Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

Case Study

Serverless Image Processing Workflow

Problem Statement:

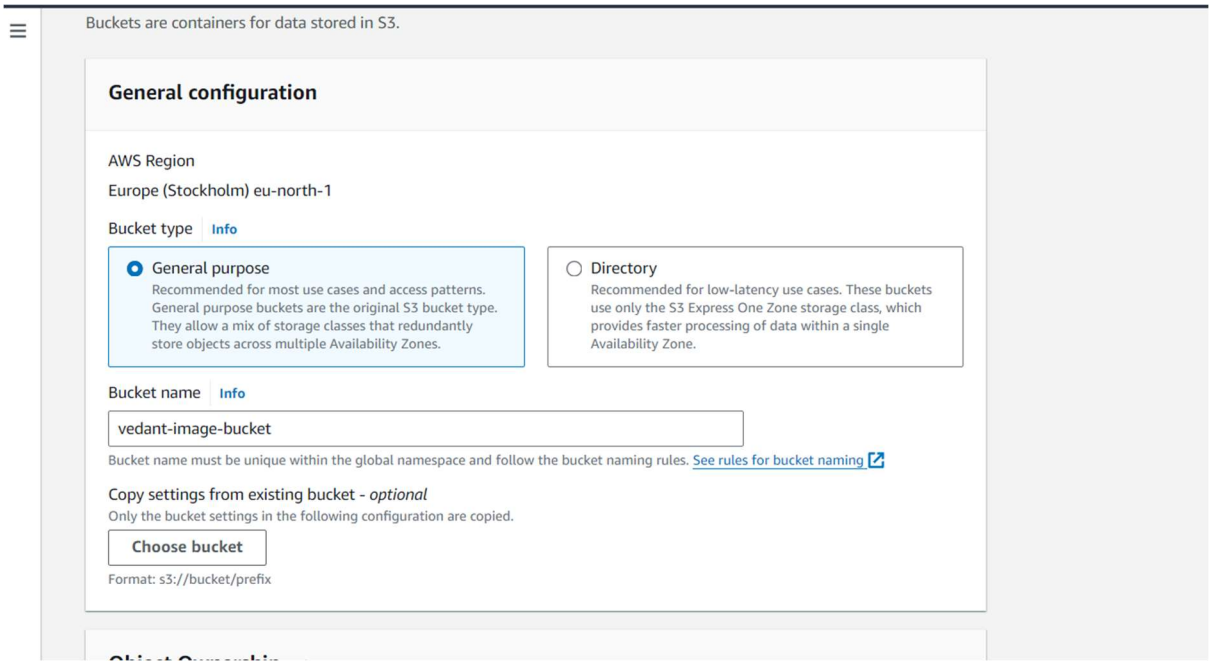
- **Concepts Used**: AWS Lambda, S3, and CodePipeline.
- **Problem Statement**: "Create a serverless workflow that triggers an AWS Lambda function when a new image is uploaded to an S3 bucket. Use CodePipeline to automate the deployment of the Lambda function."
- **Tasks**:
    - Create a Lambda function in Python that logs and processes an image when uploaded to a specific S3 bucket.
    - Set up AWS CodePipeline to automatically deploy updates to the Lambda function.
    - Upload a sample image to S3 and verify that the Lambda function is triggered and logs the event.

SOLUTION

Step 1: Set Up an S3 Bucket

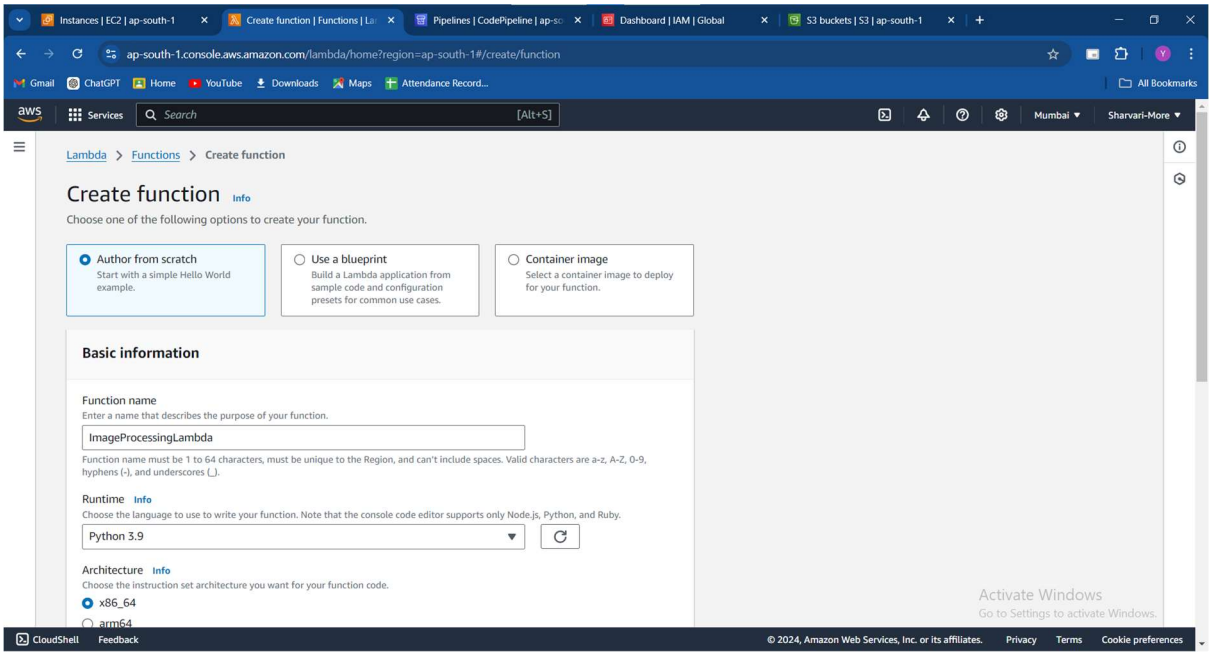
1. **Log in to AWS Console** and go to the **S3** service.
2. Click **Create Bucket**, give it a unique name (e.g., image-processing-bucket), and choose a region.
3. Enable **versioning** if needed and leave other options as default. Click **Create Bucket**.

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



Step 2: Create a Lambda Function to Process Images

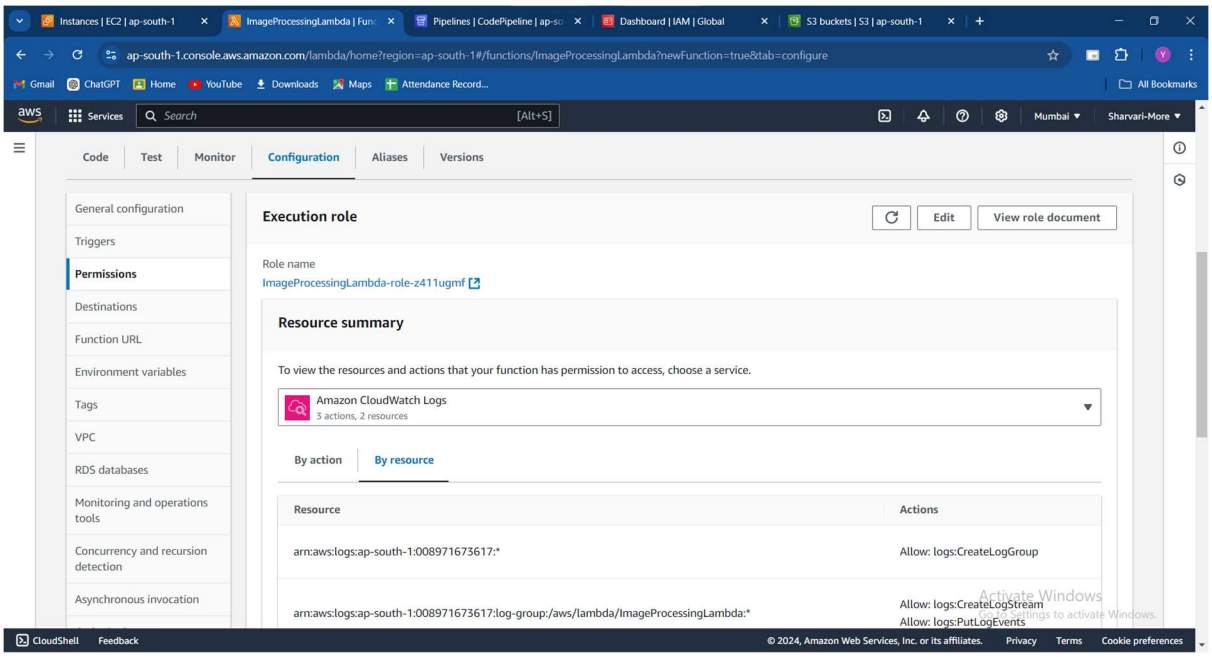
1. Go to the **Lambda** service in AWS.
2. Click **Create Function** and choose **Author from Scratch**.
    - **Name**: ImageProcessingLambda
    - **Runtime**: Python 3.x (e.g., Python 3.9)



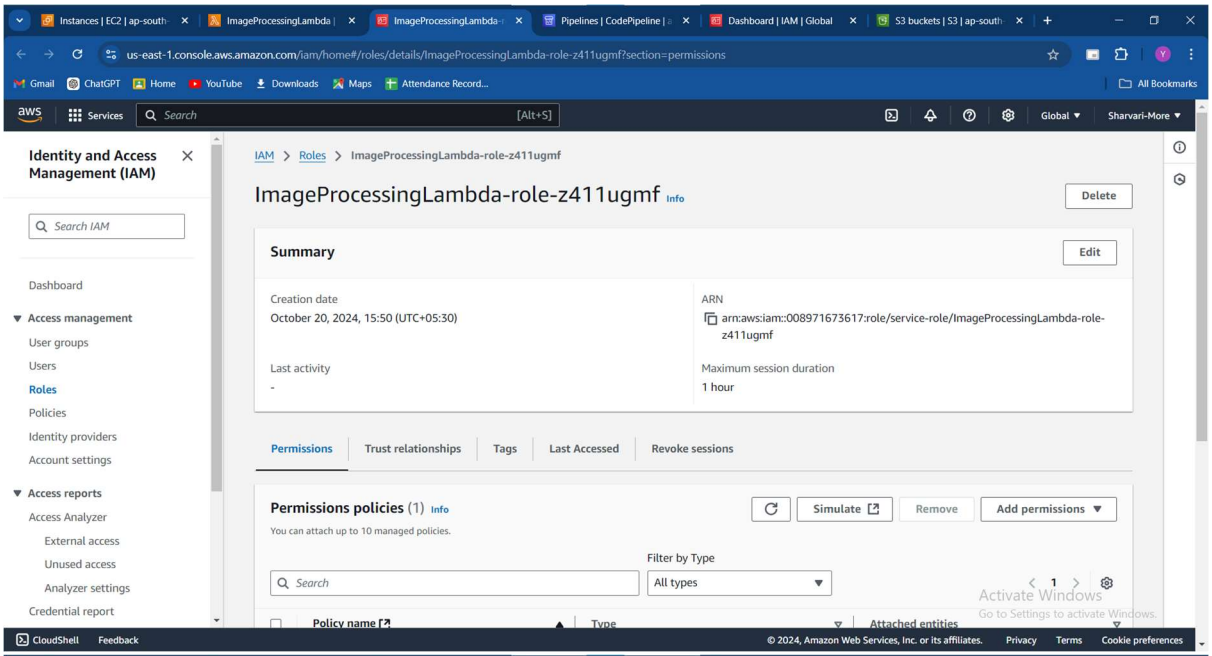
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

