

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

Buckets are containers for data stored in S3.

General configuration

AWS Region
Europe (Stockholm) eu-north-1

Bucket type [Info](#)

General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)
vedant-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

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)

Lambda > Functions > Create function

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
Enter a name that describes the purpose of your function.
`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](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
`Python 3.9`

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
 x86_64

arm64

Activate Windows
Go to Settings to activate Windows.

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

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**

The screenshot shows the AWS Lambda function configuration page for 'ImageProcessingLambda'. The left sidebar lists various function settings like General configuration, Triggers, Permissions (which is selected), Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, and Asynchronous invocation. The main panel is titled 'Execution role' and shows the role name 'ImageProcessingLambda-role-z411ugmf'. Below this is a 'Resource summary' section with a dropdown menu set to 'Amazon CloudWatch Logs' (3 actions, 2 resources). Under the 'By resource' tab, two log group entries are listed:

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

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

The screenshot shows the AWS IAM Role details page for 'ImageProcessingLambda-role-z411ugmf'. The 'Permissions' tab is selected, displaying three attached policies: 'AmazonS3FullAccess', 'AWSLambdaBasicExecutionRole', and 'CloudWatchLogsFullAccess'. A green banner at the top indicates that policies have been successfully attached.

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:

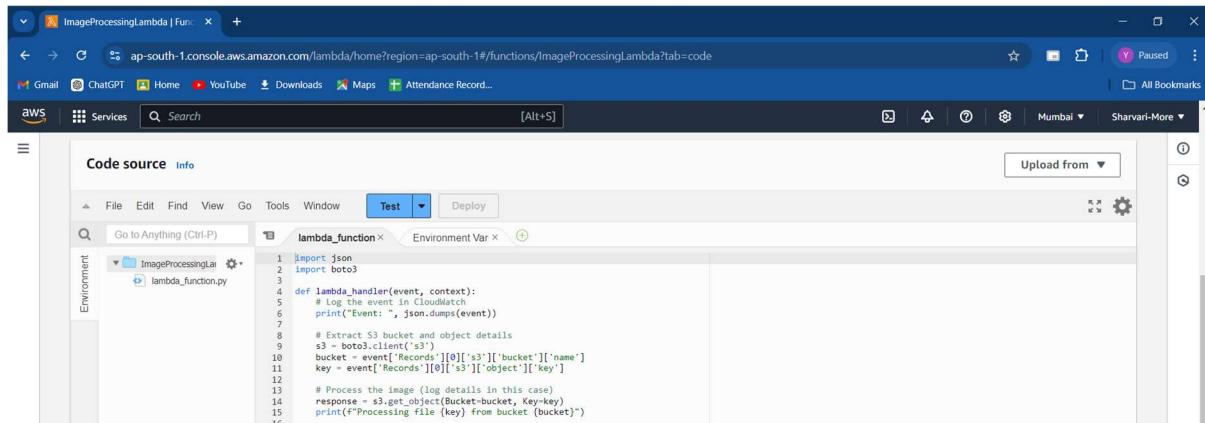
CODE:

```
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 function editor. The title bar says "ImageProcessingLambda | Function". The URL in the address bar 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" and contains the Python code provided above. There are tabs for "Environment" and "Info". At the top, there are "File", "Edit", "Find", "View", "Go", "Tools", "Window", "Test" (which is selected), and "Deploy" buttons. On the right side, there is a "Upload from" button and a gear icon.

