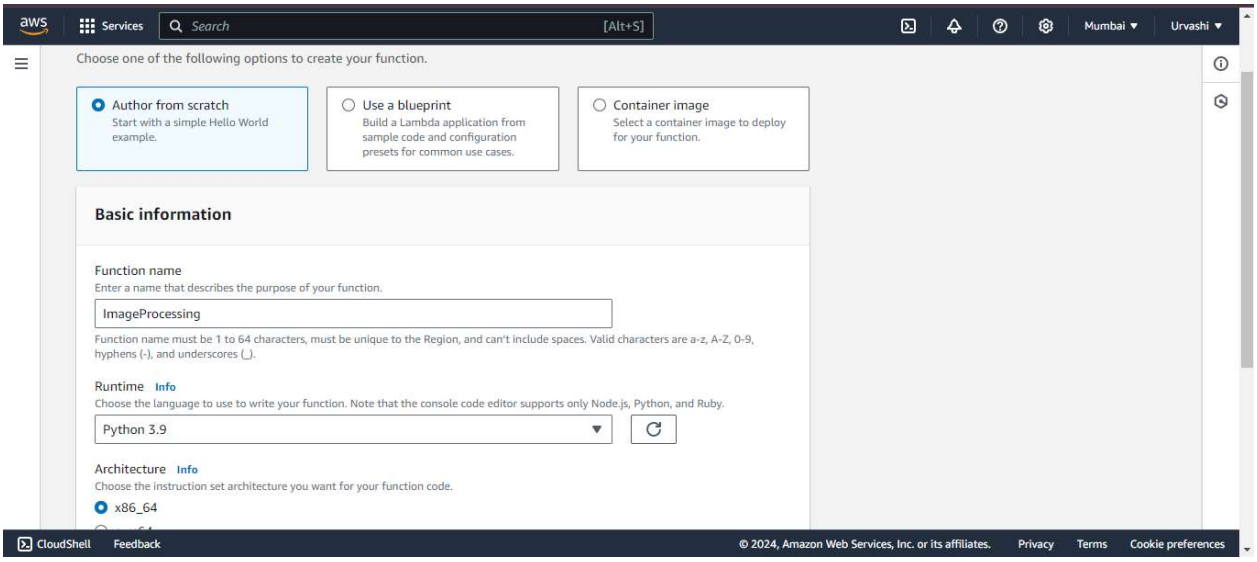
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



Step 2: Create a Lambda Function to Process Images

1. Go to the **Lambda service**, click **Create Function**, and select **Author from Scratch**.
2. Name it `ImageProcessingLambda`, choose **Python 3.x**, and create a new IAM role with **AWSLambdaBasicExecutionRole**.
3. Add **AmazonS3FullAccess** and **CloudWatchLogsFullAccess** to the Lambda execution role.
4. Add Python code to log events and process the image.

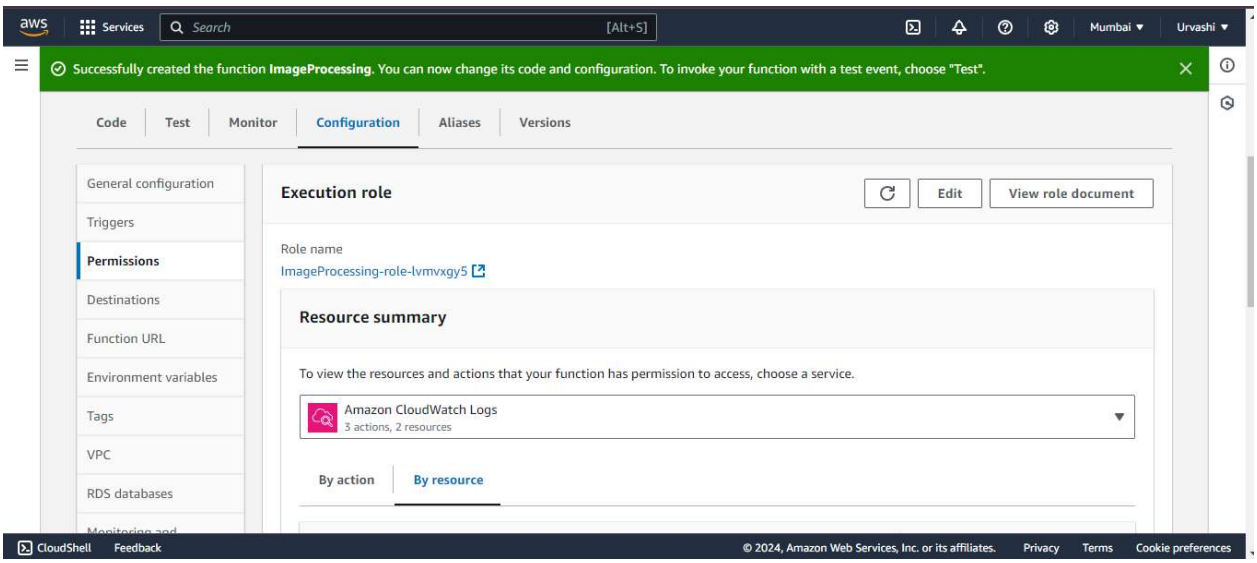


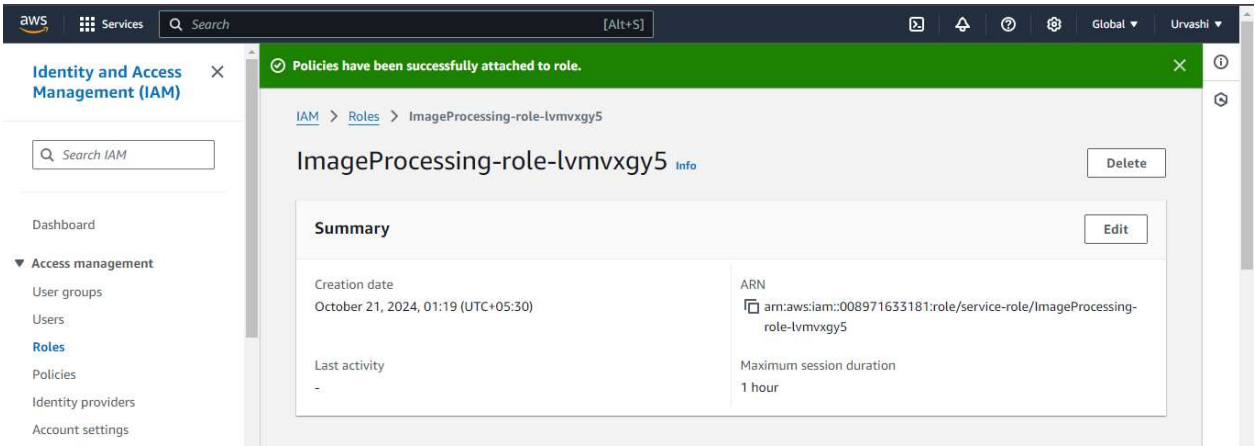
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

Set IAM Role for Lambda:

- **IAM Role**: Create a new role with basic Lambda permissions.
  - Select "Create a new role with basic Lambda permissions."
  - This will automatically assign the policy `AWSLambdaBasicExecutionRole` to the role, which allows the Lambda function to write logs to Amazon CloudWatch