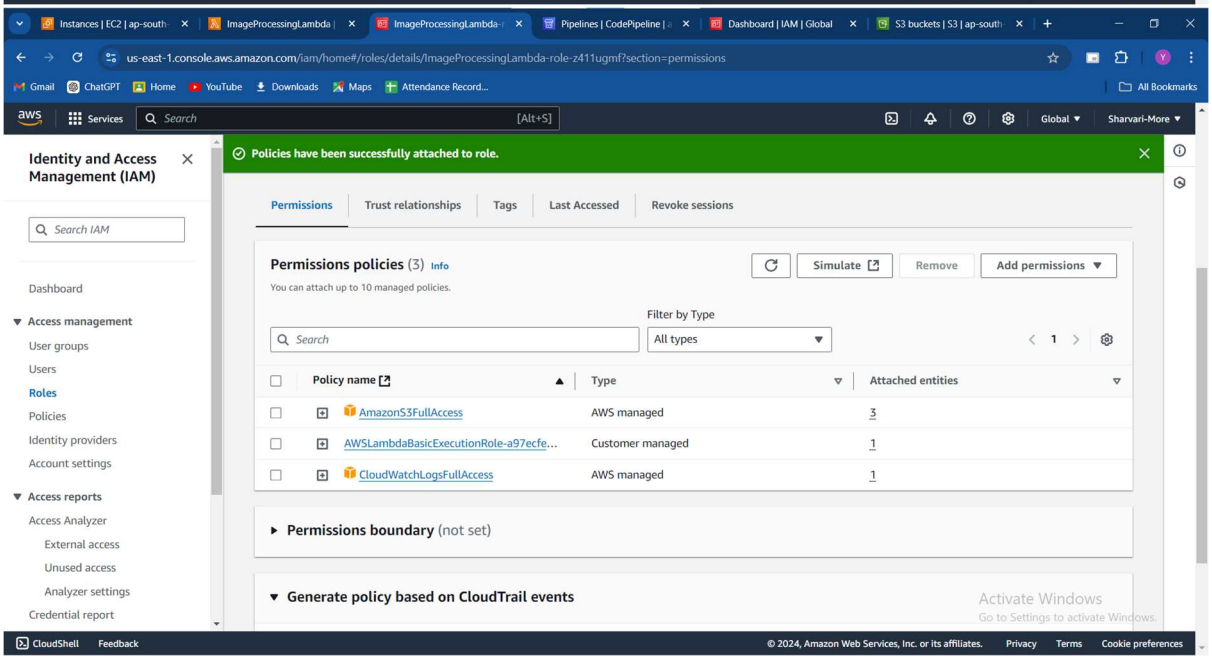
3. **IAM Role for Lambda**:
    - Create a new role with basic Lambda permissions:
        - Choose **Create a new role with basic Lambda permissions**.
        - It automatically assigns the policy **AWSLambdaBasicExecutionRole** to the role, which allows the function to write logs to **CloudWatch**.

4. After the function is created, add the following permissions to access the S3 bucket:
    - Click on **Configuration** > **Permissions** > **Execution Role**.
    - Click on the role and attach the following permissions:
        - **AmazonS3FullAccess**
        - **CloudWatchLogsFullAccess**



Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in





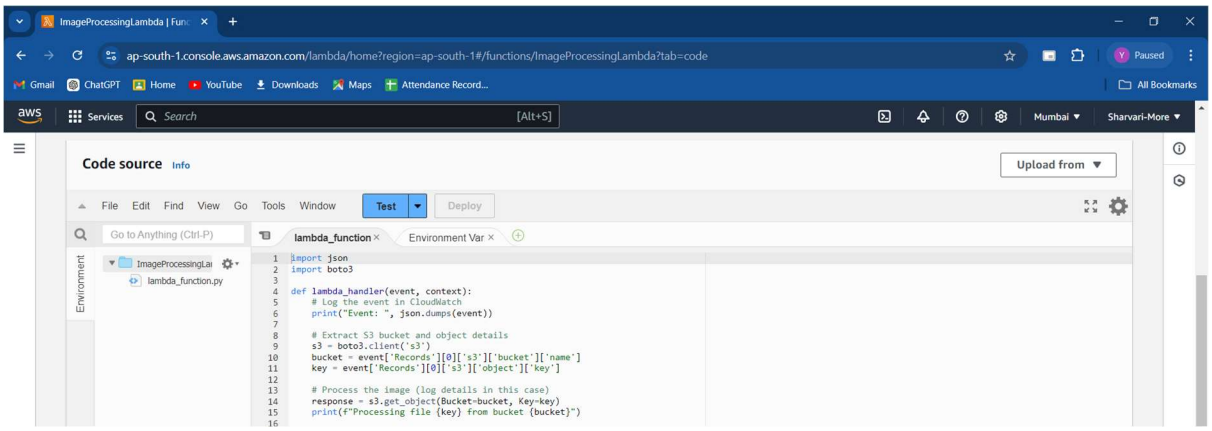
5. **Add Python Code to Process Images**:
    - Go back to **Code** section and replace the sample code with:

CODE:

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

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

```
# 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}")

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

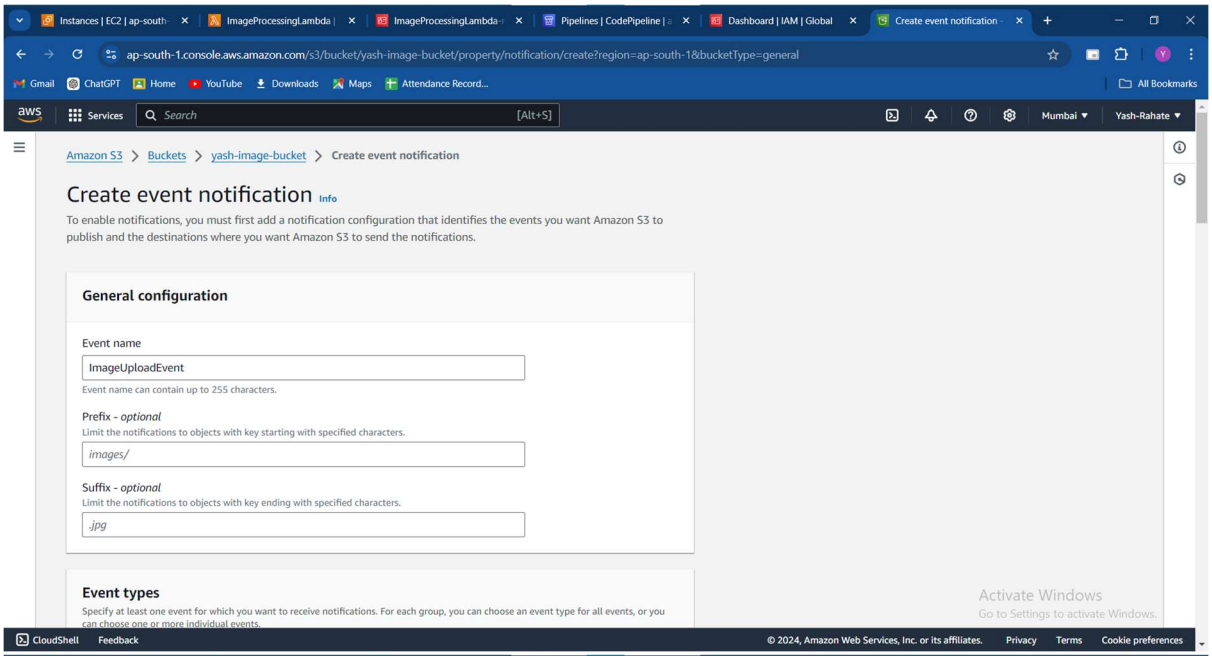


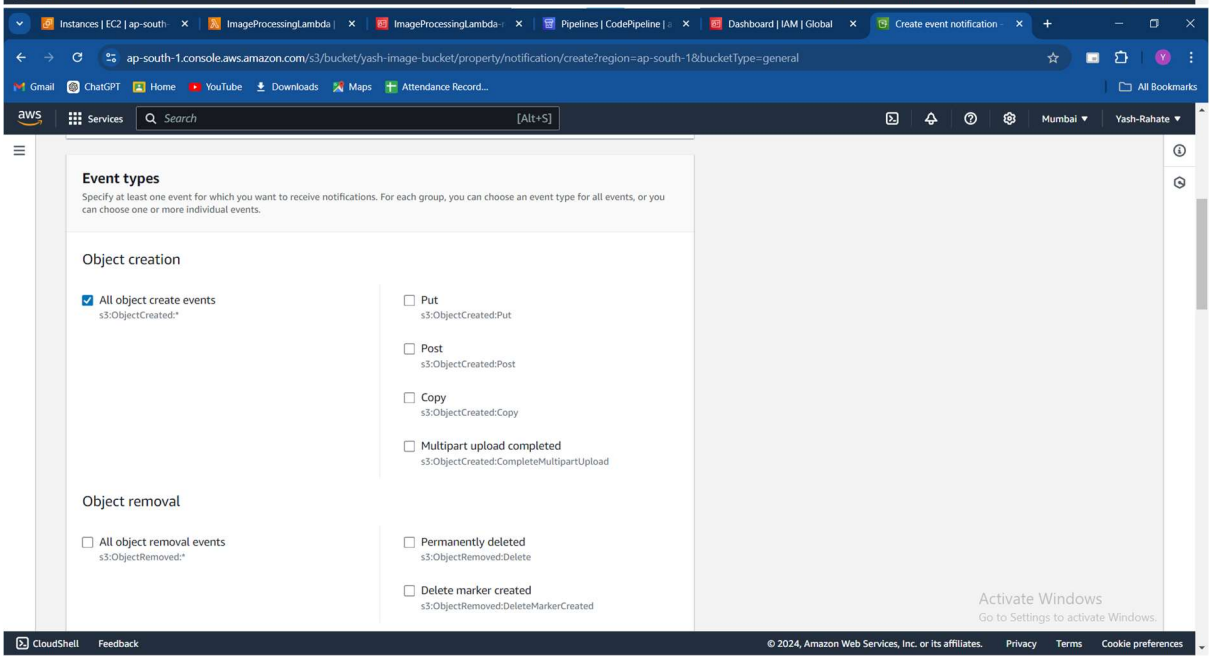
- This code logs the S3 event and retrieves basic information about the uploaded image.

Step 3: Set Up S3 Event Notification to Trigger Lambda

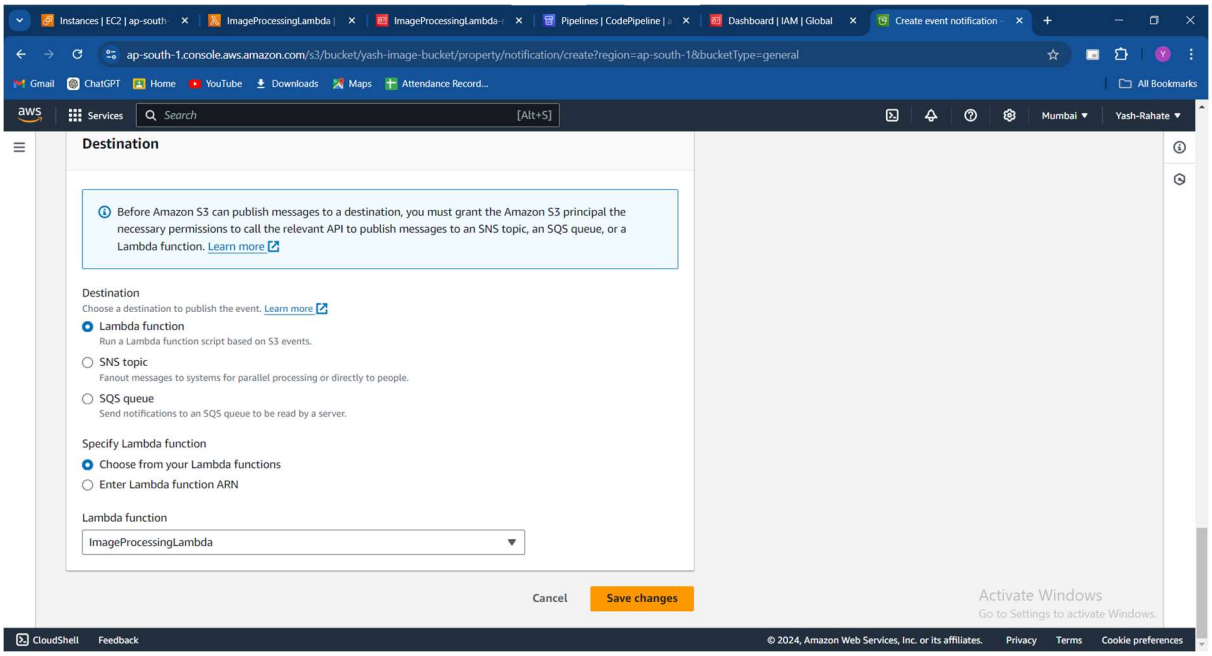
1. Go back to the **S3** service and select your bucket (image-processing-bucket).
2. In the **Properties** tab, scroll to the **Event Notifications** section and click **Create Event Notification**.
   - **Event Name**: ImageUploadEvent
   - **Event Type**: Select **All object create events** (i.e., triggers when any file is uploaded).
   - **Destination**: Choose **Lambda function** and select ImageProcessingLambda.

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in





Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



3. Click **Save Changes**.

Step 4 :Step-by-Step Guide Using CodeBuild:

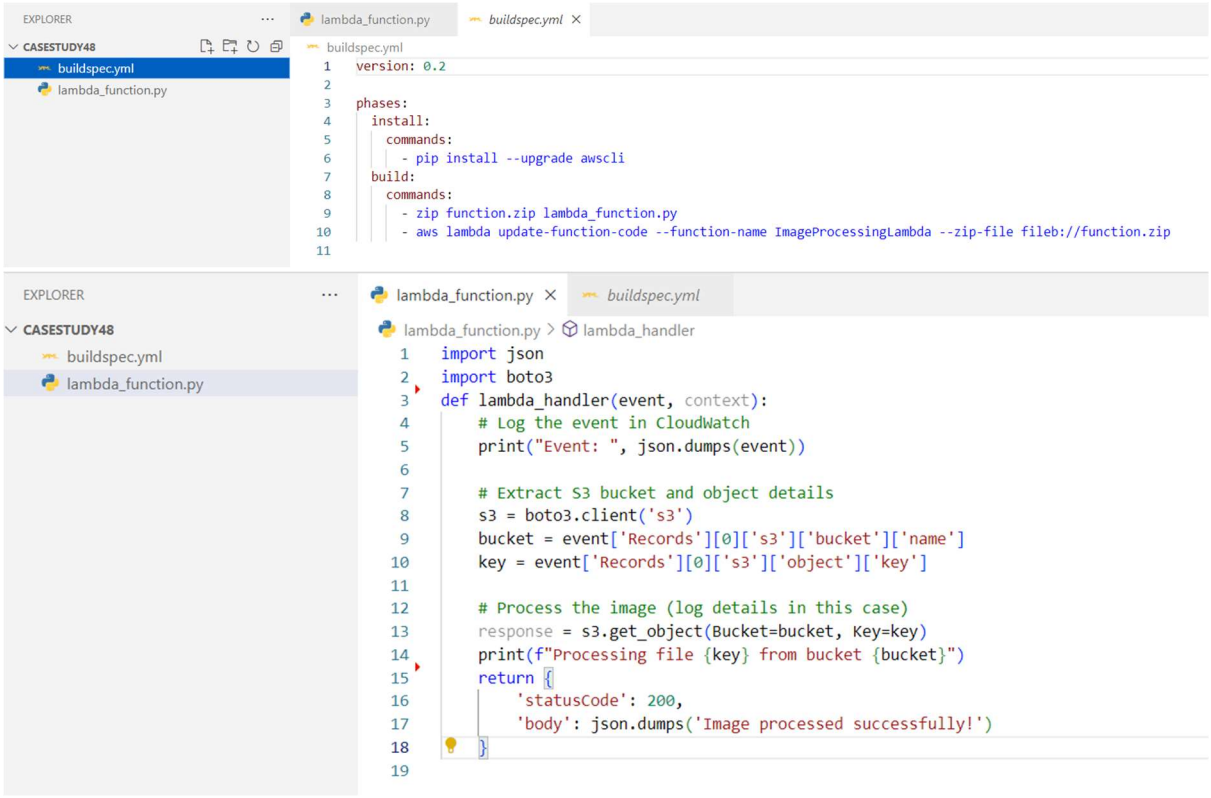
1. **Create a Buildspec File**: In your GitHub repo (where your lambda_function.py is), add a buildspec.yml file. This file will tell CodeBuild how to package and deploy your Lambda function.
   Example buildspec.yml:

   CODE:

```
version: 0.2

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

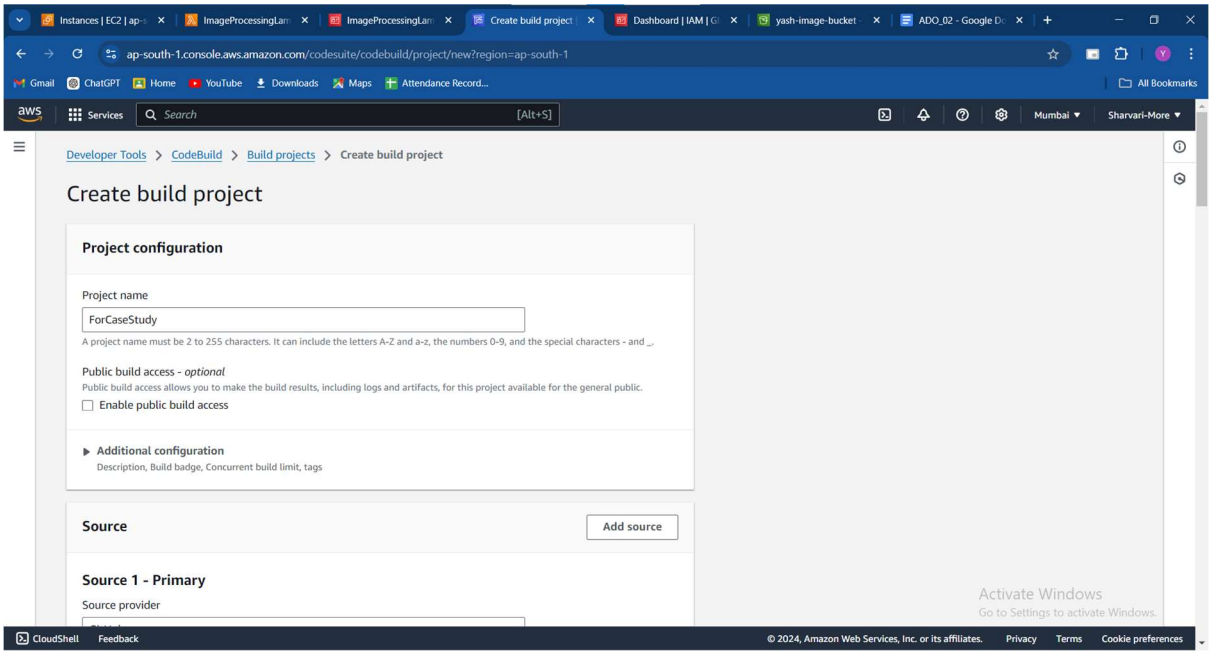
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

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



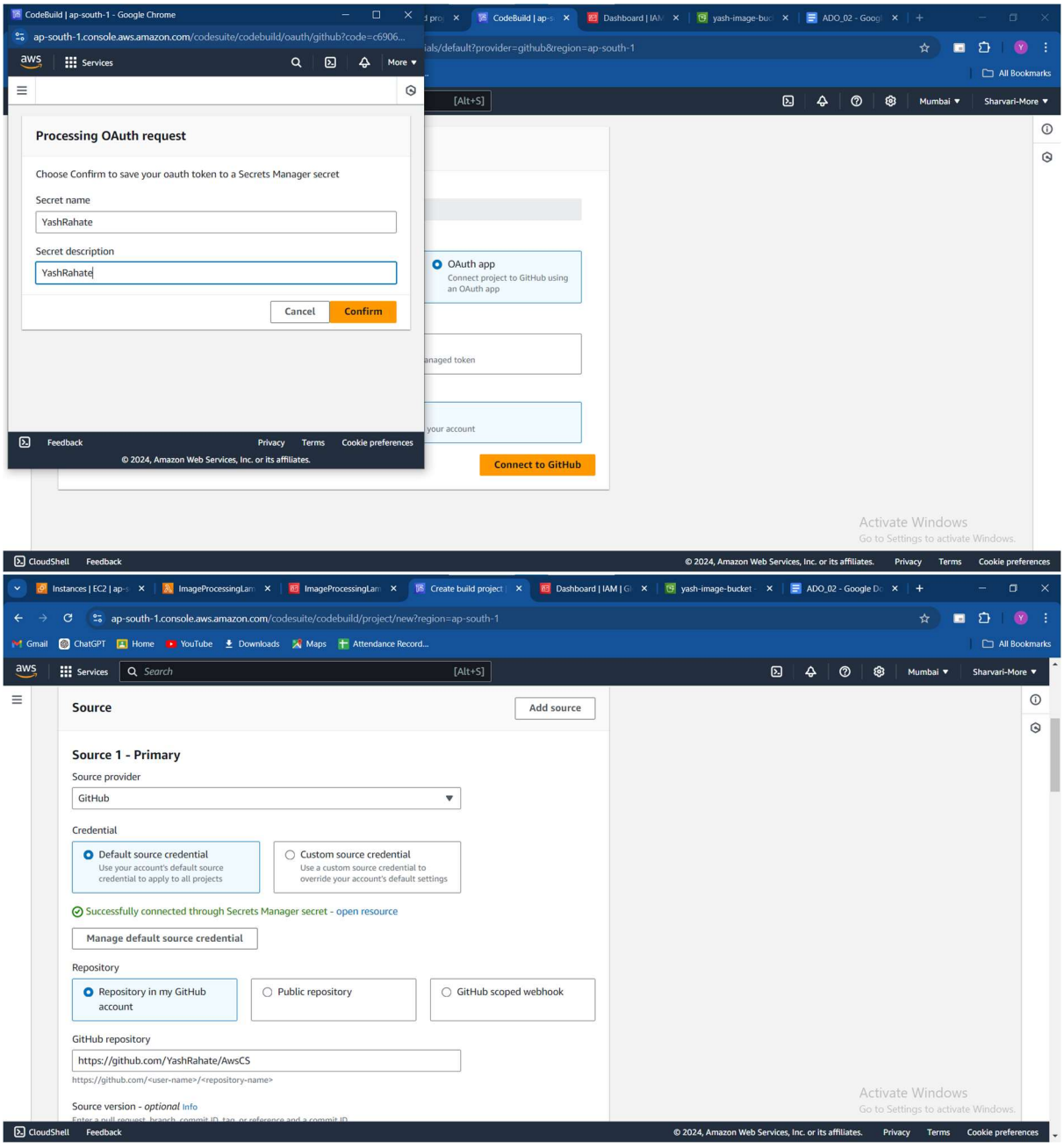
2. **Create a CodeBuild Project**:

- Go to **AWS CodeBuild** and create a new build project.