- 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

The screenshot shows the 'Create event notification' configuration page in the AWS S3 console. The top navigation bar includes links for Instances | EC2 | ap-south-1, ImageProcessingLambda, ImageProcessingLambda, Pipelines | CodePipeline, Dashboard | IAM | Global, Create event notification, Gmail, ChatGPT, Home, YouTube, Downloads, Maps, Attendance Record, and All Bookmarks. The user is signed in as Yash-Rahate.

The main content area has a title 'Create event notification' with an 'Info' link. A sub-header states: 'To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications.' Below this, there are two sections: 'General configuration' and 'Event types'.

General configuration:

- Event name:** ImageUploadEvent (maximum 255 characters)
- Prefix - optional:** images/ (limits notifications to objects starting with 'images/')
- Suffix - optional:** jpg (limits notifications to objects ending with '.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

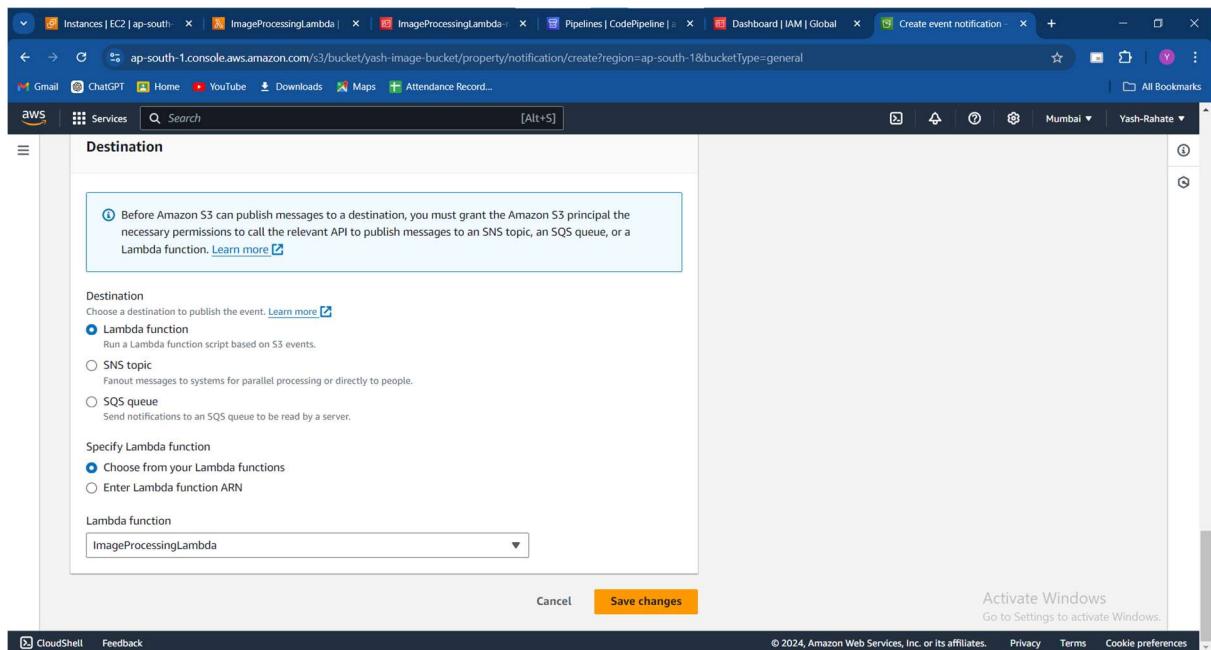
Object removal:

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

Activate Windows
Go to Settings to activate Windows.

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

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

The screenshot shows two code editor panes. The top pane displays the `buildspec.yml` file:version: 0.2phases:install:commands:- pip install --upgrade awsclibuild:commands:- zip function.zip lambda_function.py- aws lambda update-function-code --function-name ImageProcessingLambda --zip-file fileb://function.zipThe bottom pane displays the `lambda_function.py` file:import jsonimport boto3def 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.

The screenshot shows the 'Create build project' wizard in the AWS CodeBuild console. The 'Project configuration' step is selected. The 'Project name' field contains 'ForCaseStudy'. Under 'Public build access - optional', there is a checkbox for 'Enable public build access' which is unchecked. In the 'Additional configuration' section, there is a link for 'Description, Build badge, Concurrent build limit, tags'. The 'Source' step is shown below, with 'Source 1 - Primary' selected and a 'Source provider' dropdown. A watermark for 'Activate Windows' is visible in the bottom right corner.