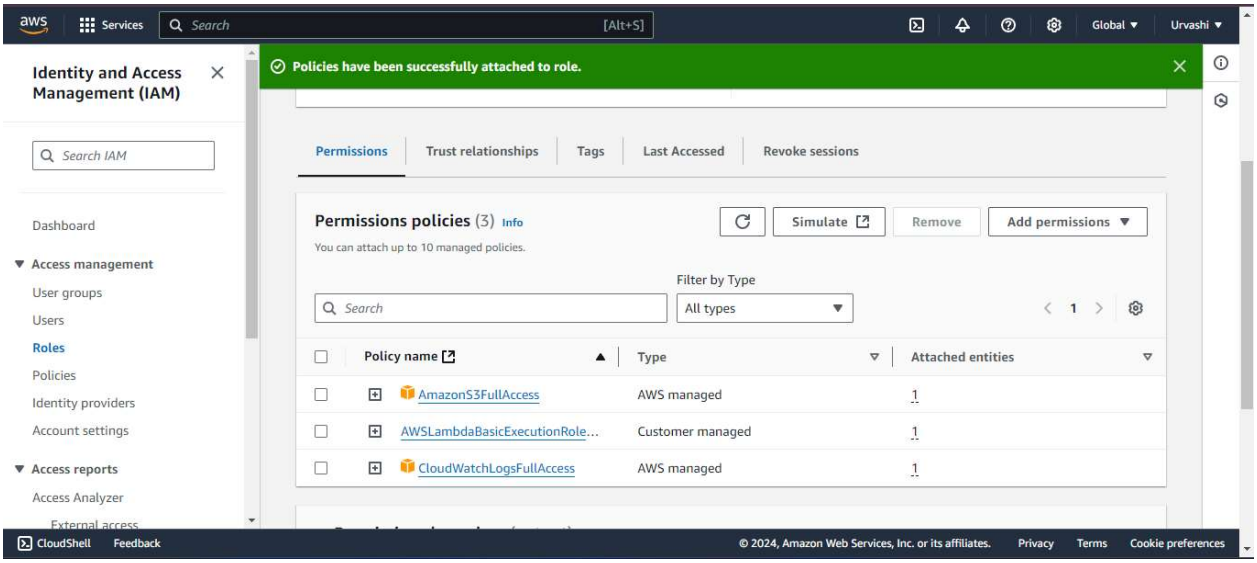
After the function is created, add the following permissions to access the S3 bucket:

Click on the role and attach the following permissions:

- AmazonS3FullAccess
- CloudWatchLogsFullAcces

Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



Add Python Code to the Lambda Function To Process images:

```
import json
import boto3
 def lambda_handler(event, context):
# Log the event in CloudWatch
   print("Event: ", json.dumps(event))

   # Extract S3 bucket and object details
   s3 = boto3.client('s3')
   bucket = event['Records'][0]['s3']['bucket']['name']
   key = event['Records'][0]['s3']['object']['key']

   # Process the image (in this example, we are simply logging its details)
   response = s3.get_object(Bucket=bucket, Key=key)
   print(f"Processing file {key} from bucket {bucket}")

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

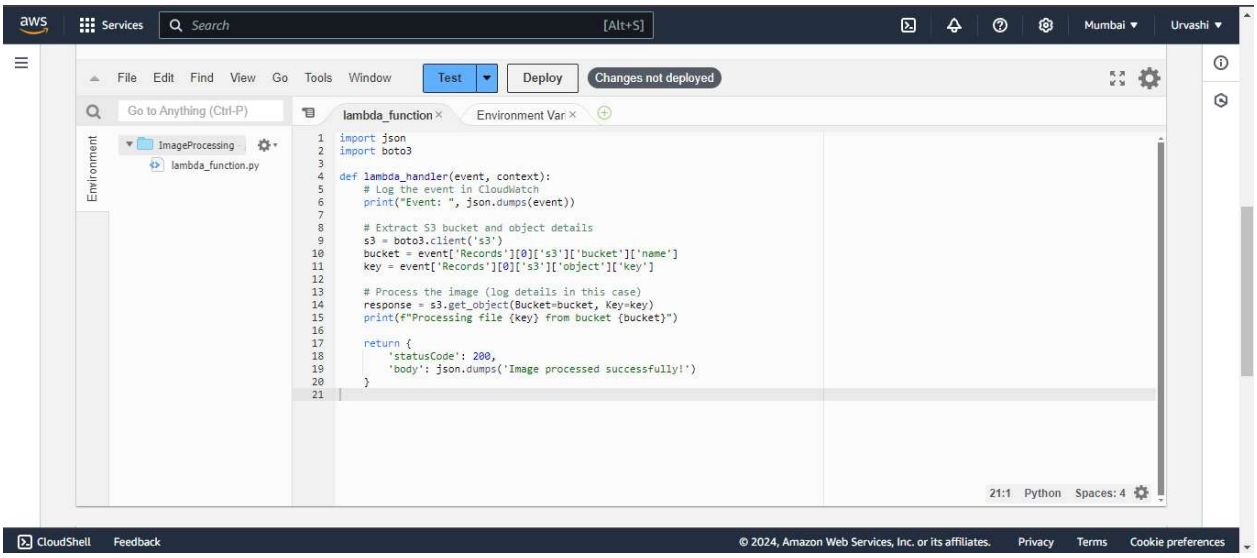
This code:

- Logs the event details in CloudWatch.

Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