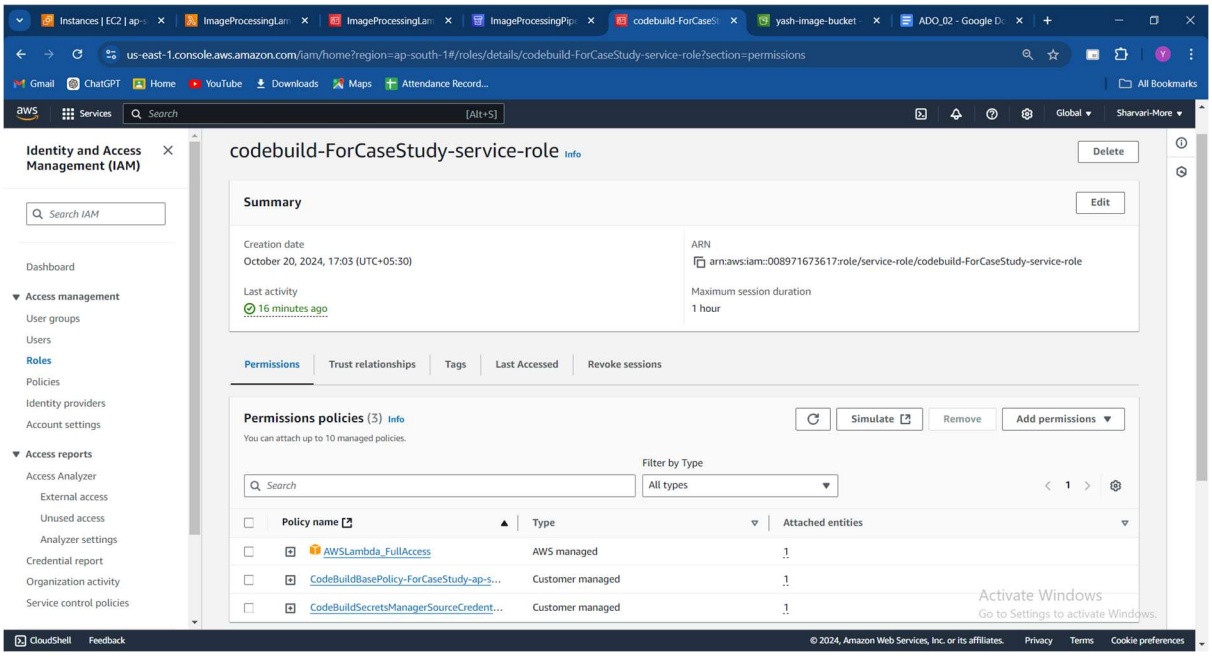
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

- For the **Source**, select the same GitHub repo you are using.

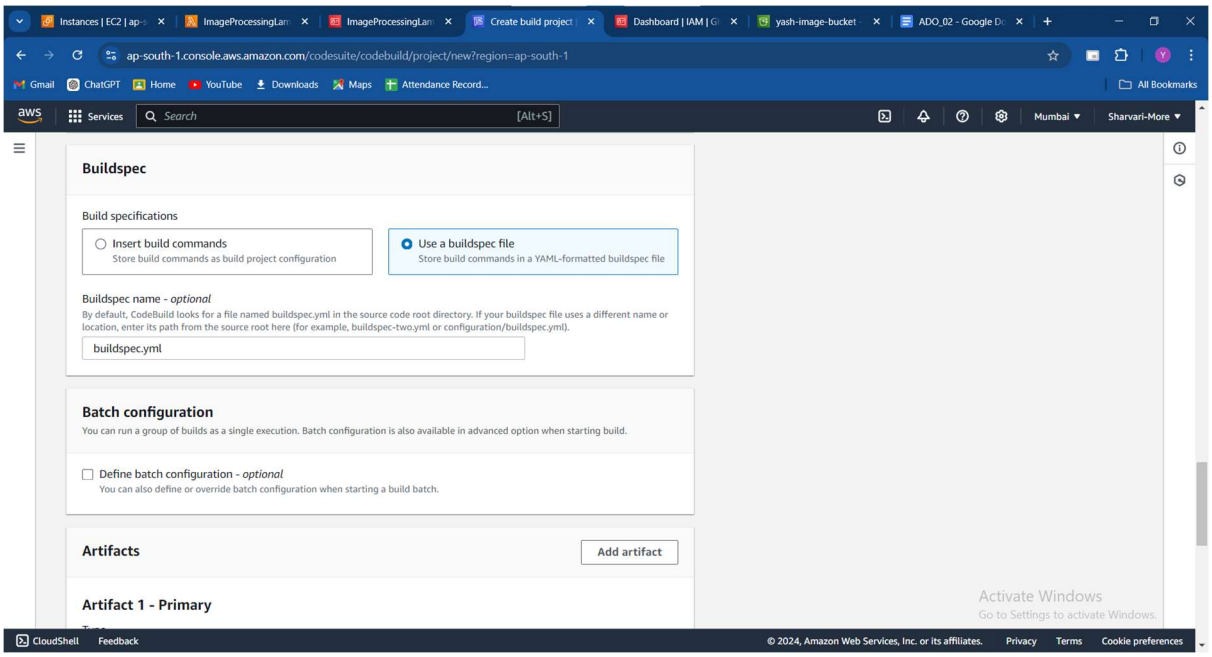


- For the **Environment**, select a managed image (e.g., Ubuntu with standard runtimes).
- Ensure that the environment has the correct permissions to update the Lambda function (using a role with AWSLambdaFullAccess or similar).

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



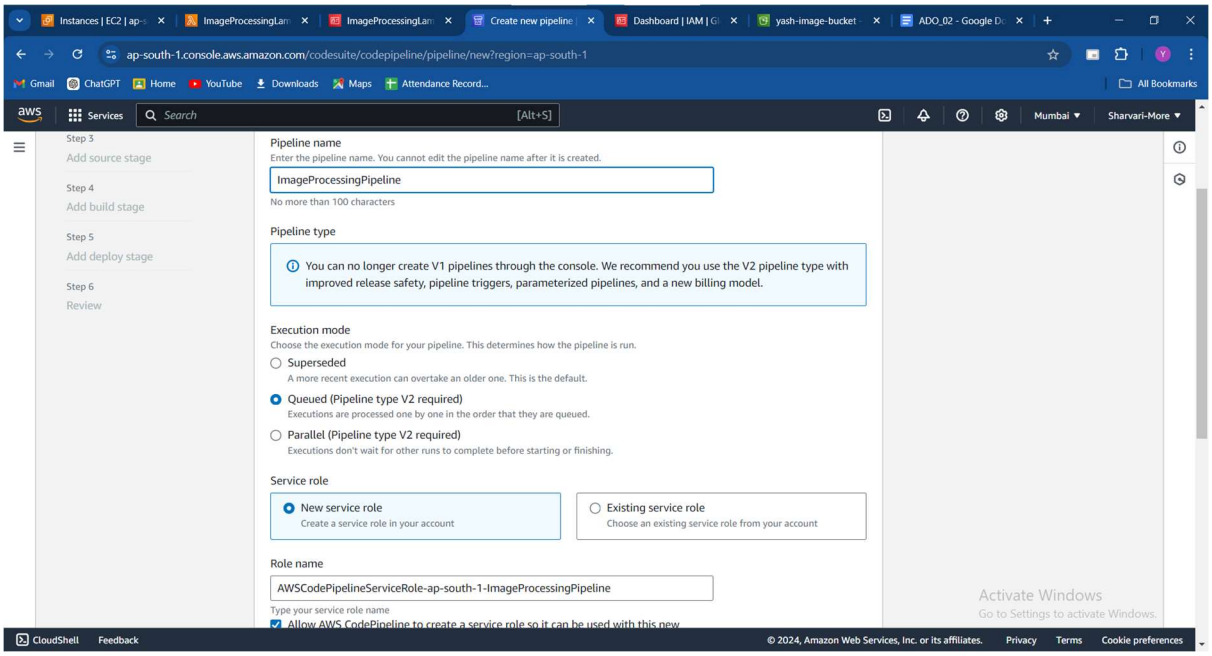
- Specify the **buildspec.yml** file from your GitHub repository.



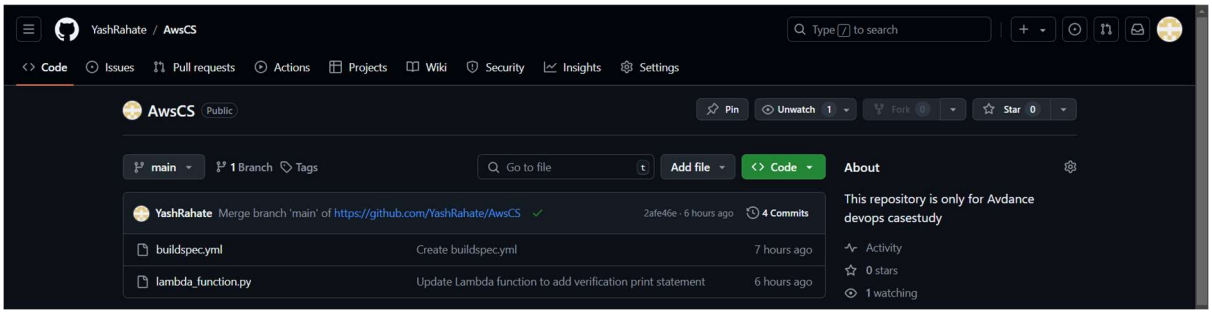
Step 5: Set Up AWS CodePipeline to Automate Lambda Deployment

1. Go to the **CodePipeline** service and click **Create Pipeline**.
2. **Pipeline Settings**:
    - **Pipeline Name**: ImageProcessingPipeline
    - **Service Role**: Allow CodePipeline to create a new role.