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

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

The screenshot shows the AWS CloudShell interface with multiple tabs open. The active tab is titled 'Source' under the 'CodeBuild' section. It displays the configuration for a GitHub source provider:

- Source provider:** GitHub
- Credential:** Default source credential (selected)
- Repository:** Repository in my GitHub account (selected)
- GitHub repository:** https://github.com/YashRahate/AwsCS
- Source version - optional**: Entrée a pull request, branch, commit ID, tag or reference and a commit ID.

At the top left, there is a 'Processing OAuth request' dialog box:

Choose Confirm to save your oauth token to a Secrets Manager secret

Secret name: YashRahate

Secret description: YashRahate

Buttons: Cancel, Confirm

On the right side of the CloudShell, there is a sidebar with the following options:

- OAuth app: Connect project to GitHub using an OAuth app
- Managed token
- Connect to GitHub

The status bar at the bottom indicates: Activate Windows Go to Settings to activate Windows.

- 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

The screenshot shows the AWS IAM service management console. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management', 'Access reports', and 'Access Analyzer'. The main content area is titled 'codebuild-ForCaseStudy-service-role' with a 'Summary' tab selected. It displays basic information such as creation date (October 20, 2024, 17:03 UTC+05:30), last activity (16 minutes ago), ARN (arn:aws:iam::008971673617:role/service-role/codebuild-ForCaseStudy-service-role), and maximum session duration (1 hour). Below the summary, there are tabs for 'Permissions', 'Trust relationships', 'Tags', 'Last Accessed', and 'Revoke sessions'. Under the 'Permissions' tab, it says 'Permissions policies (3) Info'. There are three policies listed: 'AWSLambda_FullAccess' (AWS managed), 'CodeBuildBasePolicy-ForCaseStudy-ap-s...', and 'CodeBuildSecretsManagerSourceCredent...'. Each policy has a 'Type' column indicating it's 'Customer managed'.