- Extracts the S3 bucket name and the file (key) that triggered the Lambda function.
- Logs the file name and bucket to show that the image has been processed.



Step 3: Set Up S3 Event Notification to Trigger Lambda

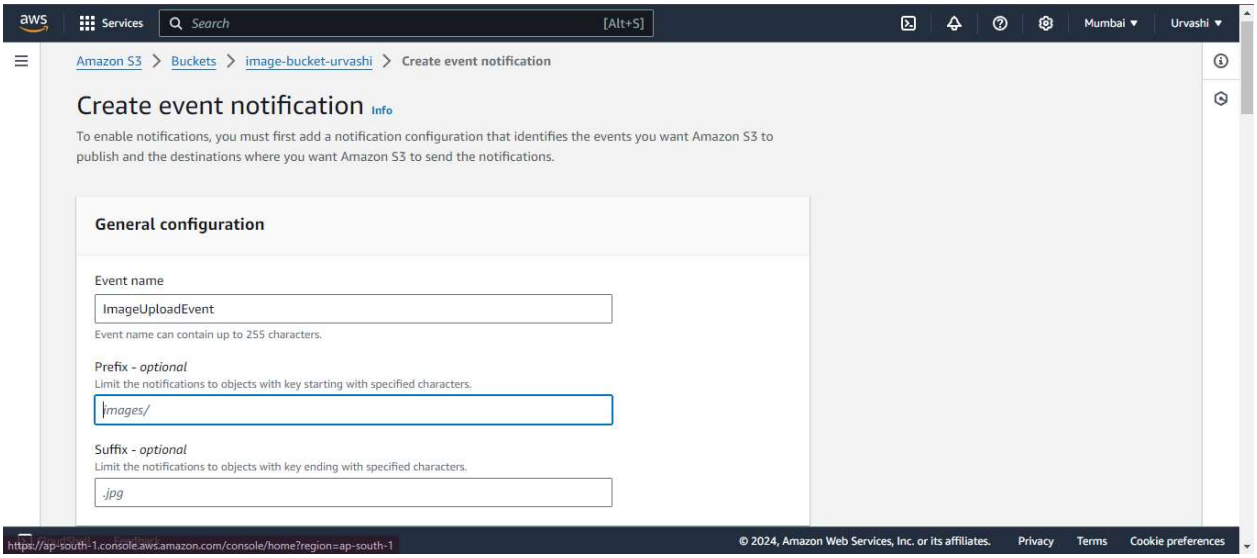
1. Click **Create Event Notification**.
2. Set **Event Name** (e.g., `ImageUploadEvent`).
3. **Event Type**: Select **All object create events**.
4. **Destination**: Choose **Lambda Function** and select `ImageProcessing`.

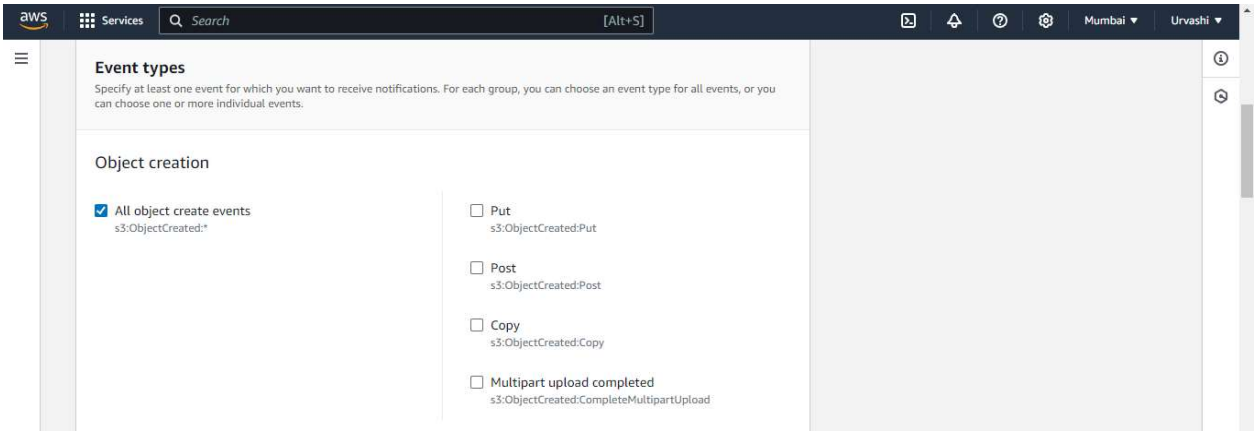
Save the Notification:

- Click **Save Changes**.

Urvashi Changlani

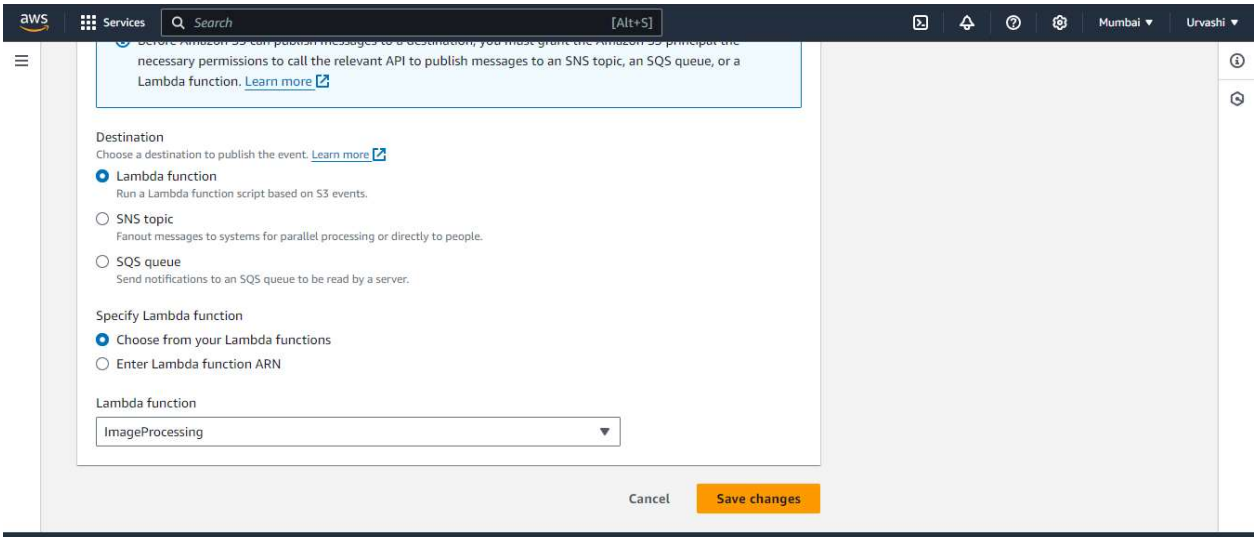
2022.urvashi.changlani@ves.ac.in

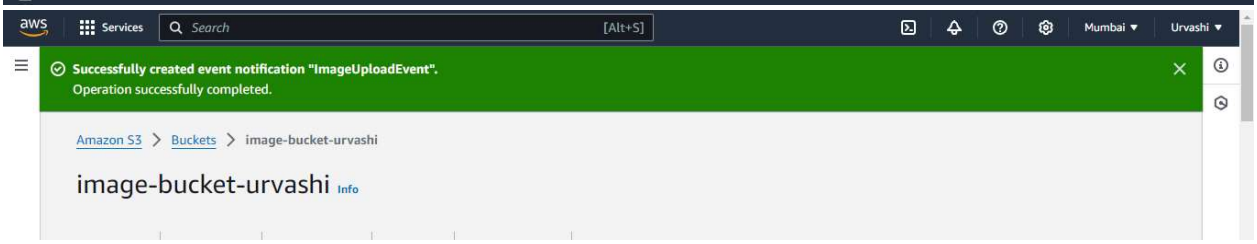




Urvashi Changlani

2022.urvashi.changlani@ves.ac.in





Step 4: Use CodeBuild for Packaging and Deploying Lambda:

Create a Buildspec File:

- In your GitHub repository (where your `lambda_function.py` is stored), add a `buildspec.yml` file. This file instructs CodeBuild on how to package and deploy your Lambda function.

Buildspec.yml