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

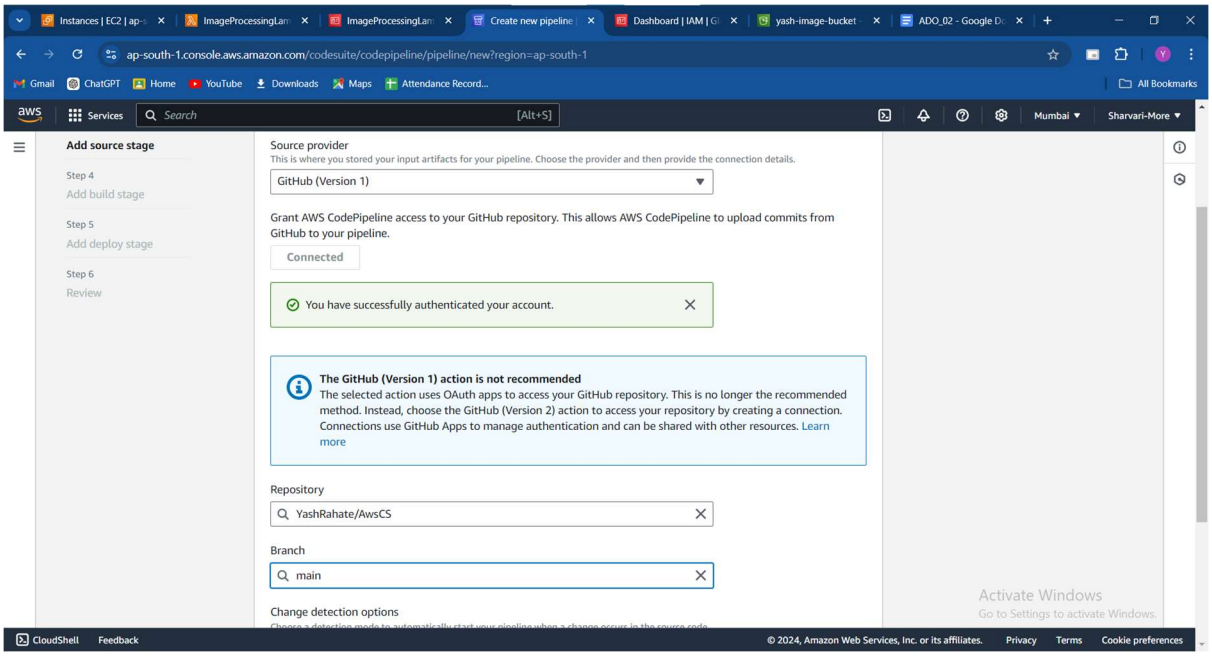


3. **Source Stage** (Code Repository):
    - For **Source Provider**, choose **GitHub** or **AWS CodeCommit** based on your code repository.



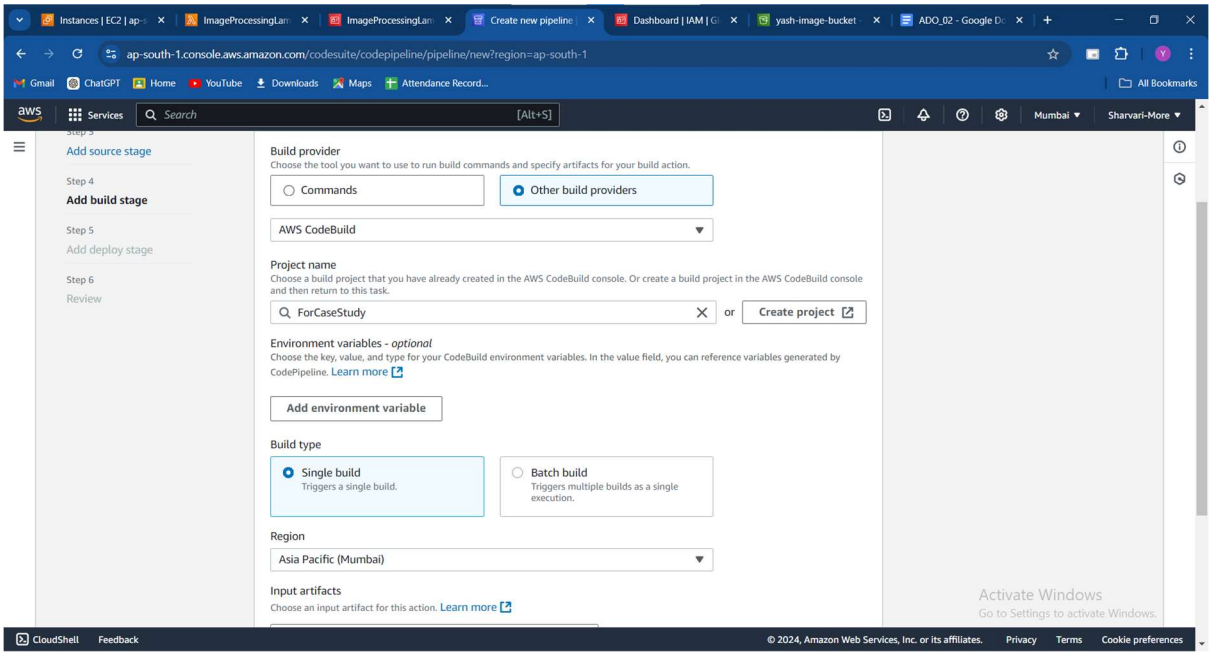
    - Connect your repository that contains the Lambda code (use the same code as in Step 2).

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



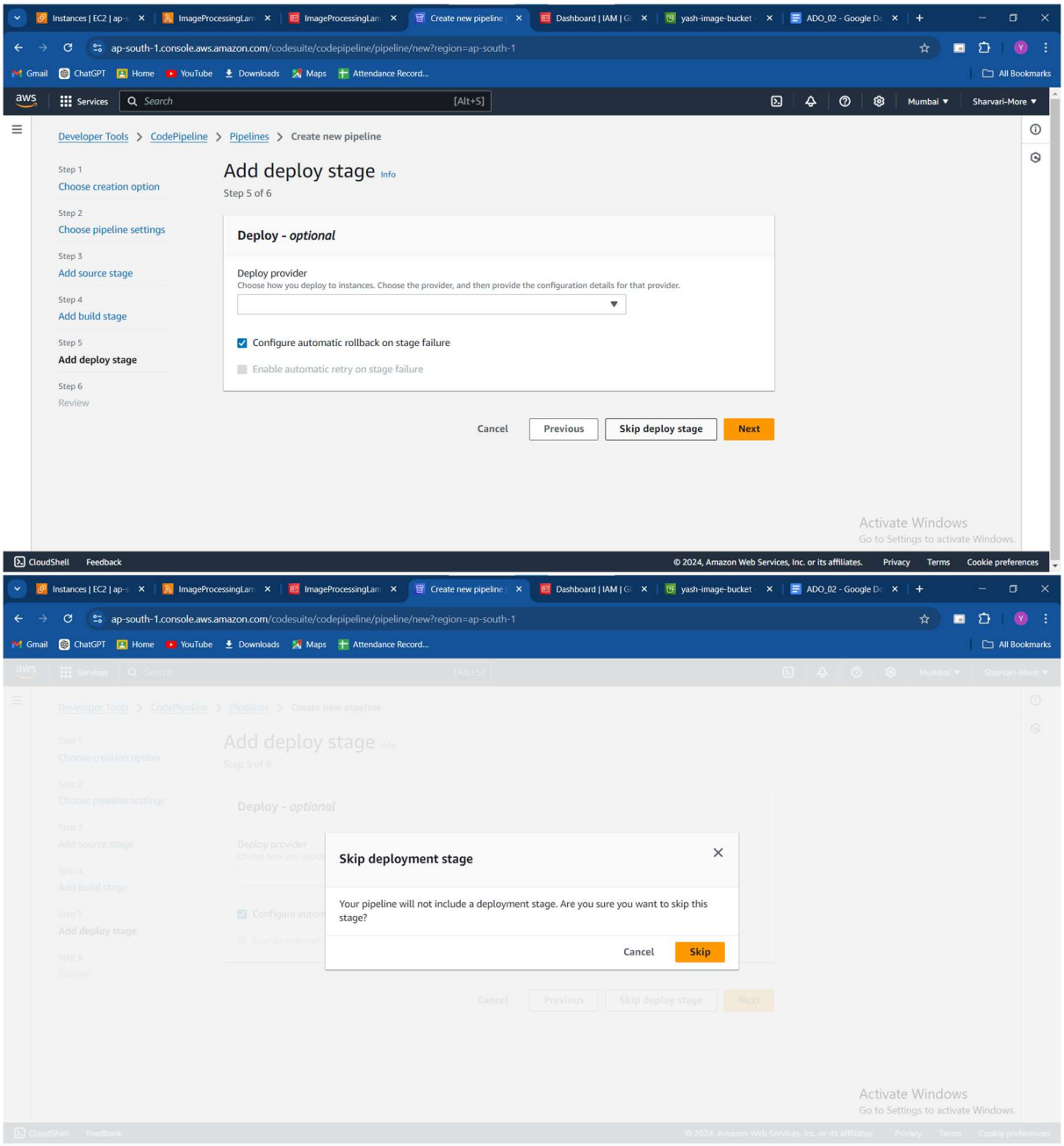
4. **Add CodeBuild to CodePipeline**:

- In your CodePipeline, add **CodeBuild** as the **Build Stage** (instead of a Deploy Stage).
- This will allow CodePipeline to trigger the CodeBuild project, which will run the buildspec.yml commands to package and deploy the Lambda function.