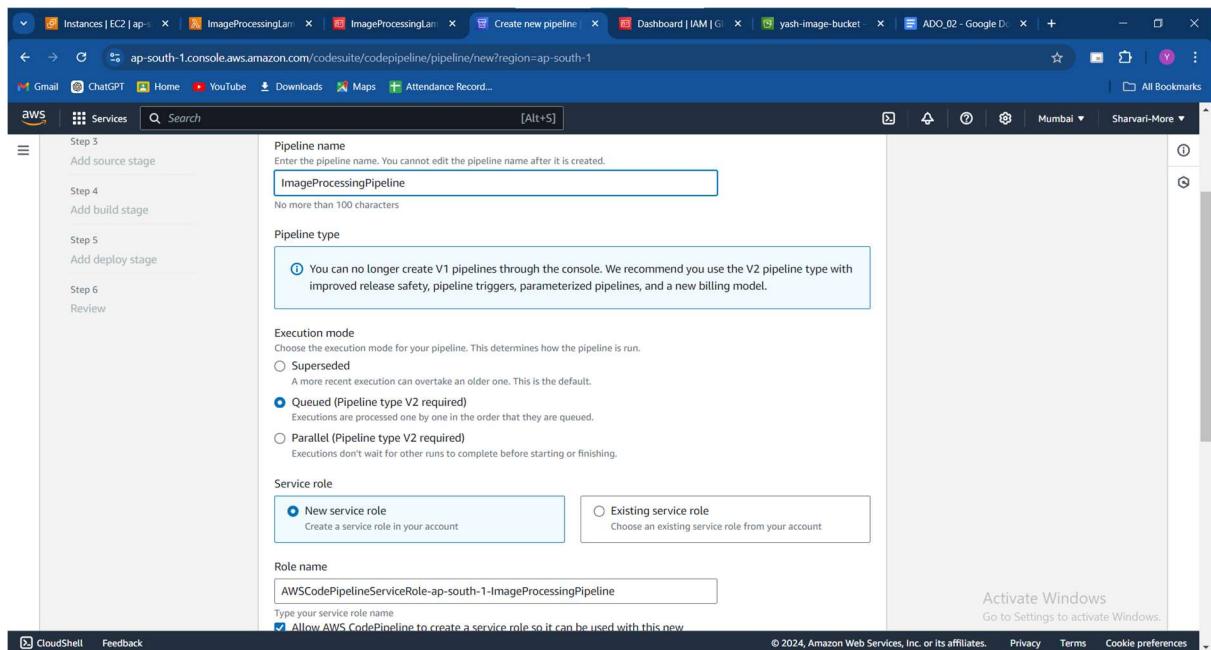
- Specify the **buildspec.yml** file from your GitHub repository.

The screenshot shows the AWS CodeBuild service management console. The main content area is titled 'Buildspec'. It contains several configuration sections: 'Build specifications' (with options for 'Insert build commands' or 'Use a buildspec file'), 'Buildspec name - optional' (with a text input field containing 'buildspec.yml'), 'Batch configuration' (with an option to 'Define batch configuration - optional'), 'Artifacts' (with a 'Add artifact' button), and 'Artifact 1 - Primary' (with a 'Remove' button). The bottom right corner of the page has a message: 'Activate Windows Go to Settings to activate Windows.'

Step 5: Set Up AWS CodePipeline to Automate Lambda Deployment

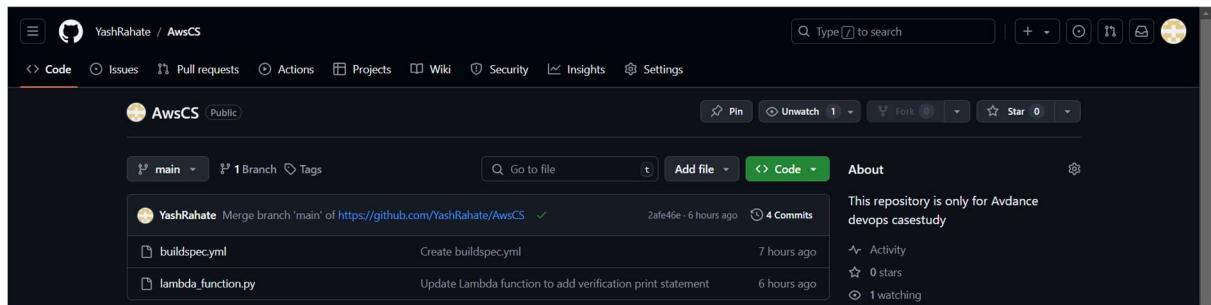
1. Go to the **CodePipeline** service and click **Create Pipeline**.
2. **Pipeline Settings:**
 - **Pipeline Name:** **ImageProcessingPipeline**
 - **Service Role:** Allow CodePipeline to create a new role.

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

The screenshot shows the AWS CodePipeline 'Create new pipeline' wizard at Step 4: Add source stage. The 'Source provider' dropdown is set to 'GitHub (Version 1)'. A green success message box says 'You have successfully authenticated your account.' Below it, a blue info 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:

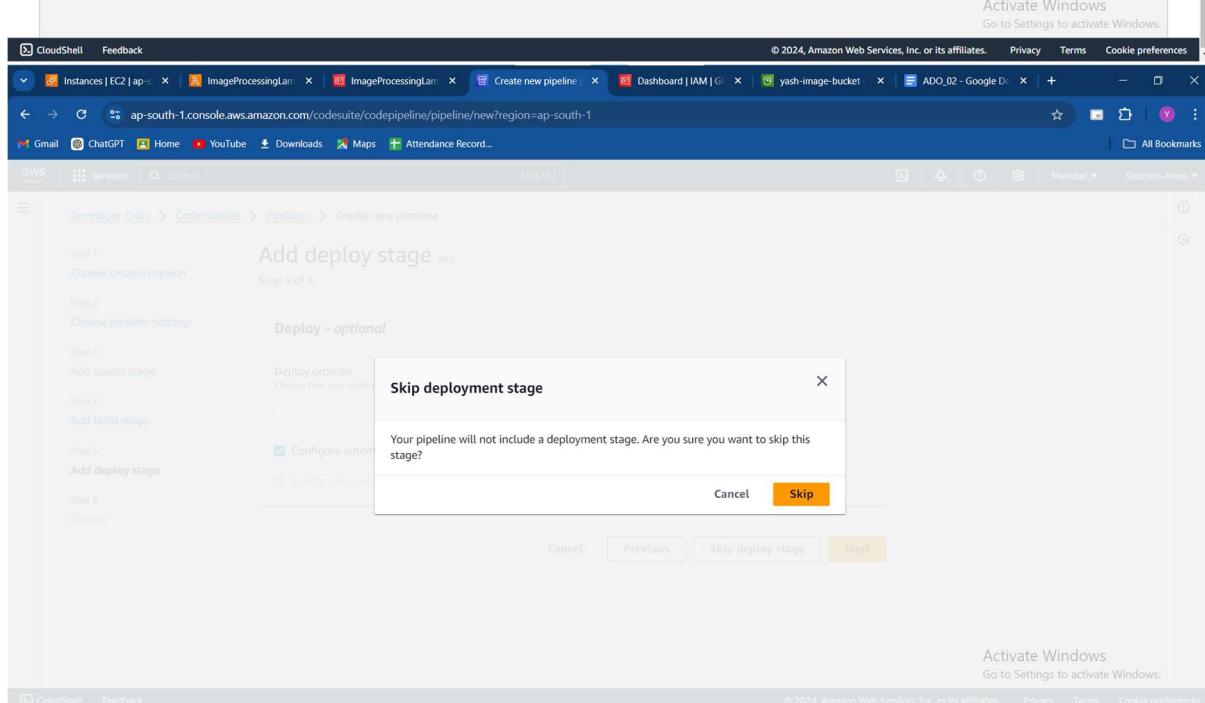
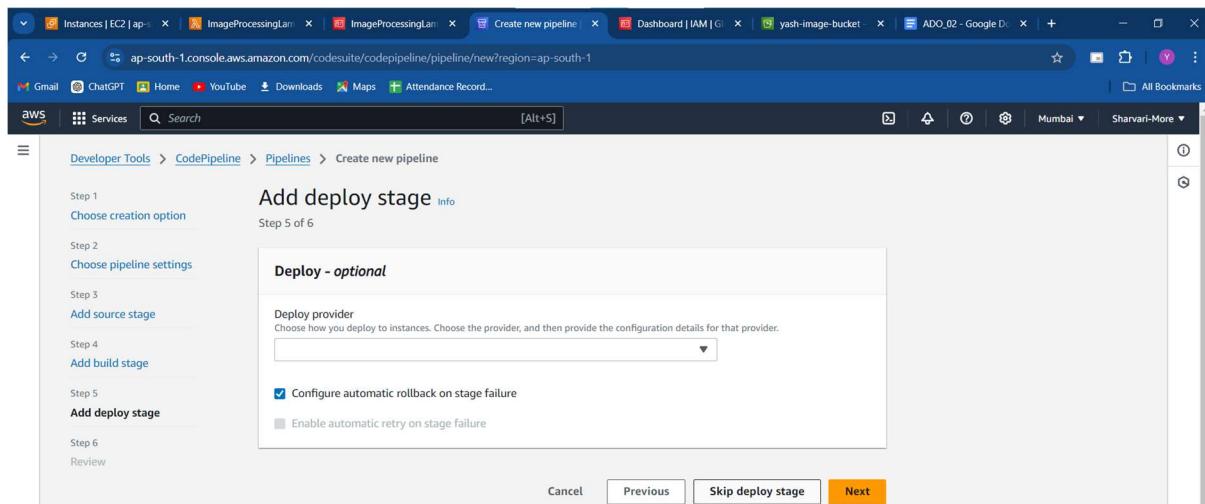
- In your CodePipeline, add **CodeBuild** as the **Build Stage** (instead of a Deploy Stage).
- This will allow CodePipeline to trigger the CodeBuild project, which will run the `buildspec.yml` commands to package and deploy the Lambda function.

The screenshot shows the AWS CodePipeline 'Create new pipeline' wizard at Step 5: Add build stage. The 'Build provider' dropdown is set to 'AWS CodeBuild'. The 'Project name' field contains 'ForCaseStudy'. The 'Build type' section shows 'Single build' selected. Other artifacts like 'Region' and 'Input artifacts' are also visible.

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

