

## Case Study

### Serverless Image Processing Workflow

**Name:** Yash Santosh Rahate

**Class:** D15B

**Roll no.:** 48

#### 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**.

**Create bucket info**

Buckets are containers for data stored in S3.

**General configuration**

AWS Region: Asia Pacific (Mumbai) ap-south-1

Bucket name: [Info](#) **yash-image-bucket**

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

**Object Ownership info**

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

Activate Windows  
Go to Settings to activate Windows.

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

## 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)

**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: **ImageProcessingLambda**

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](#) **Python 3.9**

Architecture: [Info](#) **x86\_64**

Activate Windows  
Go to Settings to activate Windows.

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

### 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**

The screenshot shows the AWS Lambda function configuration page for 'ImageProcessingLambda'. The 'Configuration' tab is selected. On the left, the 'Permissions' section is expanded, showing the 'Execution role' configuration. The 'Role name' is set to 'ImageProcessingLambda-role-z411ugmf'. Under the 'Resource summary' section, 'Amazon CloudWatch Logs' is selected. The 'By resource' tab is active, showing two resources with their respective actions:

Resource	Actions
arn:aws:logs:ap-south-1:008971673617:*	Allow: logs>CreateLogGroup
arn:aws:logs:ap-south-1:008971673617:log-group:/aws/lambda/ImageProcessingLambda:*	Allow: logs>CreateLogStream Allow: logs:PutLogEvents Activate Windows Allow: logs:PutLogEvents to activate Windows.

The screenshot shows the AWS IAM console with the 'Permissions' tab selected for the 'ImageProcessingLambda-role-z411ugmf' role. The 'Permissions policies' section displays one policy, 'AmazonS3FullAccess', which is an AWS managed policy. A green success message at the top states: 'Policies have been successfully attached to role.'

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	3
AWSLambdaBasicExecutionRole-a97ecfe...	Customer managed	1
CloudWatchLogsFullAccess	AWS managed	1

## 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))

    # 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!')
}
```

The screenshot shows the AWS Lambda console interface. In the top navigation bar, the URL is ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions/ImageProcessingLambda?tab=code. The main area is titled 'Code source' with an 'Info' tab. Below it is a toolbar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test' (which is currently selected), and 'Deploy'. A search bar says 'Search [Alt+S]'. On the left, there's a sidebar with 'Environment' and a tree view showing 'ImageProcessingLambda' and 'lambda\_function.py'. The code editor window contains the following Python script:

```

1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     # Log the event in CloudWatch
6     print("Event: ", json.dumps(event))
7
8     # Extract S3 bucket and object details
9     s3 = boto3.client('s3')
10    bucket = event['Records'][0]['s3']['bucket']['name']
11    key = event['Records'][0]['s3']['object']['key']
12
13    # Process the image (log details in this case)
14    response = s3.get_object(Bucket=bucket, Key=key)
15    print(f"Processing file {key} from bucket {bucket}")
16

```

- 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](#).

**General configuration**

Event name: ImageUploadEvent  
Event name can contain up to 255 characters.

Prefix - optional: images/

Suffix - optional: jpg

**Event types**

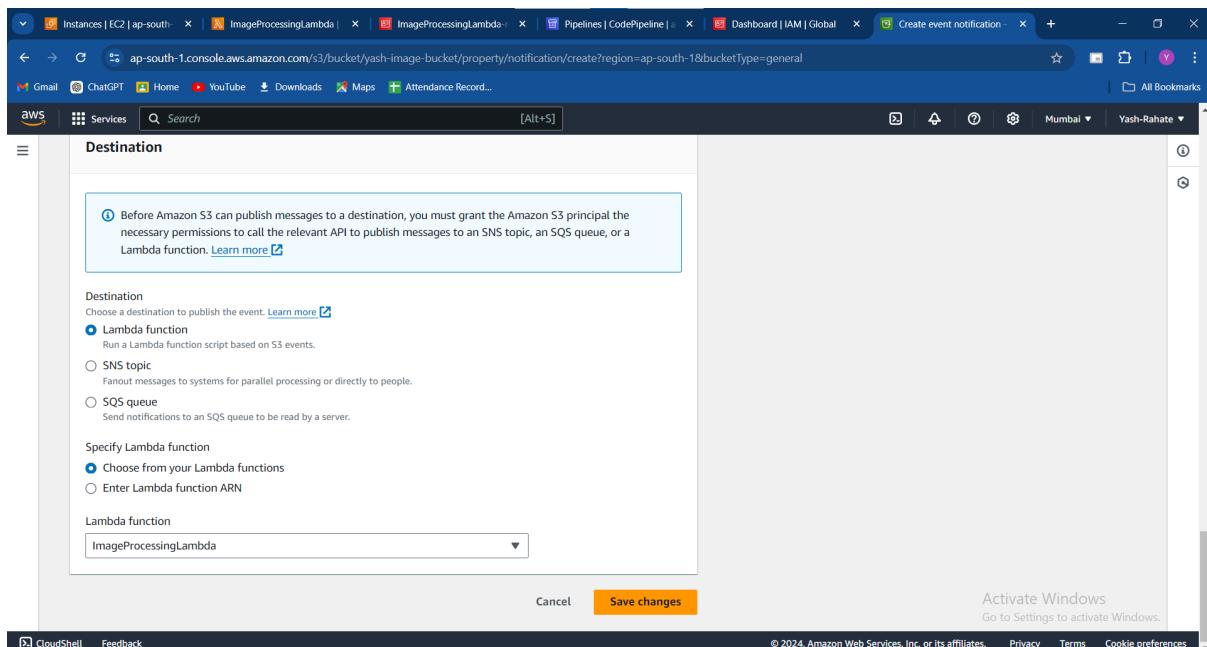
Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

**Object creation**

- All object create events  
s3:ObjectCreated:\*
- Put  
s3:ObjectCreated:Put
- Post  
s3:ObjectCreated:Post
- Copy  
s3:ObjectCreated:Copy
- Multipart upload completed  
s3:ObjectCreated:CompleteMultipartUpload

**Object removal**

- All object removal events  
s3:ObjectRemoved:\*
- Permanently deleted  
s3:ObjectRemoved:Delete
- Delete marker created  
s3:ObjectRemoved:DeleteMarkerCreated



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

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