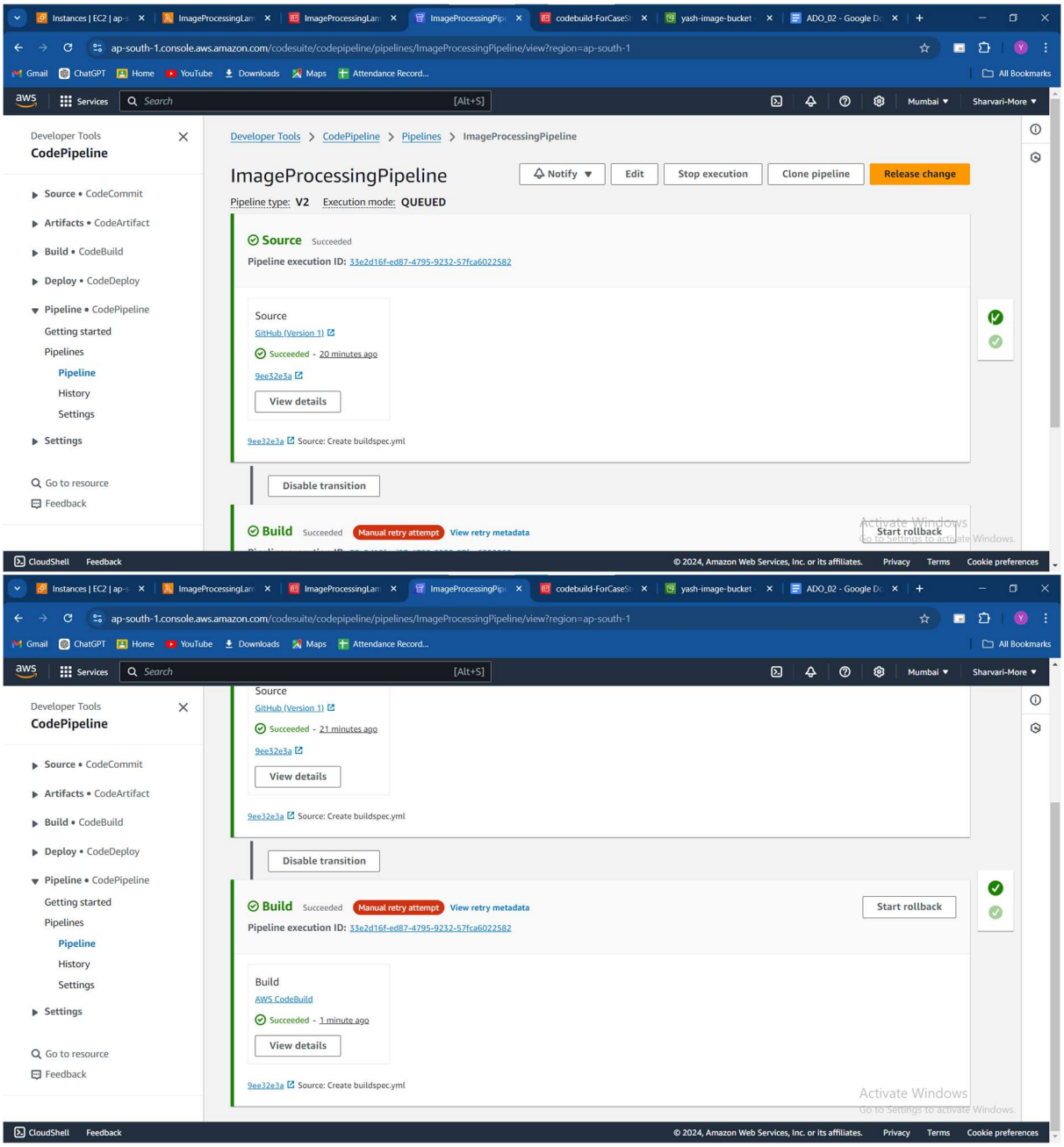
5. **Deploy Stage** (Deploy to Lambda):
    - SKIP THIS (as Choose **AWS Lambda** as the deploy provider Does not exist.)

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



6. Click **Create Pipeline** to finish setting up.

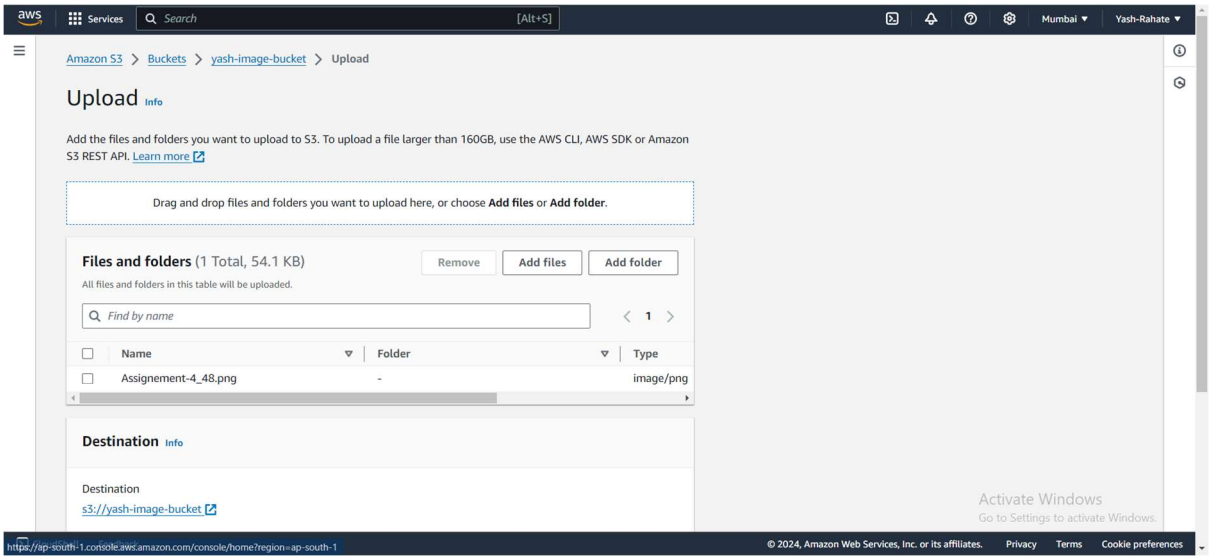
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



Step 6: Test the Serverless Workflow

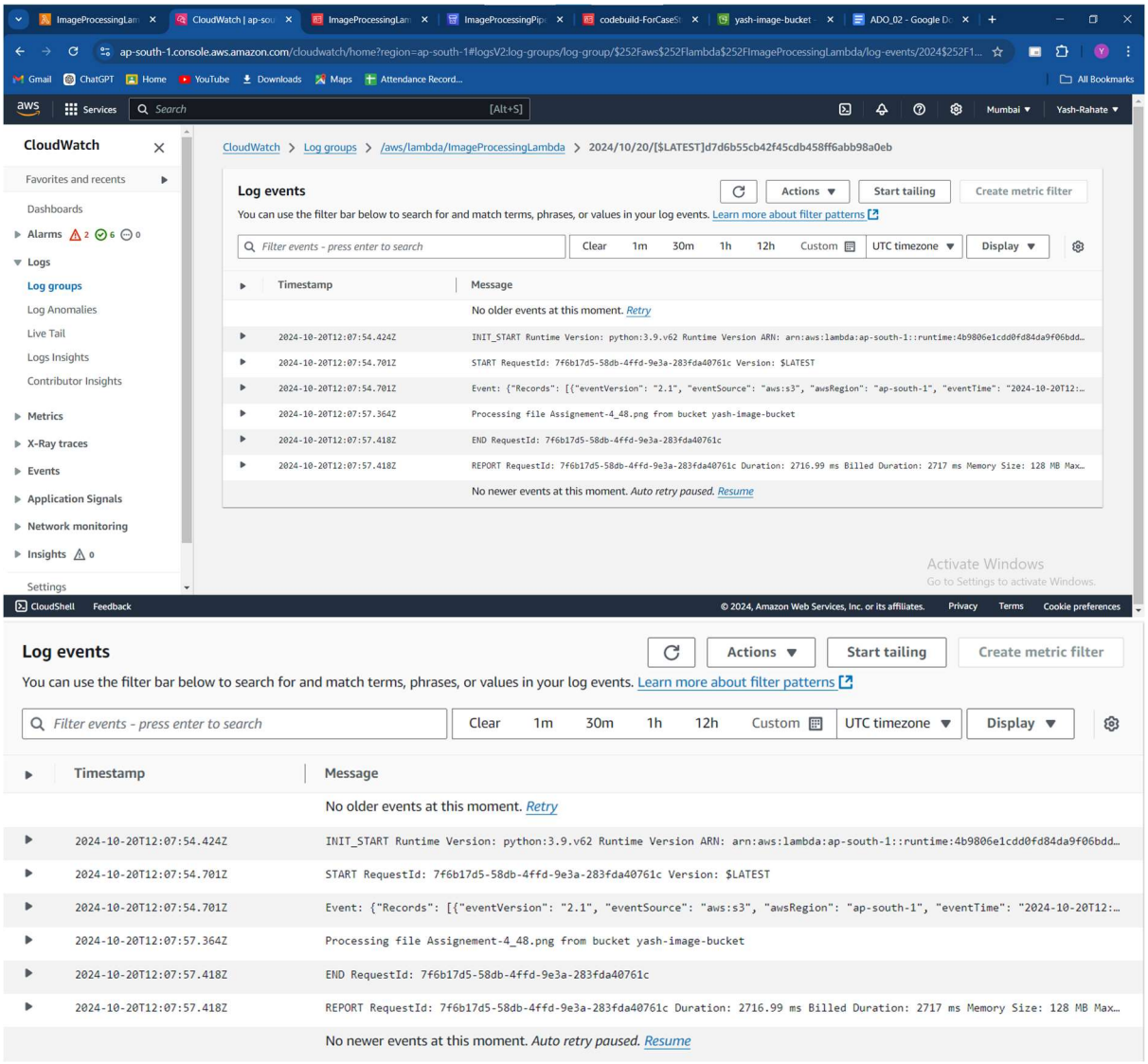
1. **Upload a sample image** to your S3 bucket:
   - Go to **S3**, select the bucket image-processing-bucket, and click **Upload**.
   - Upload any image

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



2. **Check CloudWatch Logs**:
    - Go to **CloudWatch** > **Logs** > **Log groups**.
    - You should see a new log group for ImageProcessingLambda.
    - In the logs, you'll see details about the S3 event, including the bucket name and the key (filename).

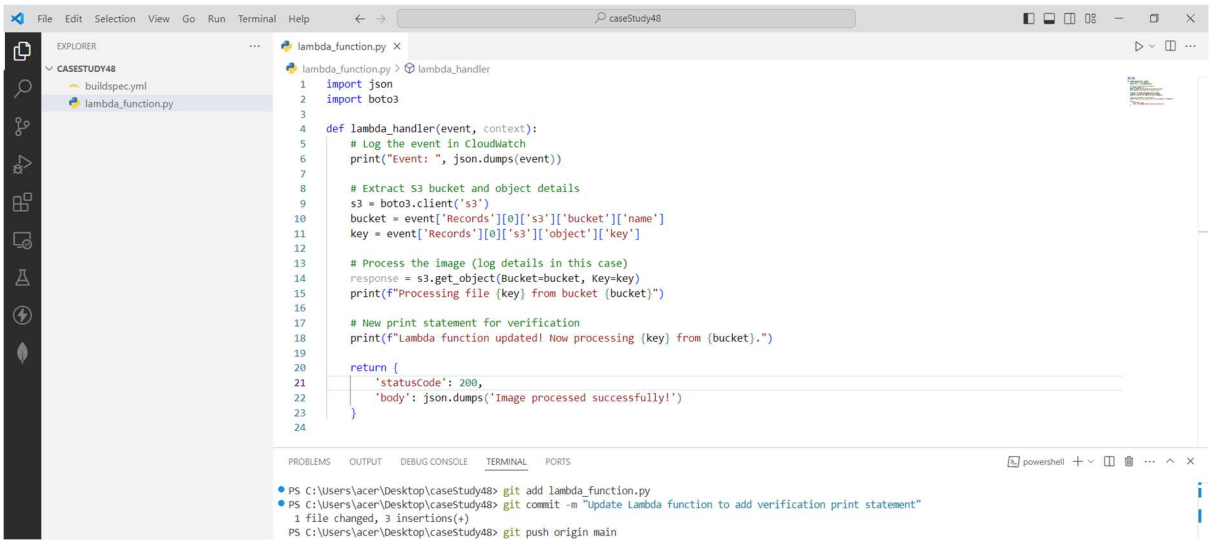
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



Step 7: Verify CodePipeline Automation

1. **Make a change** to the Lambda function code (e.g., update the print statement).