```
version: 0.2
phases:
  install:
    commands:
      - pip install --upgrade awscli
  build:
    commands:
      - zip function.zip lambda_function.py
```

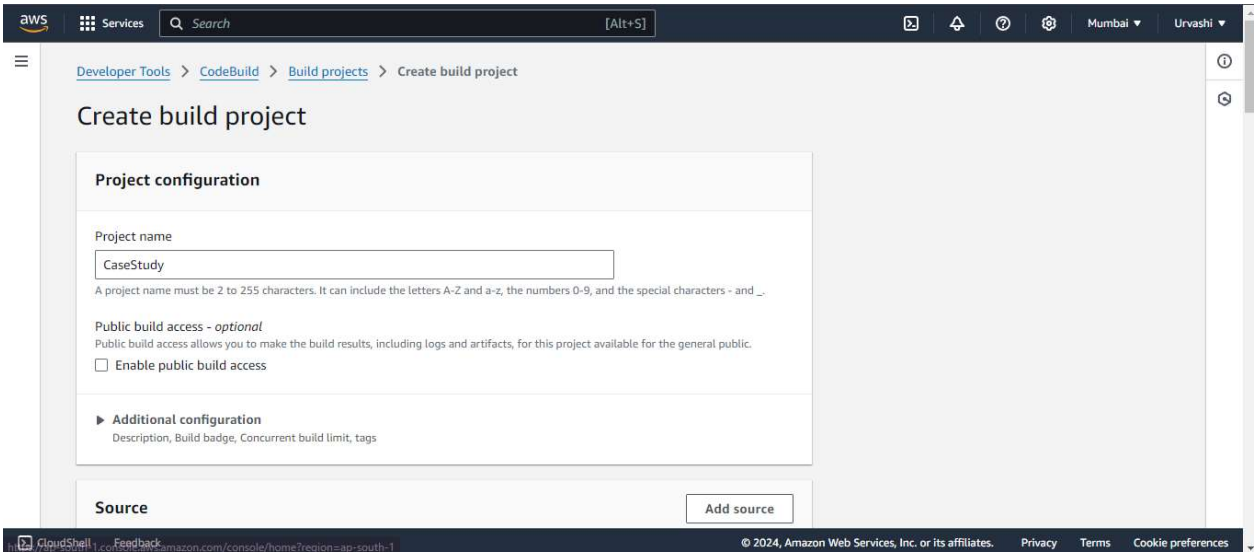
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

- aws lambda update-function-code --function-name ImageProcessingLambda --zip-file fileb://function.zip.

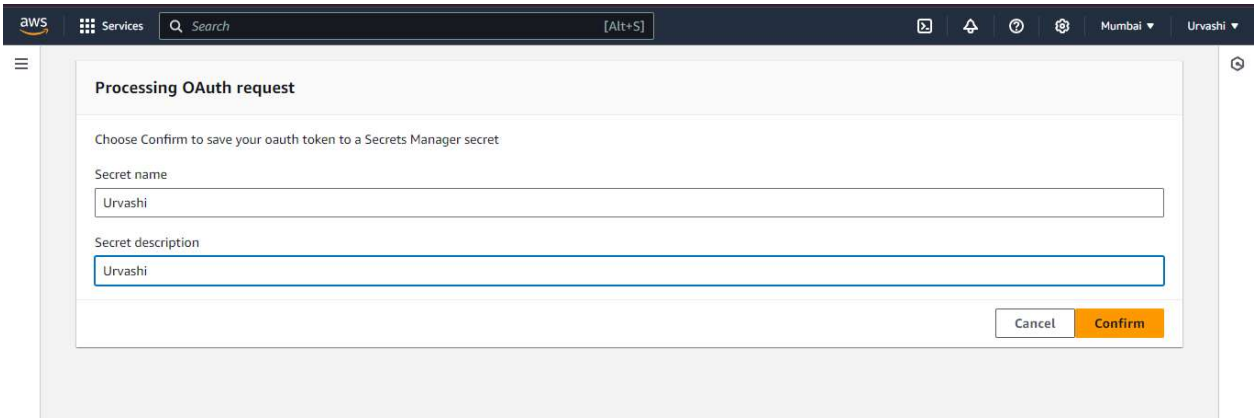
Create a CodeBuild Project:

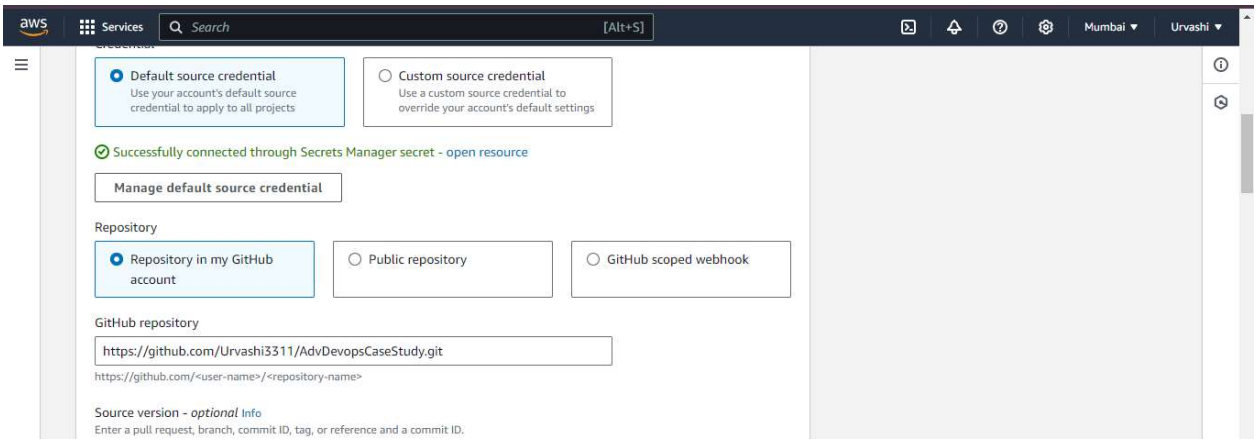
- In AWS Console, go to the CodeBuild service and create a new build project.
- **Source**: Select the same GitHub repository where your code is stored.
- **Environment**: Choose a managed image (e.g., Ubuntu with standard runtimes). Ensure the environment has the necessary permissions (like `AWSLambdaFullAccess`) to update the Lambda function.
- Specify the `buildspec.yml` file in your repository.

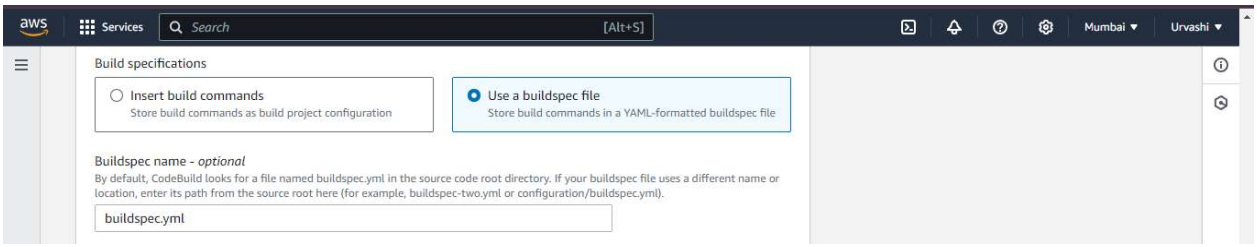