```
buildspec.yml
version: 0.2
phases:
  install:
    commands:
      - pip install --upgrade awscli
  build:
    commands:
      - zip function.zip lambda_function.py
      - aws lambda update-function-code --function-name ImageProcessingLambda --zip-file fileb://function.zip
```

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

## 2. Create a CodeBuild Project:

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

**Create build project**

**Project configuration**

Project name: ForCaseStudy

Public build access - optional

Enable public build access

Additional configuration

Source

Source 1 - Primary

Source provider

Add source

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

The screenshot shows two overlapping browser windows. The top window is titled 'Processing OAuth request' and is part of the AWS CodeBuild interface. It asks to choose 'Confirm' to save the OAuth token to a Secrets Manager secret, with fields for 'Secret name' (YashRahate) and 'Secret description' (YashRahate). A 'Confirm' button is highlighted in orange. The bottom window shows a GitHub OAuth app configuration, with a 'Connect to GitHub' button also highlighted in orange. Both windows have standard browser navigation and status bars.

- 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).

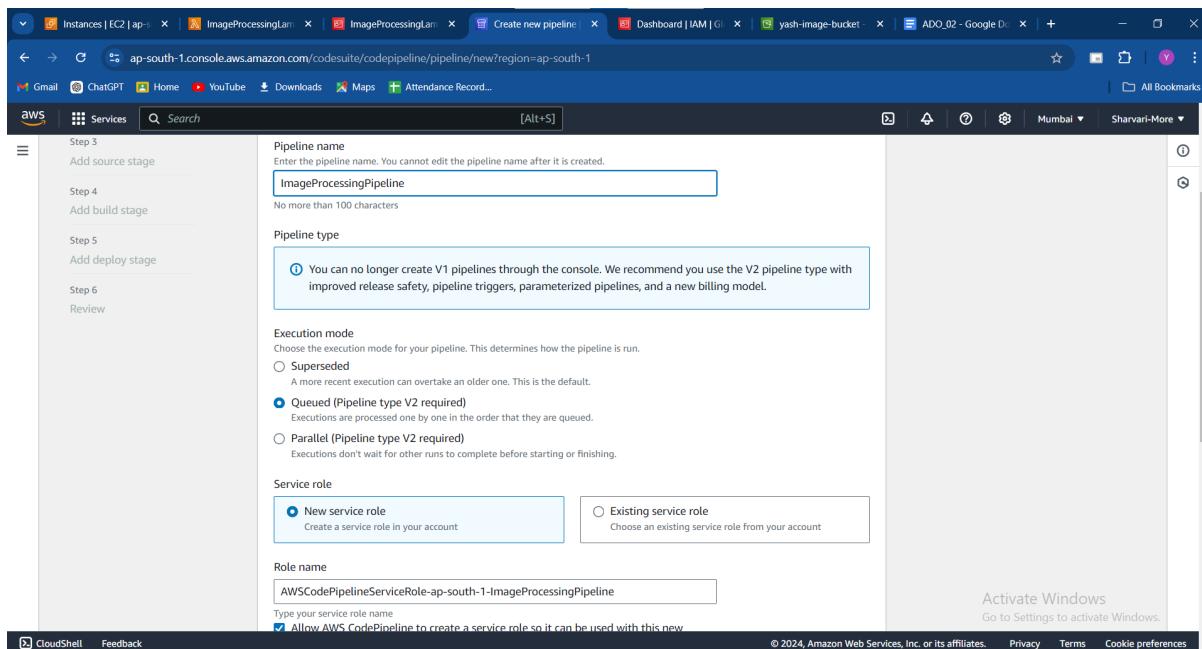
The screenshot shows the AWS Identity and Access Management (IAM) service. On the left, there's a sidebar with navigation links like Dashboard, Access management, and Access reports. The main content area is titled 'codebuild-ForCaseStudy-service-role'. It displays basic information such as creation date (October 20, 2024), last activity (16 minutes ago), ARN (arn:aws:iam::008971673617:role/service-role/codebuild-ForCaseStudy-service-role), and maximum session duration (1 hour). Below this, the 'Permissions' tab is selected, showing three attached policies: 'AWSLambda\_FullAccess' (AWS managed), 'CodeBuildBasePolicy-ForCaseStudy-ap-s...', and 'CodeBuildSecretsManagerSourceCredent...'. There are also tabs for Trust relationships, Tags, Last Accessed, and Revoke sessions.

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