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



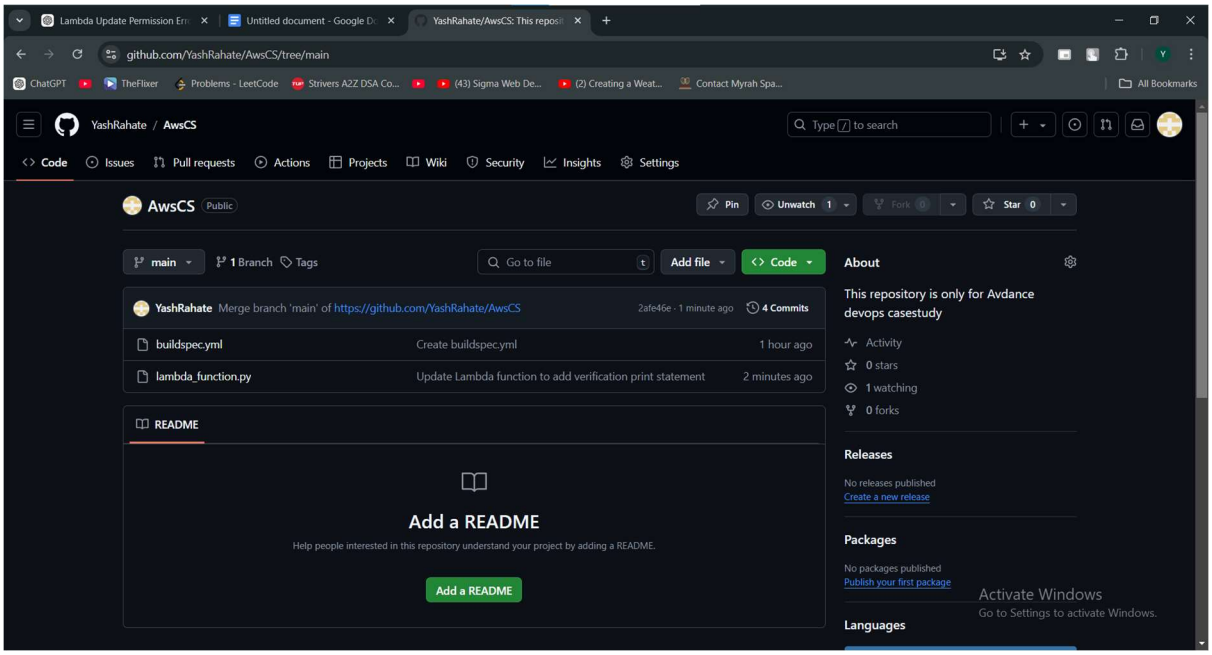
New Code:

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

    # New print statement for verification
    print(f"Lambda function updated! Now processing {key} from {bucket}.")

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

2. **Push the changes** to the GitHub or CodeCommit repository.
   git add lambda_function.py
   git commit -m "Update Lambda function to add verification print statement"
   git push origin main

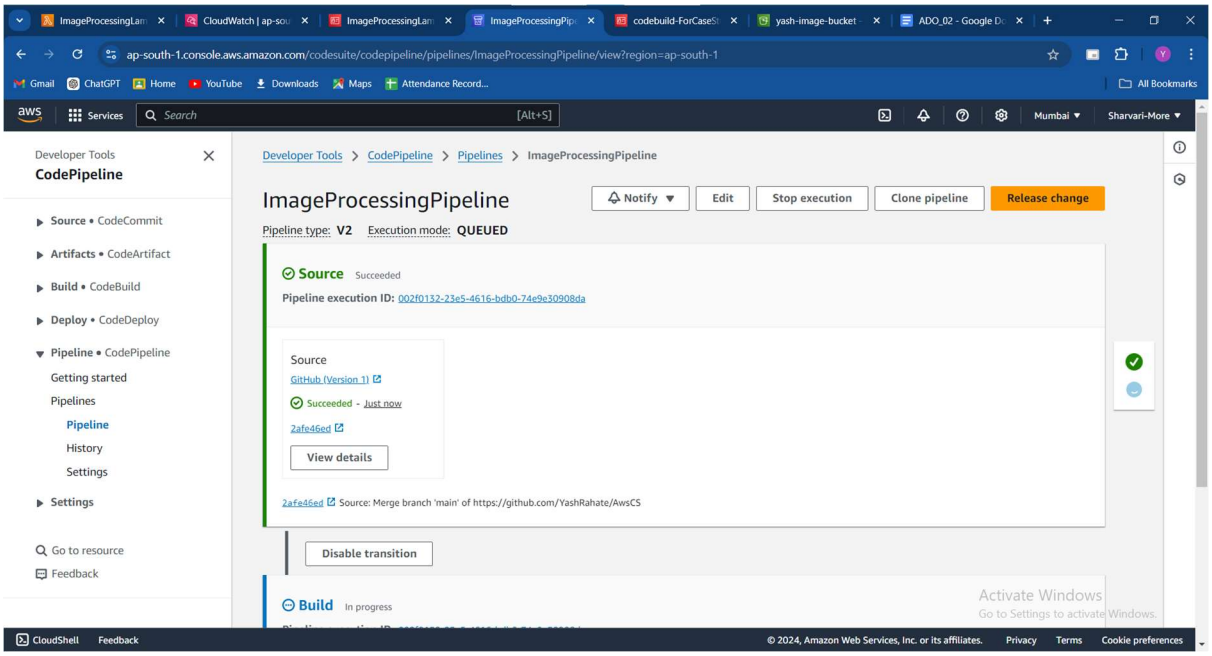
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

```
PS C:\Users\acer\Desktop\caseStudy48> git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 742 bytes | 371.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/YashRahate/AwsCS.git
```