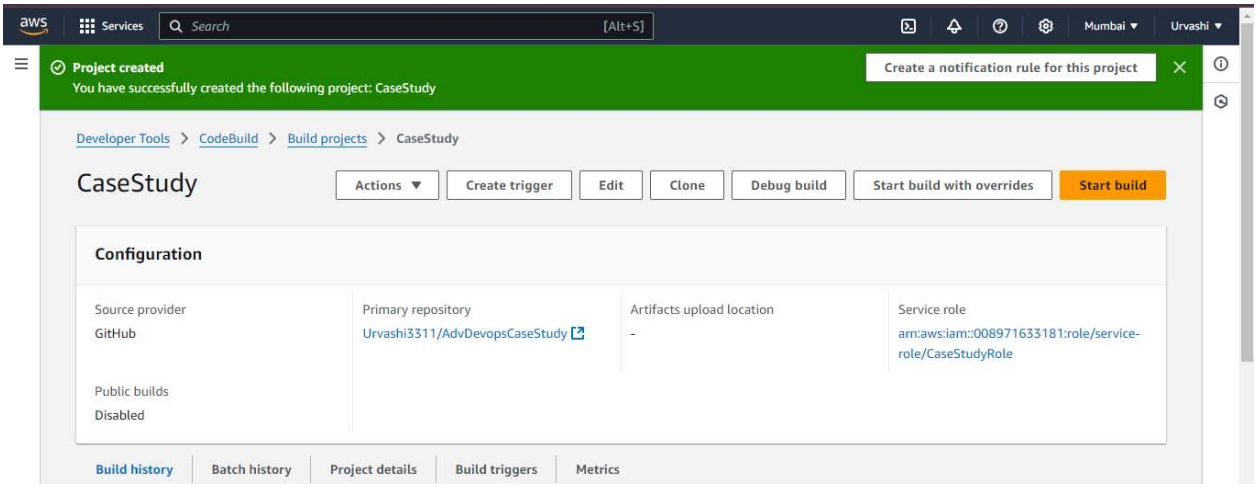
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



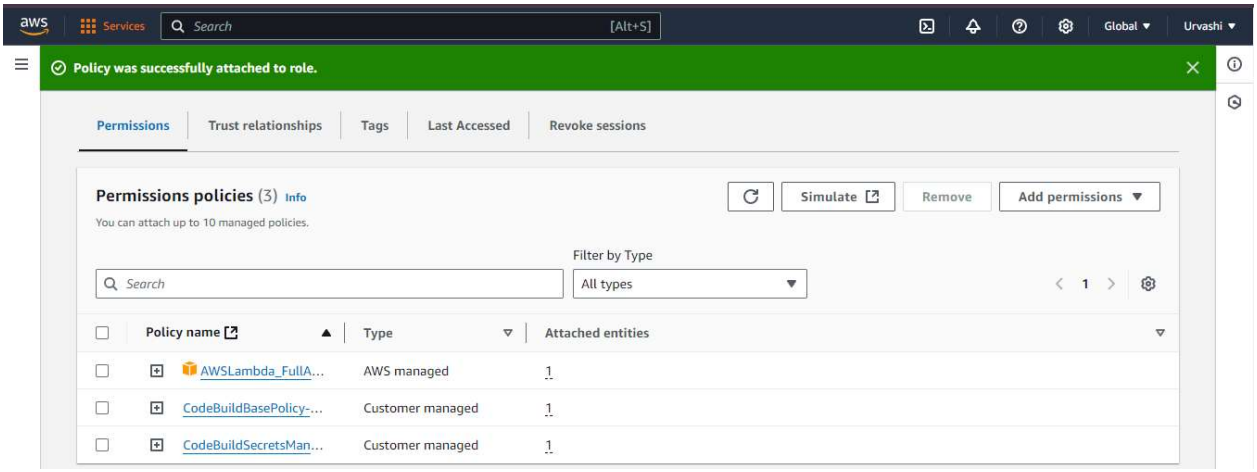






Urvashi Changlani

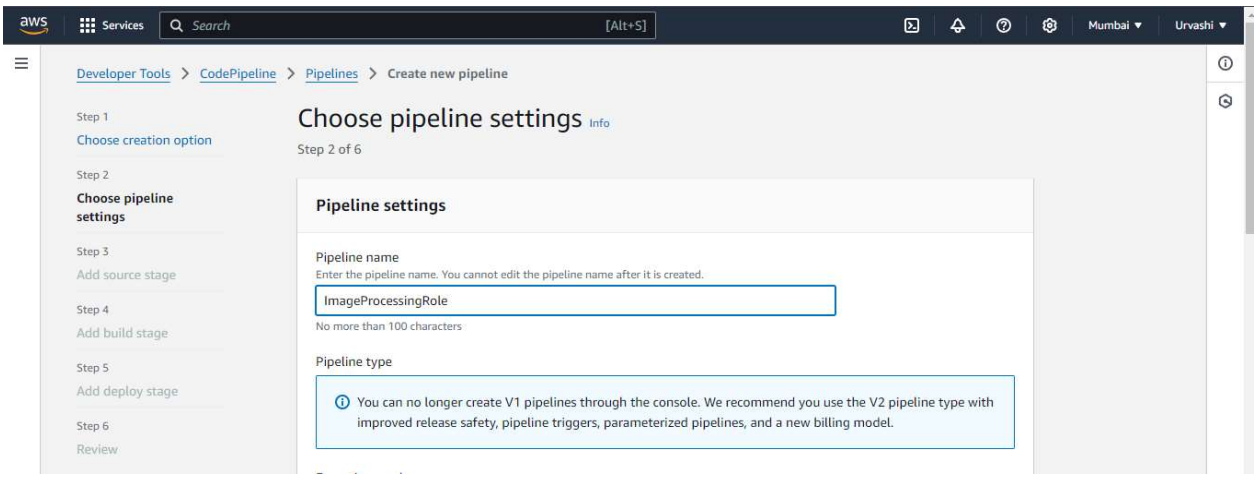
2022.urvashi.changlani@ves.ac.in



Step 5: Automate Lambda Deployment Using CodePipeline

Create a New CodePipeline:

- Go to the CodePipeline service in AWS and click "Create Pipeline."
- **Pipeline Name**: Name your pipeline (e.g., `ImageProcessingPipeline`).
- **Service Role**: Allow CodePipeline to create a new role automatically.



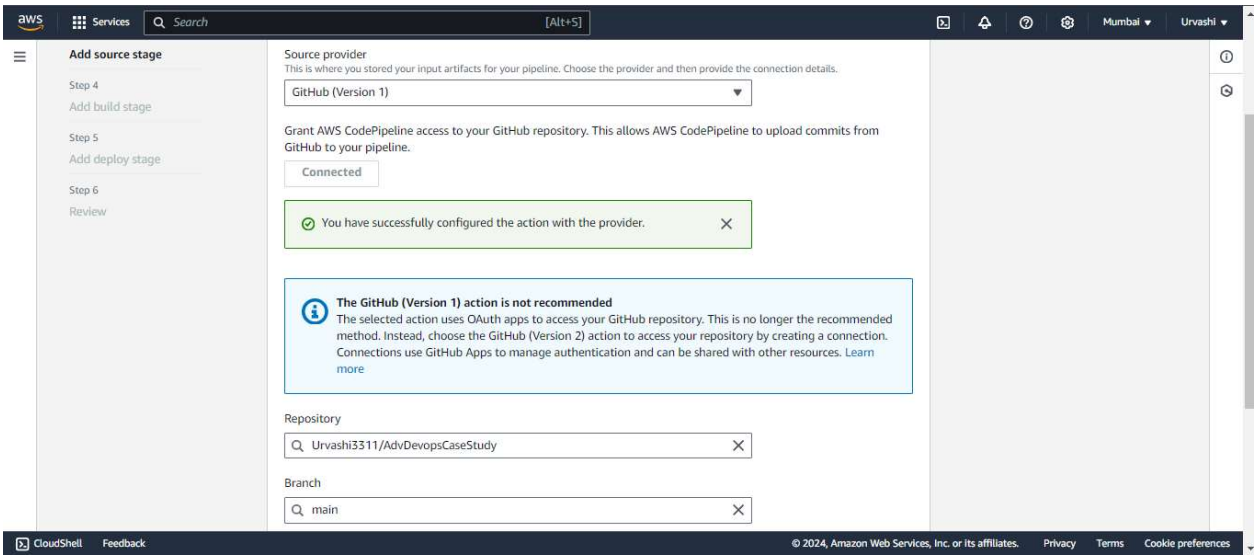
Add Source Stage:

- **Source Provider**: Choose GitHub (or CodeCommit if using AWS's repository service).

Urvashi Changlani

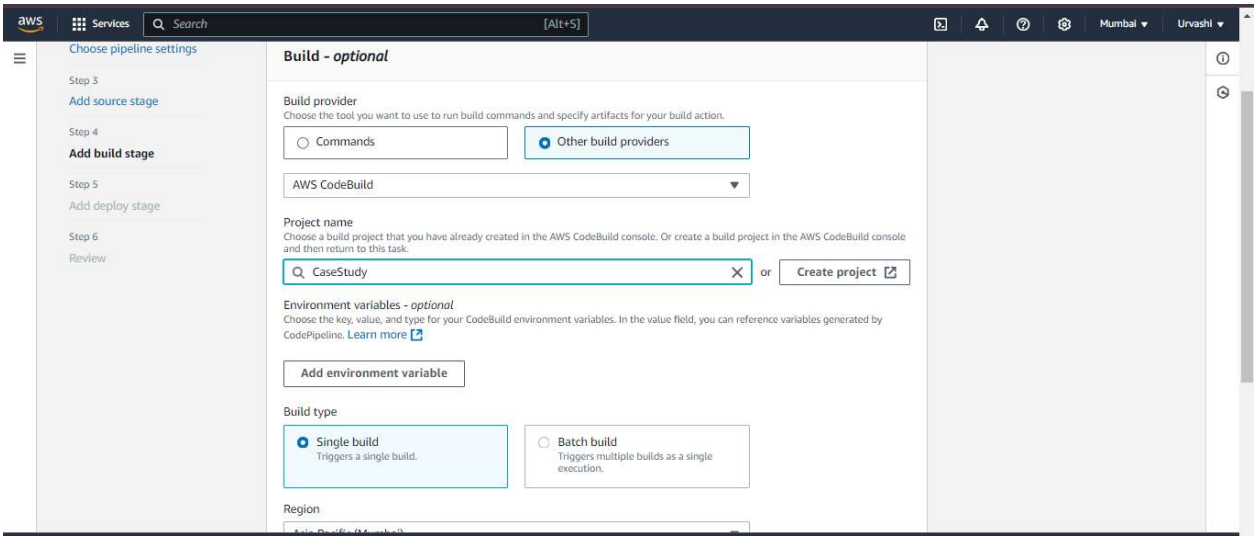
2022.urvashi.changlani@ves.ac.in

- **Repository**: Connect your repository that contains the Lambda code (the same repository as used in Step 2).