The screenshot shows the AWS CodePipeline console with two pipeline executions. The top execution is at the Source stage (GitHub) and the Build stage (AWS CodeBuild). The bottom execution is also at the Source stage (GitHub) and the Build stage (AWS CodeBuild). A tooltip 'Activate Windows' is visible on the right side of the interface.

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

The screenshot shows the AWS S3 'Upload' interface. At the top, there's a navigation bar with 'Search' and 'Mumbai' selected. Below it, the path 'Amazon S3 > Buckets > yash-image-bucket > Upload' is shown. The main area is titled 'Upload' with a 'info' link. A note says 'Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)'.

A large dashed box allows dragging and dropping files. Below it, a table lists 'Files and folders (1 Total, 54.1 KB)'. It shows one item: 'Assignment-4_48.png' (image/png). There are 'Remove', 'Add files', and 'Add folder' buttons. A search bar 'Find by name' is also present.

The 'Destination' section shows 's3://yash-image-bucket'. On the right, there are links to 'Activate Windows' and 'Go to Settings to activate Windows.', along with copyright information: '© 2024, Amazon Web Services, Inc. or its affiliates.' and links to 'Privacy', 'Terms', and 'Cookie preferences'.

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

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar is titled "CloudWatch" and includes sections for Dashboards, Alarms, Logs (selected), Log groups, Metrics, X-Ray traces, Events, Application Signals, Network monitoring, and Insights. The main content area shows log events for the "/aws/lambda/ImageProcessingLambda" log group. The log entries are as follows:

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": "2024-10-20T12:07:54.701Z", "s3": {"region": "ap-south-1", "bucket": "yash-image-bucket", "object": "4_48.png"}]}
2024-10-20T12:07:57.364Z	Processing file Assignement-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...

Below the log entries, a message states "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).

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

The screenshot shows the Visual Studio Code interface. The left sidebar has icons for file operations like Open, Save, Find, and Run. The Explorer sidebar shows a folder named 'CASESTUDY48' containing 'buildspec.yml' and 'lambda_function.py'. The main code editor window displays the following Python code:

```
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    }
```

The terminal at the bottom shows the command line history:

```
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}.")
```

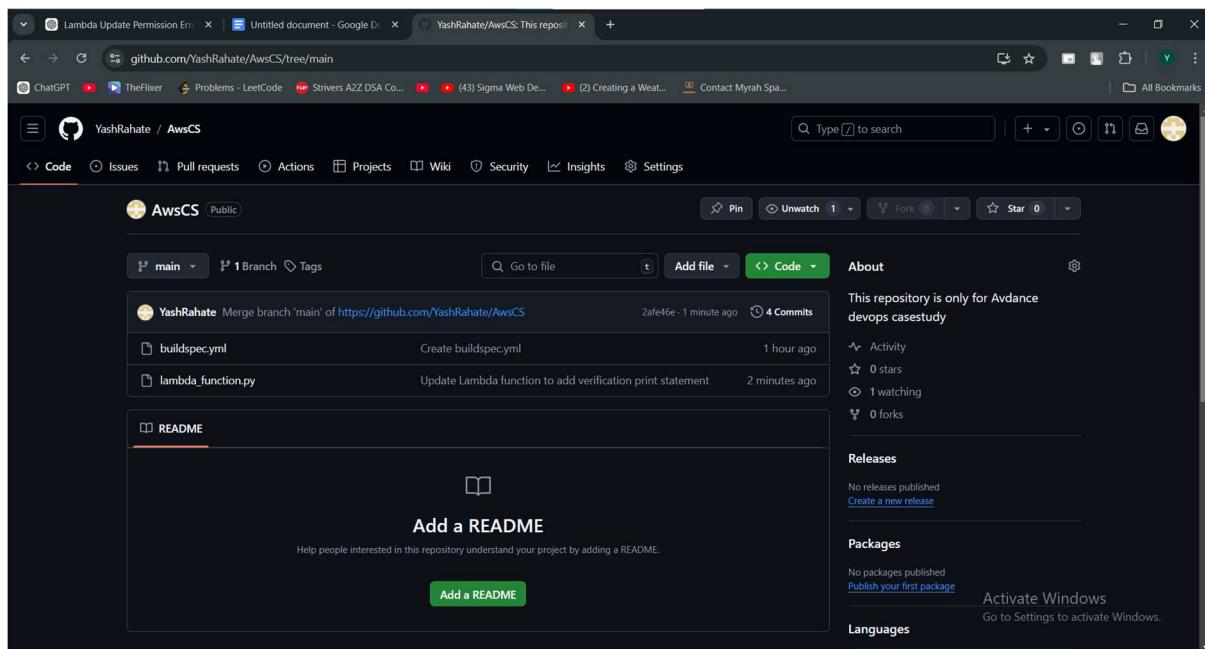
```
return {
    'statusCode': 200,
    'body': json.dumps('Image processed successfully!')
}
```

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
```