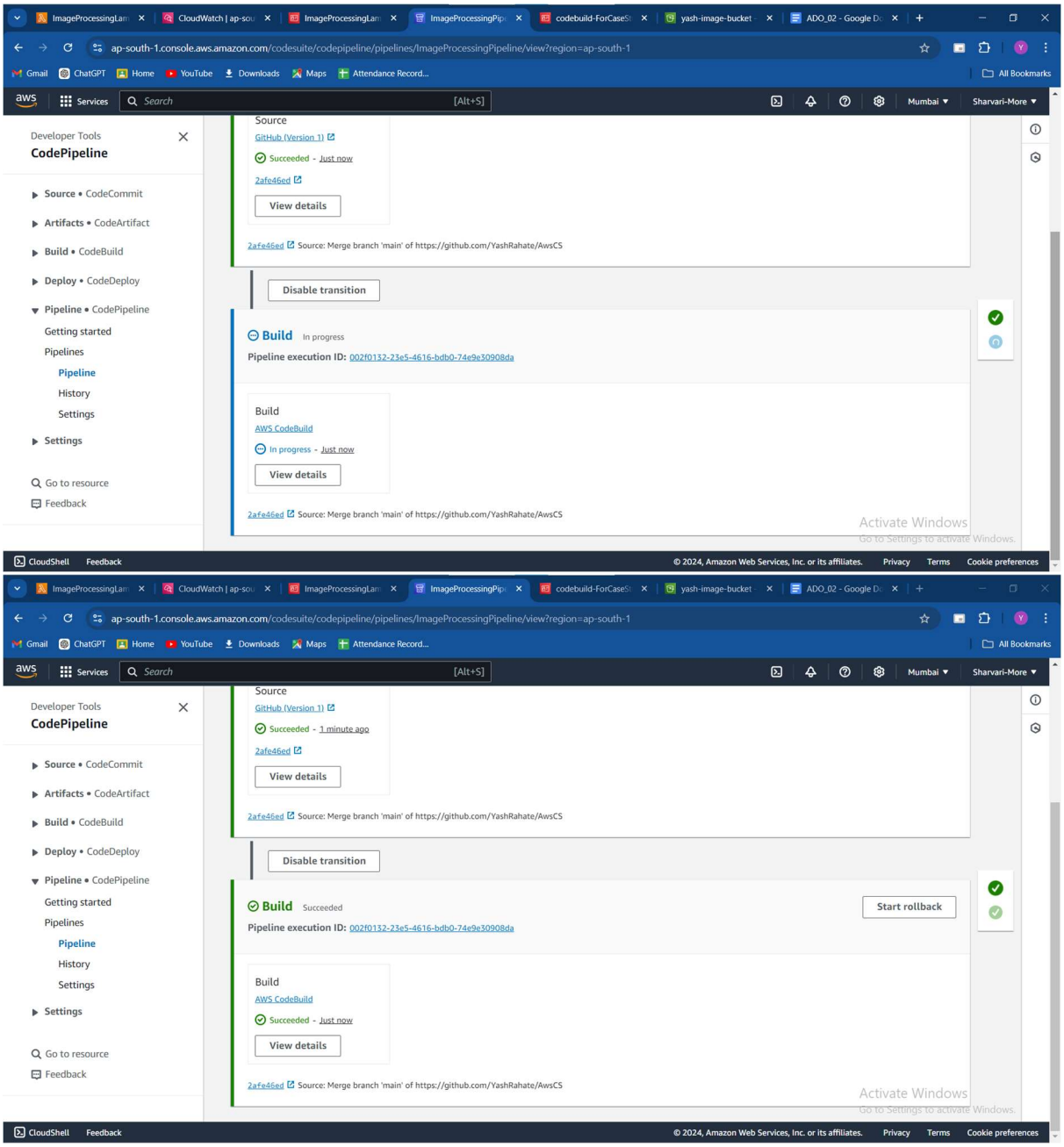


3. CodePipeline will automatically detect the changes and redeploy the updated Lambda function.

Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in

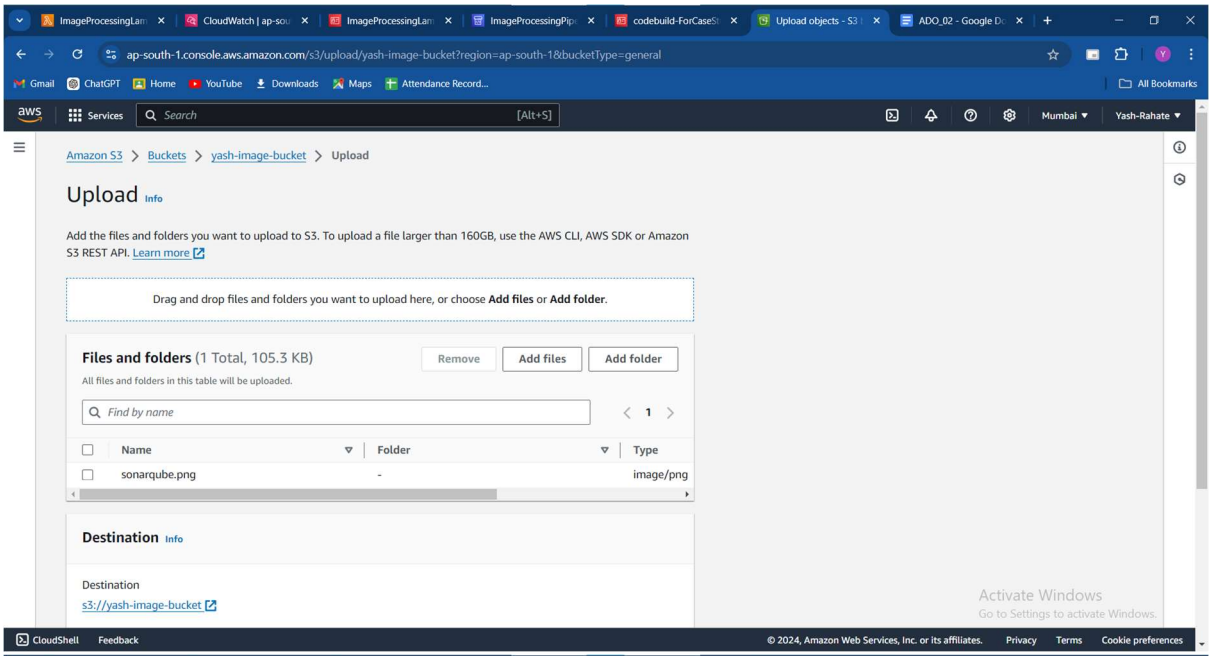


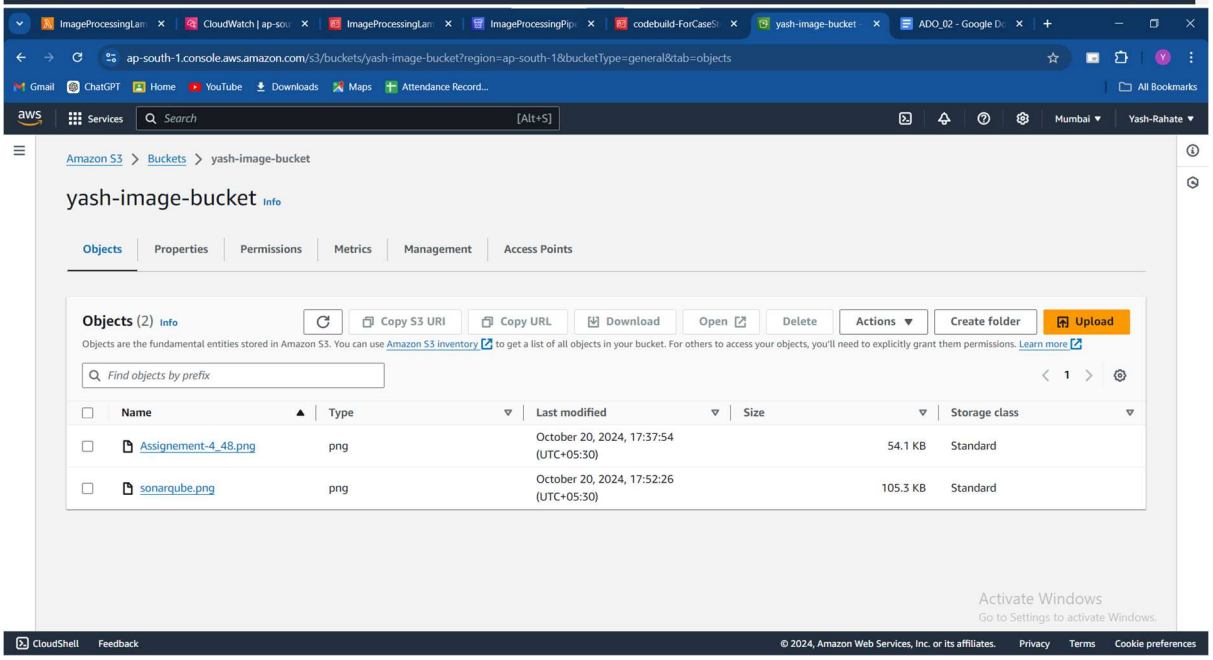
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



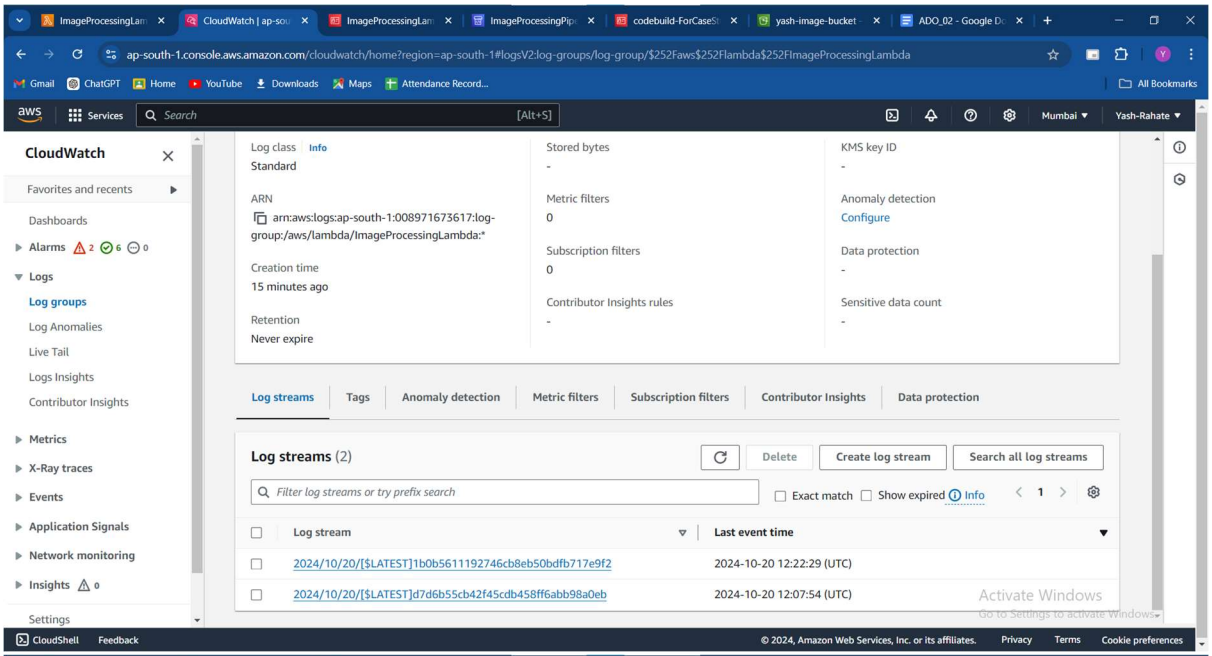
4. Verify that the updated function gets deployed by checking CloudWatch logs after uploading another image.

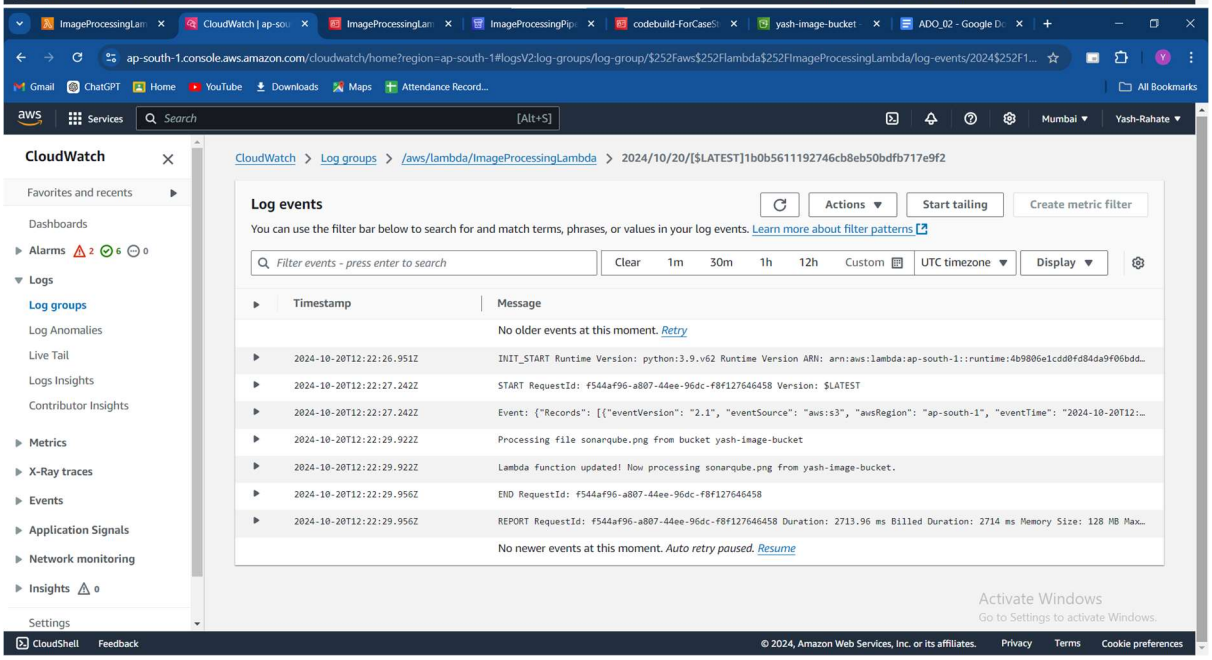
Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



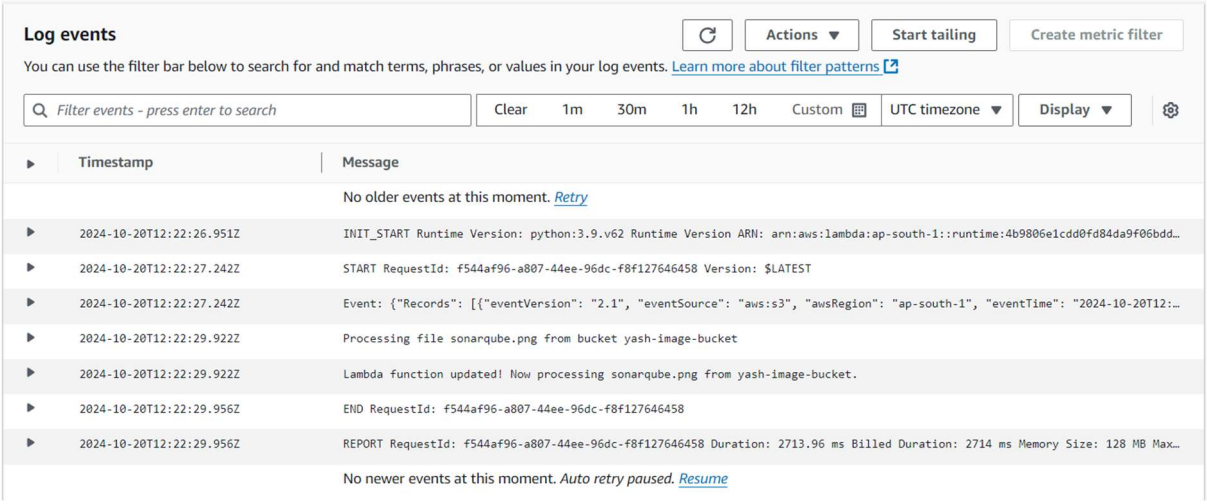


Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in





Vedant Sanap
Roll no.:48
D15A
2022.vedant.sanap@ves.ac.in



Conclusion

This workflow will set up a fully serverless image processing system that triggers an AWS Lambda function whenever a new image is uploaded to S3, and it will automate the deployment using AWS CodePipeline.