Add CodeBuild to CodePipeline:

a. Add CodeBuild as the **Build Stage**. CodePipeline will trigger the CodeBuild project, which will execute the `buildspec.yml` to package and deploy the Lambda function.

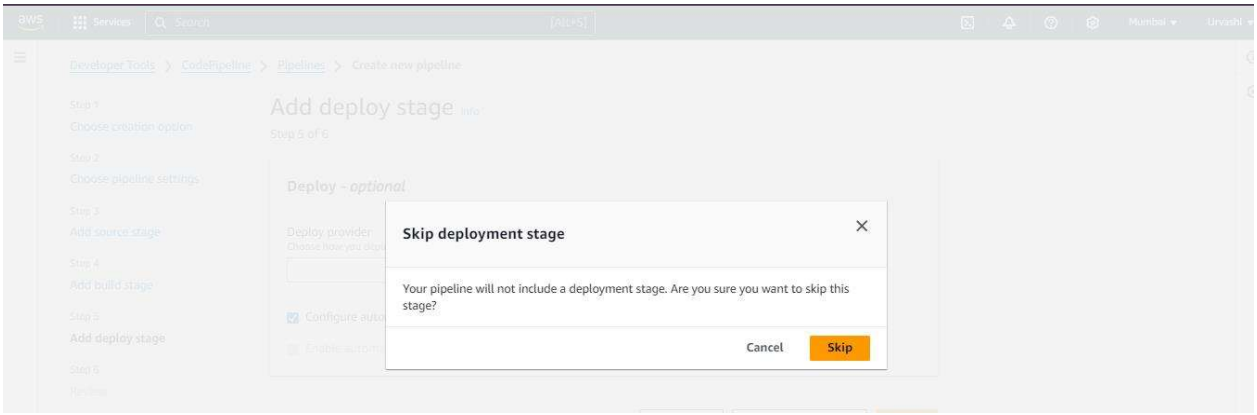


Deploy Stage (Deploy to Lambda):

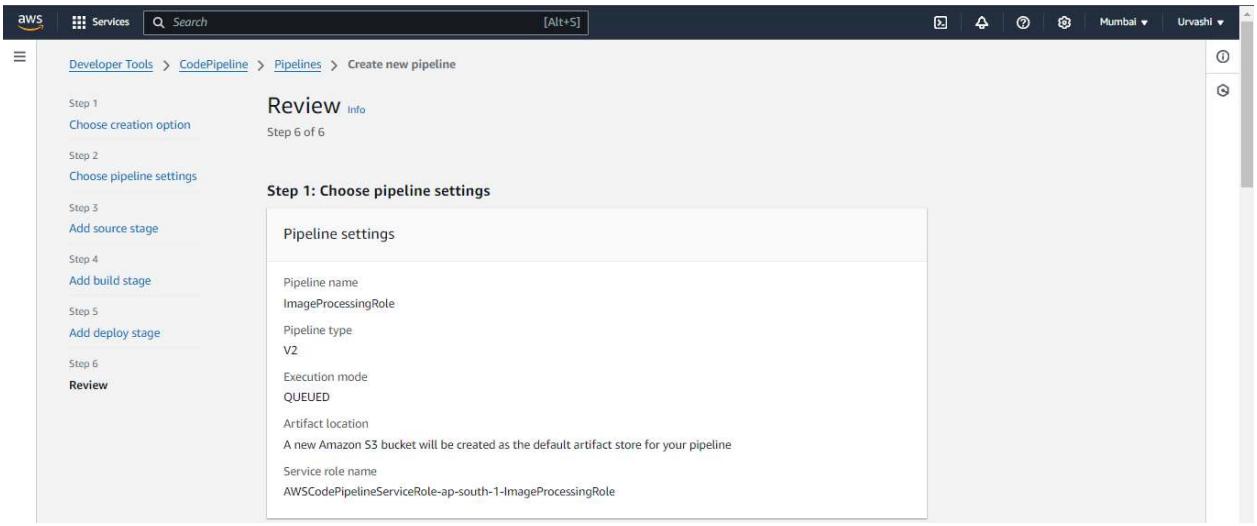
SKIP THIS (as Choose AWS Lambda as the deploy provider Does not exist.

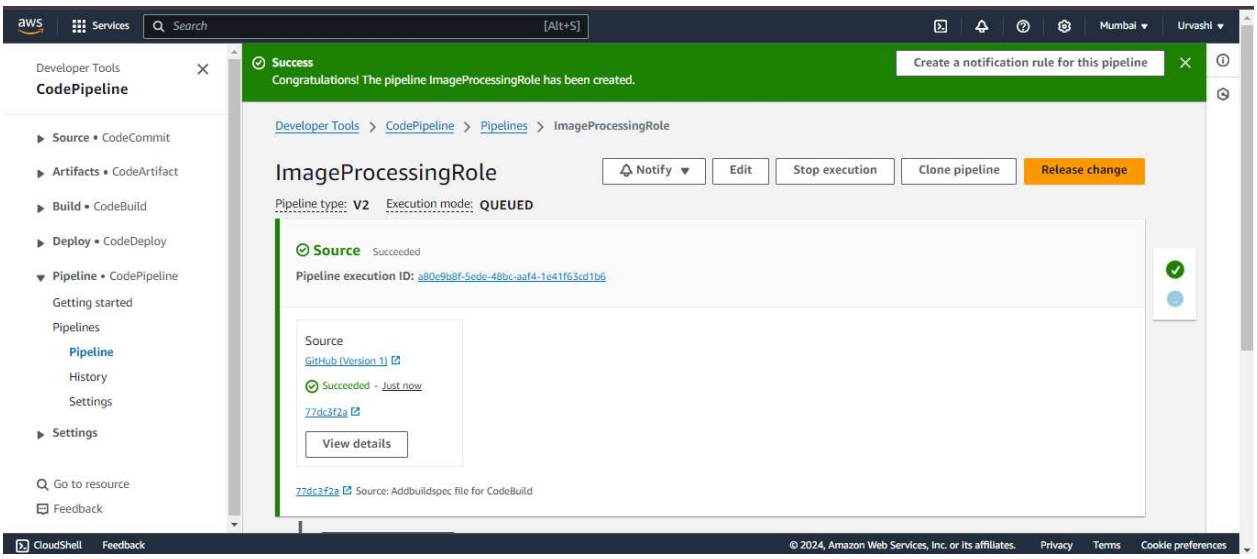
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



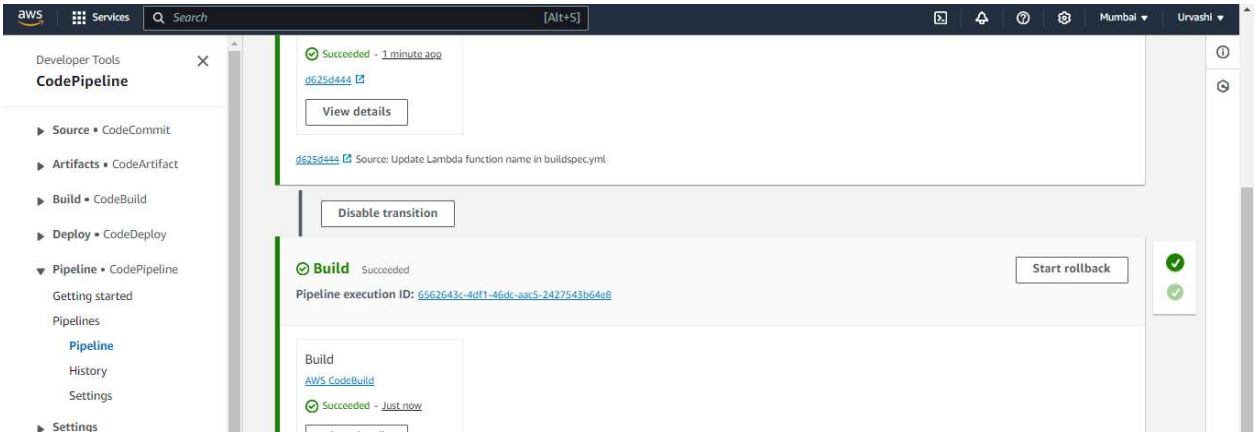
Click Create Pipeline to finish setting up





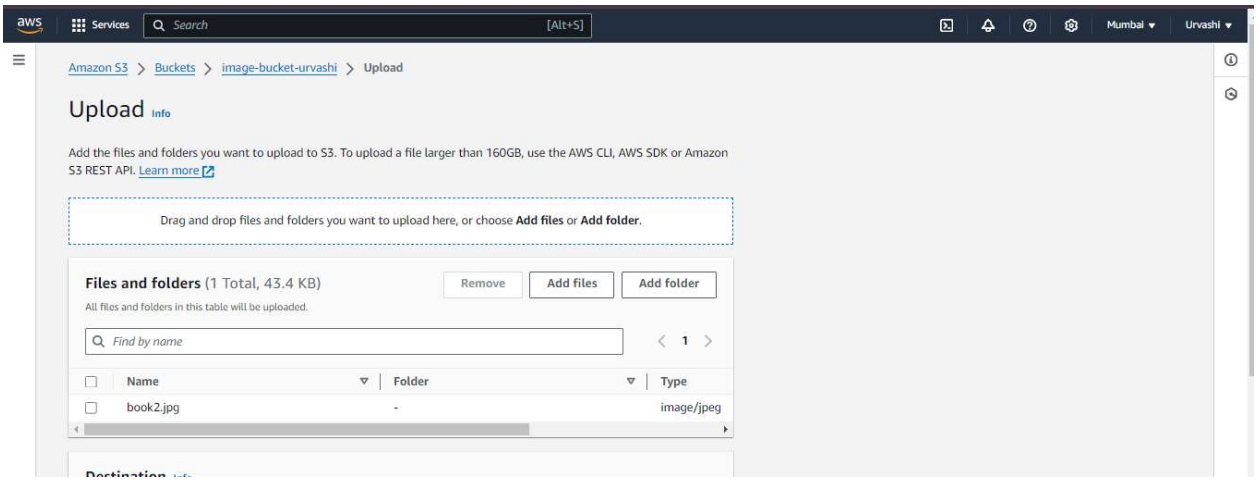
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



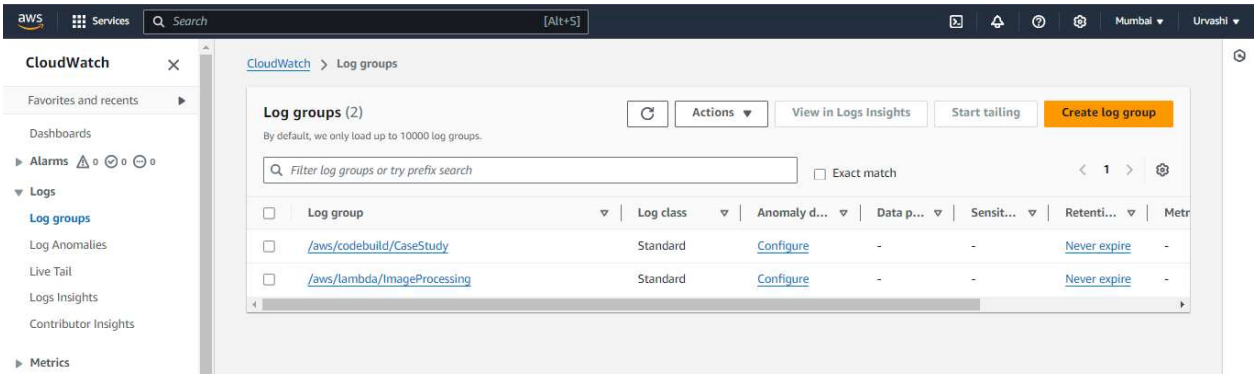
Step 6: Test the Serverless Workflow

1. **Upload an Image to S3**:
    a. Go to the S3 bucket (`image-processing-bucket`).
    b. Click "Upload" and upload any image to test the workflow.