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

The screenshot shows the AWS CodePipeline console interface. The pipeline name is "ImageProcessingPipeline". The pipeline type is V2 and the execution mode is QUEUED. The Source stage is listed as Succeeded, with a Pipeline execution ID of 002f0132-23e5-4616-bdb0-74e9e30908da. The Build stage is listed as In progress. The pipeline has a GitHub (Version 1) source provider. A tooltip for the GitHub provider indicates it is a Source Merge branch main of https://github.com/YashPahate/AwsCS. There are buttons for Notify, Edit, Stop execution, Clone pipeline, and Release change. The sidebar on the left shows the pipeline structure: Source (CodeCommit), Artifacts (CodeArtifact), Build (CodeBuild), Deploy (CodeDeploy), Pipeline (CodePipeline), and Settings. The Pipeline section includes links for Getting started, History, and Settings.

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

The screenshot shows the AWS CodePipeline console for a pipeline named "ImageProcessingPipeline". The pipeline has three stages: Source, Build, and Deploy. The Source stage is configured to use GitHub (Version 1) as the provider, and it has succeeded just now. The Build stage is currently in progress, managed by AWS CodeBuild, and has just started. The Deploy stage is not yet active. The pipeline execution ID is 002f0132-23e5-4616-bdb0-74e9e30908da. On the right side of the interface, there are two green circular icons with checkmarks, likely indicating successful steps or stages.