The screenshot shows the 'Create build project' wizard in the AWS CodeBuild service. The first step is 'Buildspec'. It provides two options: 'Insert build commands' (which stores build commands as build project configuration) and 'Use a buildspec file' (which stores build commands in a YAML-formatted buildspec file, which is selected). There's also a 'Buildspec name - optional' field where 'buildspec.yml' is entered. Below this, there's a 'Batch configuration' section with an optional checkbox for defining batch configuration. The next section is 'Artifacts', with a primary artifact named 'Artifact 1 - Primary'. At the bottom right, there are activation links for Windows.

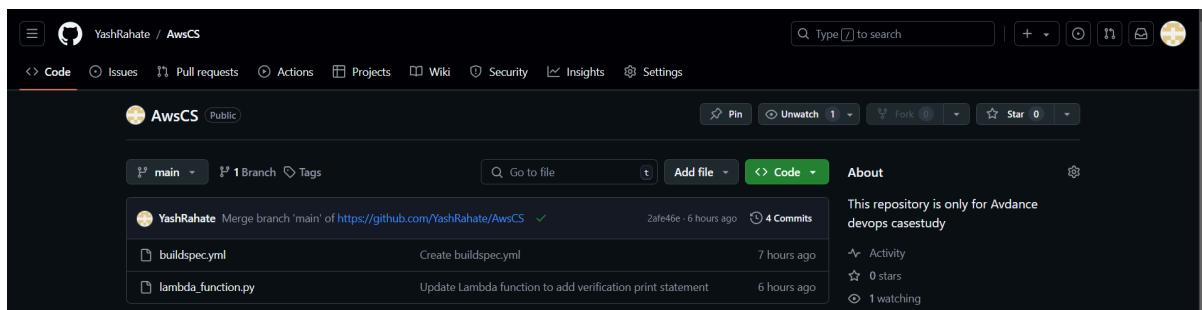
## Step 5: Set Up AWS CodePipeline to Automate Lambda Deployment

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



### 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).

The screenshot shows the AWS CodePipeline 'Create new pipeline' wizard at Step 3: Add source stage. The 'Source provider' dropdown is set to 'GitHub (Version 1)'. A green success message box displays: 'You have successfully authenticated your account.' Below it, a note in a blue box states: 'The GitHub (Version 1) action is not recommended. The selected action uses OAuth apps to access your GitHub repository. This is no longer the recommended method. Instead, choose the GitHub (Version 2) action to access your repository by creating a connection. Connections use GitHub Apps to manage authentication and can be shared with other resources. [Learn more](#)'.

#### 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.

The screenshot shows the AWS CodePipeline 'Create new pipeline' wizard at Step 3: Add build stage. The 'Build provider' dropdown is set to 'AWS CodeBuild'. Under 'Build type', the 'Single build' option is selected. Other options include 'Batch build' (which triggers multiple builds as a single execution). The 'Region' dropdown is set to 'Asia Pacific (Mumbai)'.

#### 5. Deploy Stage (Deploy to Lambda):

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

Step 1 Choose creation option

Step 2 Choose pipeline settings

Step 3 Add source stage

Step 4 Add build stage

Step 5 Add deploy stage

Step 6 Review

**Add deploy stage**

Step 5 of 6

**Deploy - optional**

Deploy provider Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Configure automatic rollback on stage failure

Enable automatic retry on stage failure

Cancel Previous Skip deploy stage Next