Urvashi Changlani

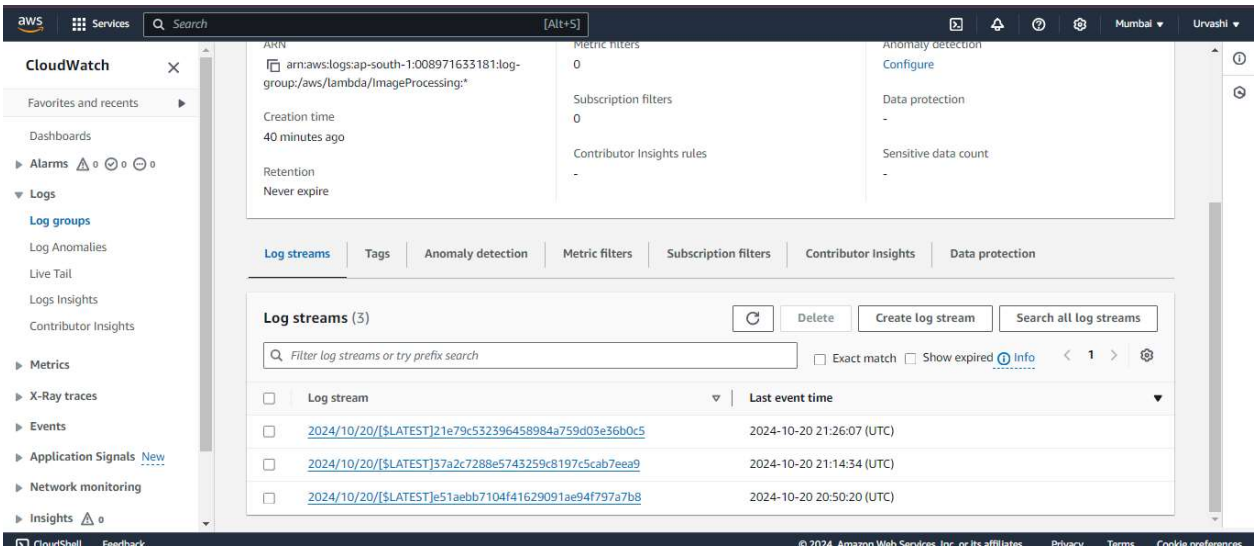
2022.urvashi.changlani@ves.ac.in



2. Check Logs in CloudWatch:

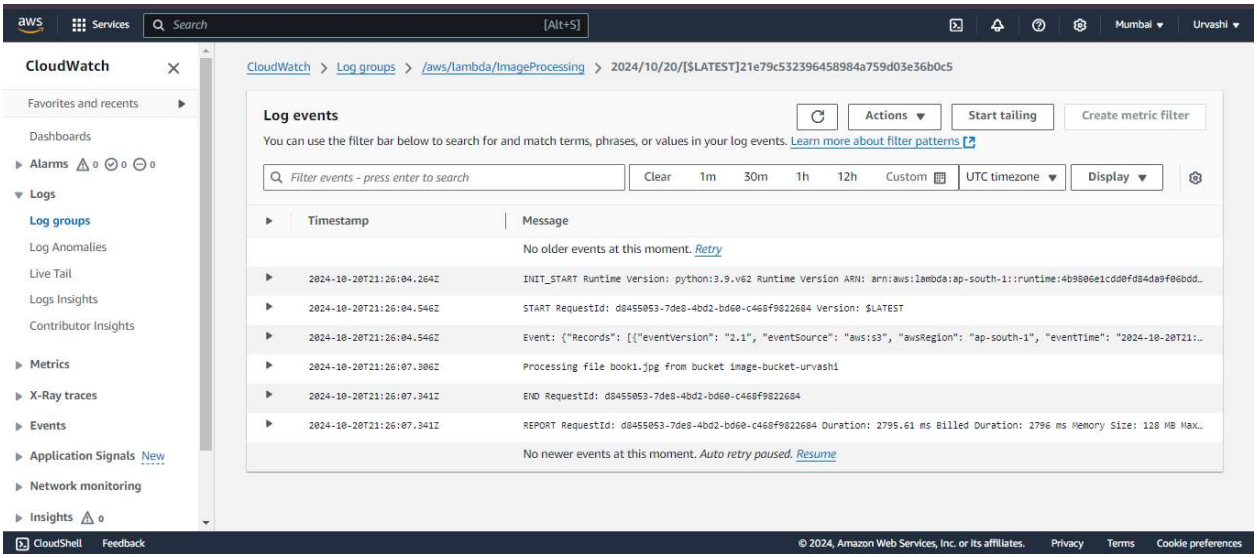
a. Go to the CloudWatch service and navigate to **Logs > Log Groups**.

b. You should see a new log group for `ImageProcessingLambda`. Review the logs to verify that the Lambda function was triggered and the image was processed successfully.



Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



Step 7: Verify CodePipeline Automation

1. **Modify the Lambda Function Code**:
    a. Make a small change to the `lambda_function.py` file, such as updating the print statement for verification:

print(f"Lambda function updated! Now processing {key} from {bucket}."

Urvashi Changlani

2022.urvashi.changlani@ves.ac.in

lambda_function.py 1, M × ! buildspec.yml

lambda_function.py > lambda_handler