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

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

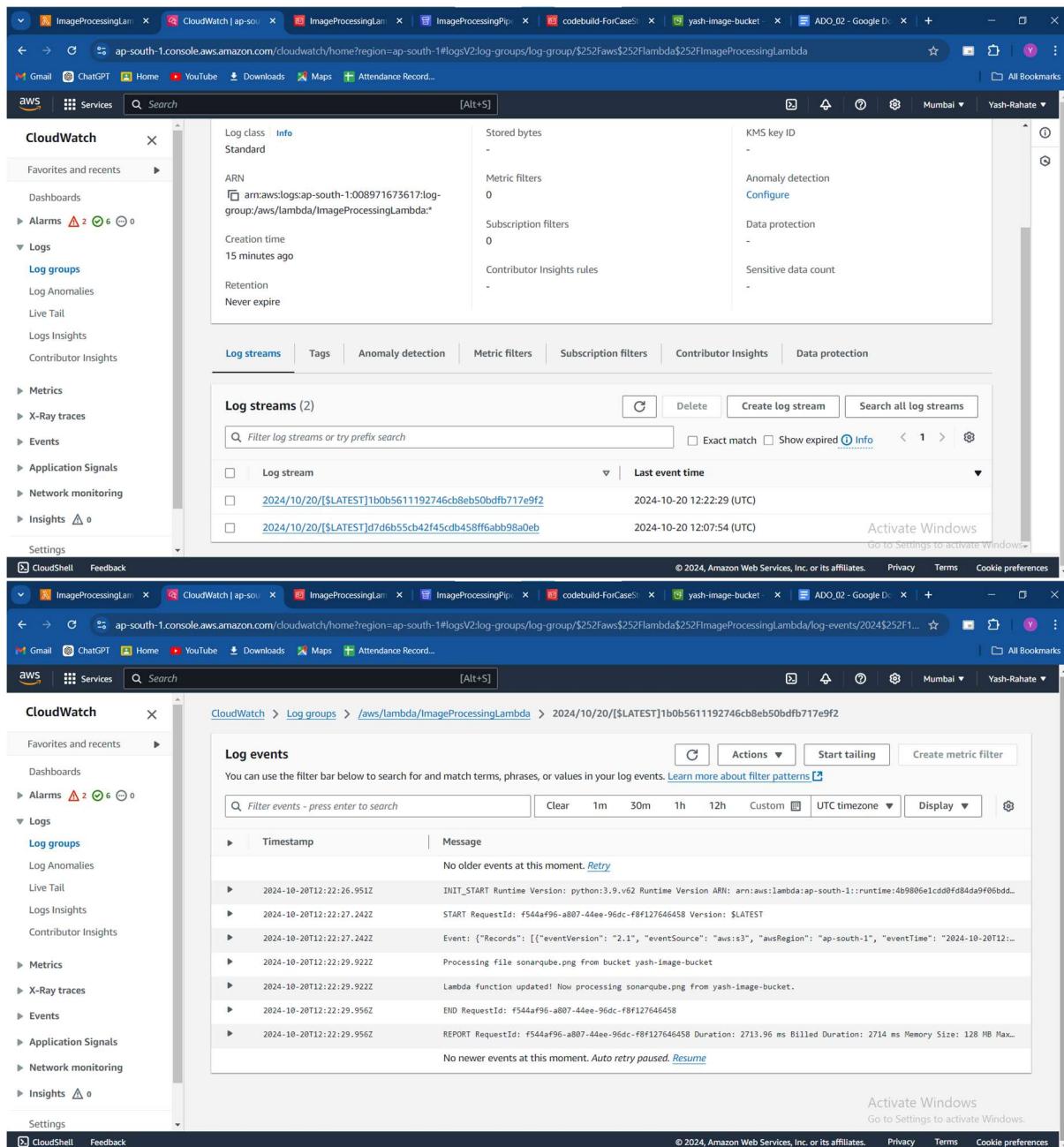
Screenshot 1: Upload Process

The user is uploading files to the 'yash-image-bucket'. The 'Upload' page shows a file named 'sonarqube.png' being selected for upload. The file details are: Name: sonarqube.png, Type: image/png, Size: 105.3 KB. The destination is set to 's3://yash-image-bucket'. A message at the bottom indicates that files larger than 160GB should be uploaded using the AWS CLI, AWS SDK or Amazon S3 REST API.

Screenshot 2: Bucket Contents

The user has completed the upload and is now viewing the contents of the 'yash-image-bucket'. The 'Objects' tab is selected, showing two objects: 'Assigment-4_48.png' and 'sonarqube.png'. Both files are of type 'png' and have a size of 54.1 KB. The last modified date for both files is October 20, 2024, 17:37:54 (UTC+05:30). The storage class for both files is 'Standard'.

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



The screenshot shows the AWS CloudWatch Logs interface. On the left, the navigation menu is expanded to show 'Log groups' under 'Logs'. The main panel displays the details for the log group '/aws/lambda/ImageProcessingLambda'. It shows the ARN (arn:aws:logs:ap-south-1:008971673617:log-group:/aws/lambda/ImageProcessingLambda), which is highlighted. Other details include 'Standard' log class, 'Info' level, 'Stored bytes' (0), 'Metric filters' (0), 'Subscription filters' (0), 'Creation time' (15 minutes ago), 'Retention' (Never expire), 'Contributor Insights rules' (0), and 'KMS key ID' (None). Below this, the 'Log streams' tab is selected, showing two log streams: '2024/10/20/[LATEST]1b0b5611192746cb8eb50bdfb717e9f2' (Last event time: 2024-10-20 12:22:29 (UTC)) and '2024/10/20/[LATEST]d7d6b55cb42f45cd8ff6abb98a0eb' (Last event time: 2024-10-20 12:07:54 (UTC)). The interface includes a search bar, filter options, and various AWS service icons at the top.

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

Log events		Action	Actions ▾	Start tailing	Create metric filter					
Filter events - press enter to search		Clear	1m	30m	1h	12h	Custom	UTC timezone ▾	Display ▾	⚙️
▶	Timestamp	Message								
No older events at this moment. Retry										
▶	2024-10-20T12:22:26.951Z	INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:4b9806e1cdd0fd84da9f06bdd...								
▶	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": {"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								
▶	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...								
No newer events at this moment. Auto retry paused. Resume										

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.