Activate Windows  
Go to Settings to activate Windows.

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Step 1 Choose creation option

Step 2 Choose pipeline settings

Step 3 Add source stage

Step 4 Add build stage

Step 5 Add deploy stage

Step 6 Review

**Add deploy stage**

Step 5 of 6

**Deploy - optional**

Deploy provider Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Configure automatic rollback on stage failure

Enable automatic retry on stage failure

**Skip deployment stage**

Your pipeline will not include a deployment stage. Are you sure you want to skip this stage?

Cancel Skip

Cancel Previous Skip deploy stage Next

Activate Windows  
Go to Settings to activate Windows.

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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

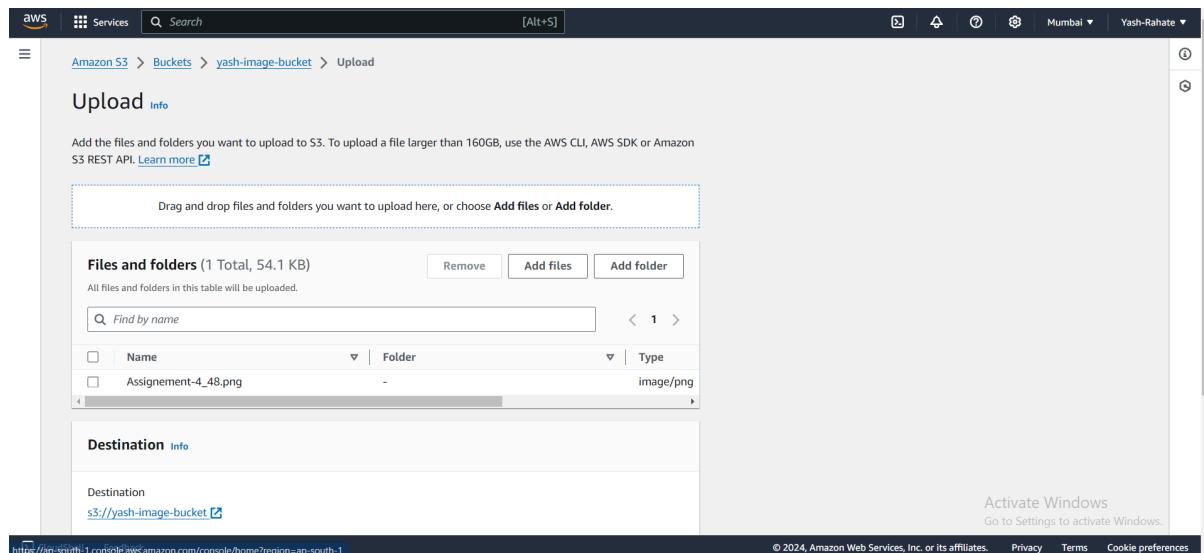
The screenshot shows the AWS CodePipeline console with two stages: Source and Build.

- Source Stage:** Pipeline execution ID: 33e2d16f-ed87-4795-9232-57fca6022582. Status: Succeeded. Last updated: 20 minutes ago. Action: GitHub (Version 1). Sub-action: Create buildspec.yml.
- Build Stage:** Pipeline execution ID: 33e2d16f-ed87-4795-9232-57fca6022582. Status: Succeeded. Last updated: 1 minute ago. Action: AWS CodeBuild. Sub-action: Create buildspec.yml.

## 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



## 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).

**Log events**

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Timestamp	Message
2024-10-20T12:07:54.424Z	INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:4b9806e1cdd0fd84da9f06bdd...
2024-10-20T12:07:54.701Z	START RequestId: 7f6b17d5-58db-4ffd-9e3a-283fda40761c Version: \$LATEST
2024-10-20T12:07:54.701Z	Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "ap-south-1", "eventTime": "2024-10-20T12:..."}]}
2024-10-20T12:07:57.364Z	Processing file Assignment-4_48.png from bucket yash-image-bucket
2024-10-20T12:07:57.418Z	END RequestId: 7f6b17d5-58db-4ffd-9e3a-283fda40761c
2024-10-20T12:07:57.418Z	REPORT RequestId: 7f6b17d5-58db-4ffd-9e3a-283fda40761c Duration: 2716.99 ms Billed Duration: 2717 ms Memory Size: 128 MB Max...

No newer events at this moment. Auto retry paused. [Resume](#)

## Step 7: Verify CodePipeline Automation

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