```python

def lambda_handler(event, context):
    # Log the event in CloudWatch
    print("Event: ", json.dumps(event))

    # Extract S3 bucket and object details
    s3 = boto3.client('s3')
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Process the image (log details in this case)
    response = s3.get_object(Bucket=bucket, Key=key)
    print(f"Processing file {key} from bucket {bucket}")

    print("Lambda function updated!Now Processing{key} from {bucket}.")

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

2. Push the Changes to Your Repository:

git add lambda_function.py

git commit -m "Update Lambda function to add verification print statement"

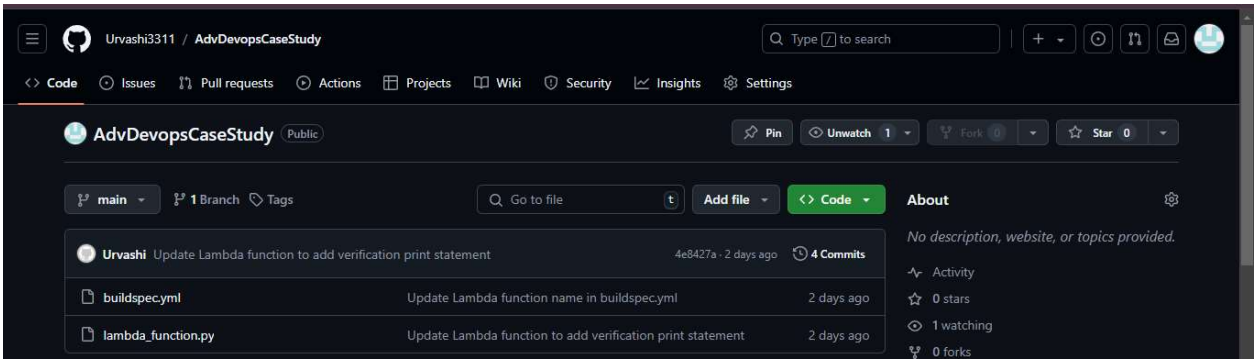
git push origin main

OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE powershell

```
PS C:\Users\chang\AdvDevopsCaseStudy> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 426 bytes | 106.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Urvashi3311/AdvDevopsCaseStudy.git
   d625d44..4e8427a  main -> main
PS C:\Users\chang\AdvDevopsCaseStudy>
```

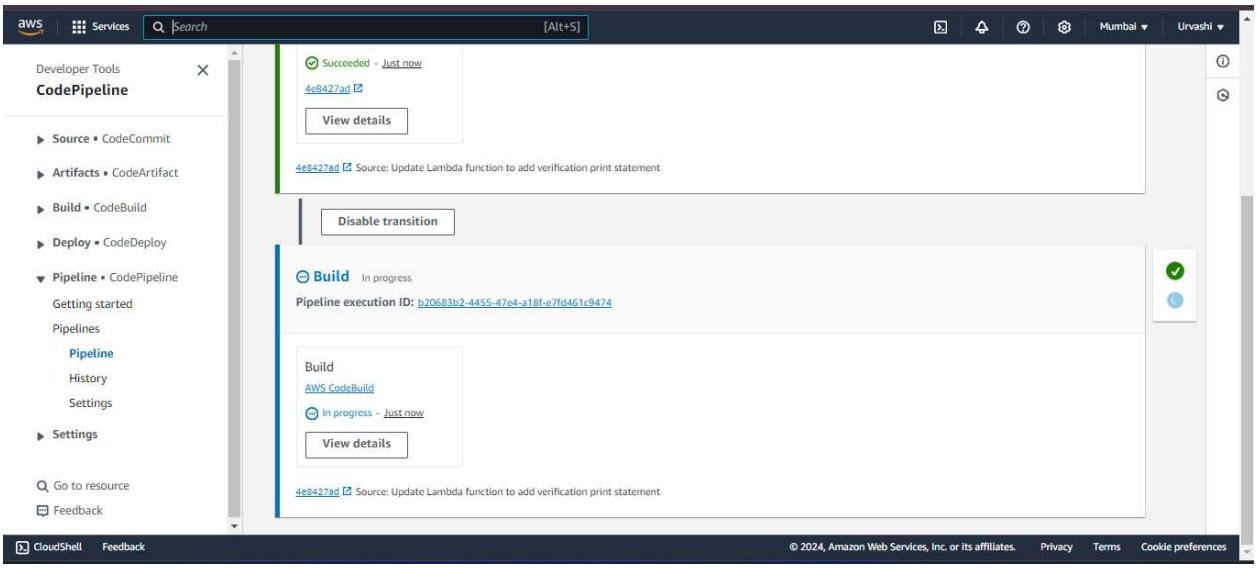
Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



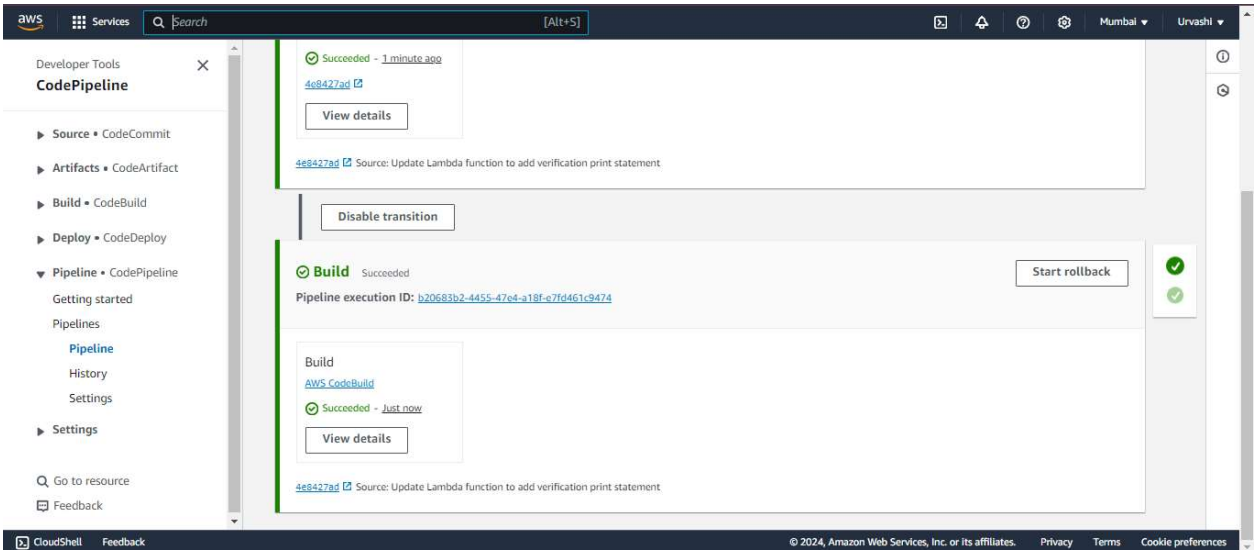
3. Verify CodePipeline Deployment:

a. CodePipeline will automatically detect changes to the repository and redeploy the Lambda function. Check CloudWatch logs after uploading another image to verify the changes.



Urvashi Changlani

2022.urvashi.changlani@ves.ac.in



Conclusion: This workflow sets up a fully automated, serverless image processing system using AWS services. It triggers an AWS Lambda function whenever a new image is uploaded to an S3 bucket, and the deployment process is automated through AWS CodePipeline. This ensures scalability, efficient event-driven processing, and continuous deployment for the Lambda function.