```

lambda_function.py > lambda_handler
1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     # Log the event in CloudWatch
6     print("Event: ", json.dumps(event))
7
8     # Extract S3 bucket and object details
9     s3 = boto3.client('s3')
10    bucket = event['Records'][0]['s3']['bucket']['name']
11    key = event['Records'][0]['s3']['object']['key']
12
13    # Process the image (log details in this case)
14    response = s3.get_object(Bucket=bucket, Key=key)
15    print(f"Processing file {key} from bucket {bucket}")
16
17    # New print statement for verification
18    print(f"Lambda function updated! Now processing {key} from {bucket}.")
19
20    return {
21        'statusCode': 200,
22        'body': json.dumps('Image processed successfully!')
23    }

```

TERMINAL

```

PS C:\Users\acer\Desktop\caseStudy48> git add lambda_function.py
PS C:\Users\acer\Desktop\caseStudy48> git commit -m "Update Lambda function to add verification print statement"
1 file changed, 3 insertions(+)
PS C:\Users\acer\Desktop\caseStudy48> git push origin main

```

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!')
    }

```

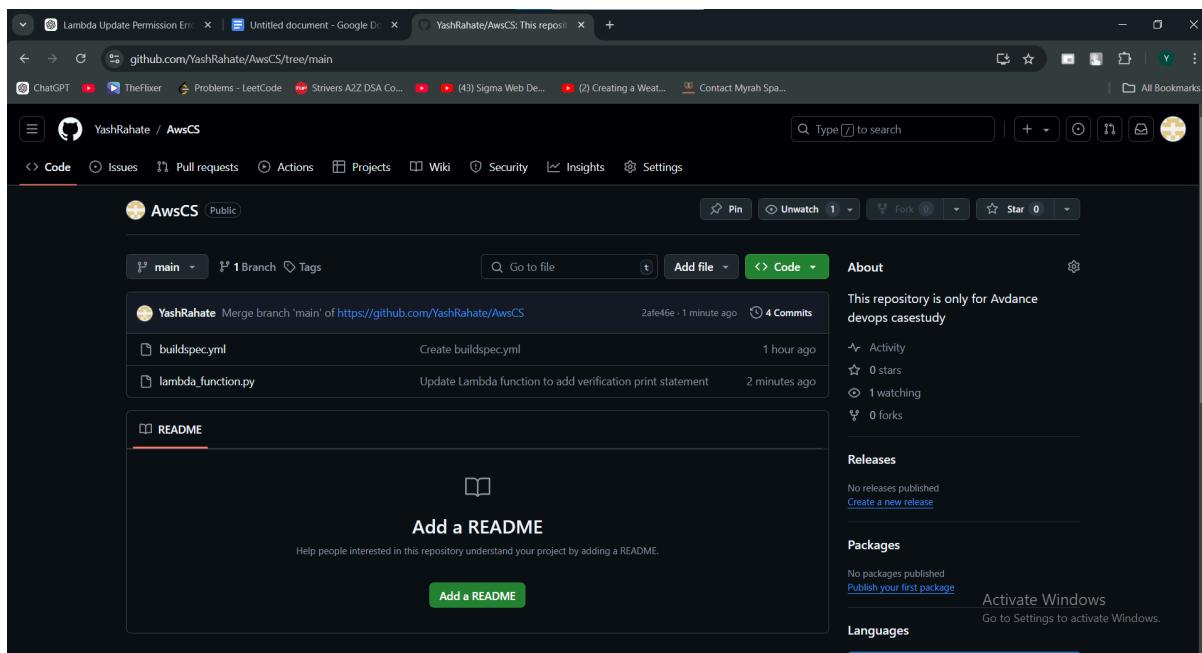
## 2. Push the changes to the GitHub or CodeCommit repository.

```

git add lambda_function.py
git commit -m "Update Lambda function to add verification print statement"
git push origin main

```

```
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.

The screenshot shows the AWS CodePipeline console with the following details:

- Pipeline Type:** V2
- Execution Mode:** QUEUED
- Source Stage:** Succeeded. Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da. It shows a GitHub commit (2afe46ed) from the 'main' branch of https://github.com/YashRahate/AwsCS.
- Build Stage:** In progress.

The sidebar on the left lists pipeline stages: Source, Artifacts, Build, Deploy, Pipeline, and Settings. The Pipeline stage is currently selected. The bottom right corner displays a Windows activation message: "Activate Windows" and "Go to Settings to activate Windows".

The screenshot shows the AWS CodePipeline console with a successful pipeline run. The pipeline consists of three stages: Source, Build, and Deploy. The Source stage (GitHub) has succeeded just now. The Build stage (AWS CodeBuild) is also in progress. The Deploy stage (AWS Lambda) has succeeded just now. Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da.

**Source**  
GitHub (Version 1) Succeeded - Just now  
zafe46ed View details

**Build** In progress  
Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da

**Build**  
AWS CodeBuild In progress - Just now  
View details

**Activate Windows**  
Go to Settings to activate Windows.

**Deploy** Succeeded  
Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da Start rollback

**Build**  
AWS Lambda Succeeded - Just now  
View details

**Activate Windows**  
Go to Settings to activate Windows.

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

The screenshot shows two consecutive screenshots of the AWS S3 console interface.

**Screenshot 1: Upload Process**

This screenshot shows the "Upload" step in the S3 console. The user is uploading files to the "yash-image-bucket". A file named "sonarqube.png" (image/png, 105.3 KB) is selected for upload. The destination is set to "s3://yash-image-bucket".

**Screenshot 2: Bucket Contents**

This screenshot shows the "Objects" tab in the S3 console for the "yash-image-bucket". It displays two objects: "Assignment-4\_48.png" (png, 54.1 KB, last modified: October 20, 2024, 17:37:54 UTC+05:30) and "sonarqube.png" (png, 105.3 KB, last modified: October 20, 2024, 17:52:26 UTC+05:30).

Screenshot of the AWS CloudWatch Log Groups interface for the ImageProcessingLambda function:

**Log class:** Standard

**ARN:** arn:aws:logs:ap-south-1:008971673617:log-group:/aws/lambda/ImageProcessingLambda:\*

**Creation time:** 15 minutes ago

**Retention:** Never expire

**Metric filters:** 0

**Subscription filters:** 0

**Contributor Insights rules:** -

**KMS key ID:** -

**Anomaly detection:** Configure

**Data protection:** -

**Sensitive data count:** -

**Log streams:** (2)

- 2024/10/20/[SLATEST]1b0b5611192746cb8eb50bdfb71e9f2 (Last event time: 2024-10-20 12:22:29 (UTC))
- 2024/10/20/[SLATEST]d7d6b55cb42f45cd458ff6abb98a0eb (Last event time: 2024-10-20 12:07:54 (UTC))

**CloudWatch Log groups:**

**Log events:**

Timestamp	Message
2024-10-20T12:22:26.951Z	INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:4b9806e1cdd0fd84da9f0bddd...
2024-10-20T12:22:27.242Z	START RequestId: f544af96-a807-44ee-96dc-f8f127646458 Version: \$LATEST
2024-10-20T12:22:27.242Z	Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "ap-south-1", "eventTime": "2024-10-20T12:22:27.242Z", "s3": {"region": "ap-south-1", "bucket": "yash-image-bucket", "object": "sonarqube.png"}]}
2024-10-20T12:22:29.922Z	Processing file sonarqube.png from bucket yash-image-bucket
2024-10-20T12:22:29.922Z	Lambda function updated! Now processing sonarqube.png from yash-image-bucket.
2024-10-20T12:22:29.956Z	END RequestId: f544af96-a807-44ee-96dc-f8f127646458 Duration: 2713.96 ms Billed Duration: 2714 ms Memory Size: 128 MB Max...
2024-10-20T12:22:29.956Z	REPORT RequestId: f544af96-a807-44ee-96dc-f8f127646458 Duration: 2713.96 ms Billed Duration: 2714 ms Memory Size: 128 MB Max...

**Log events:**

Timestamp	Message
2024-10-20T12:22:26.951Z	INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:4b9806e1cdd0fd84da9f0bddd...
2024-10-20T12:22:27.242Z	START RequestId: f544af96-a807-44ee-96dc-f8f127646458 Version: \$LATEST
2024-10-20T12:22:27.242Z	Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "ap-south-1", "eventTime": "2024-10-20T12:22:27.242Z", "s3": {"region": "ap-south-1", "bucket": "yash-image-bucket", "object": "sonarqube.png"}]}
2024-10-20T12:22:29.922Z	Processing file sonarqube.png from bucket yash-image-bucket
2024-10-20T12:22:29.922Z	Lambda function updated! Now processing sonarqube.png from yash-image-bucket.
2024-10-20T12:22:29.956Z	END RequestId: f544af96-a807-44ee-96dc-f8f127646458 Duration: 2713.96 ms Billed Duration: 2714 ms Memory Size: 128 MB Max...
2024-10-20T12:22:29.956Z	REPORT RequestId: f544af96-a807-44ee-96dc-f8f127646458 Duration: 2713.96 ms Billed Duration: 2714 ms Memory Size: 128 MB Max...

**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.