Case Study

Serverless Image Processing Workflow

Name: Eesha Chavan
Class: D15A
Roll no.: 8

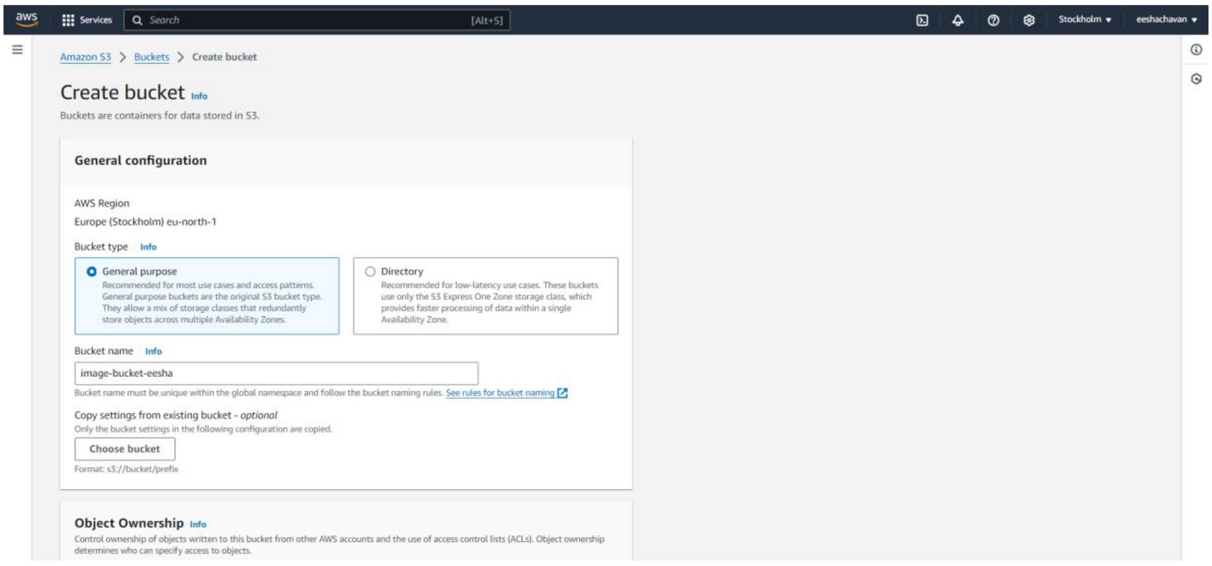
Problem Statement:

- **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**.



Amazon S3 > Buckets > Create bucket

Create bucket Info

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

image-bucket-eesha

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

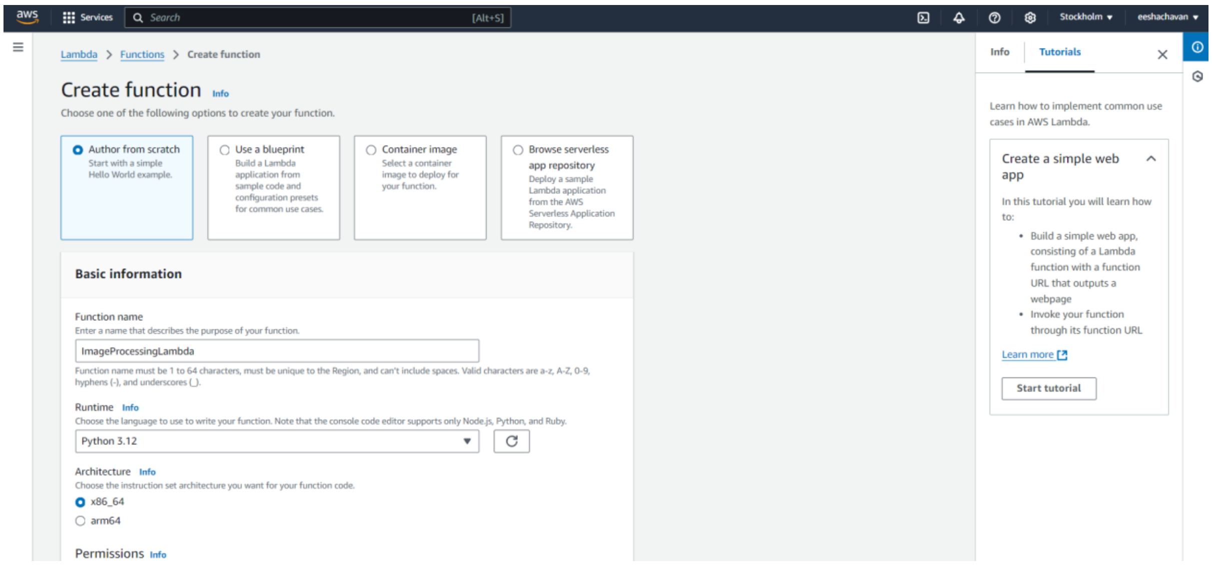
Format: s3://bucket/prefix

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.

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.

Browse serverless app repository: Deploy a sample Lambda application from the AWS Serverless Application Repository.

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 3.12

Architecture Info

Choose the instruction set architecture you want for your function code.

[✓ x86_64] arm64

Permissions Info

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

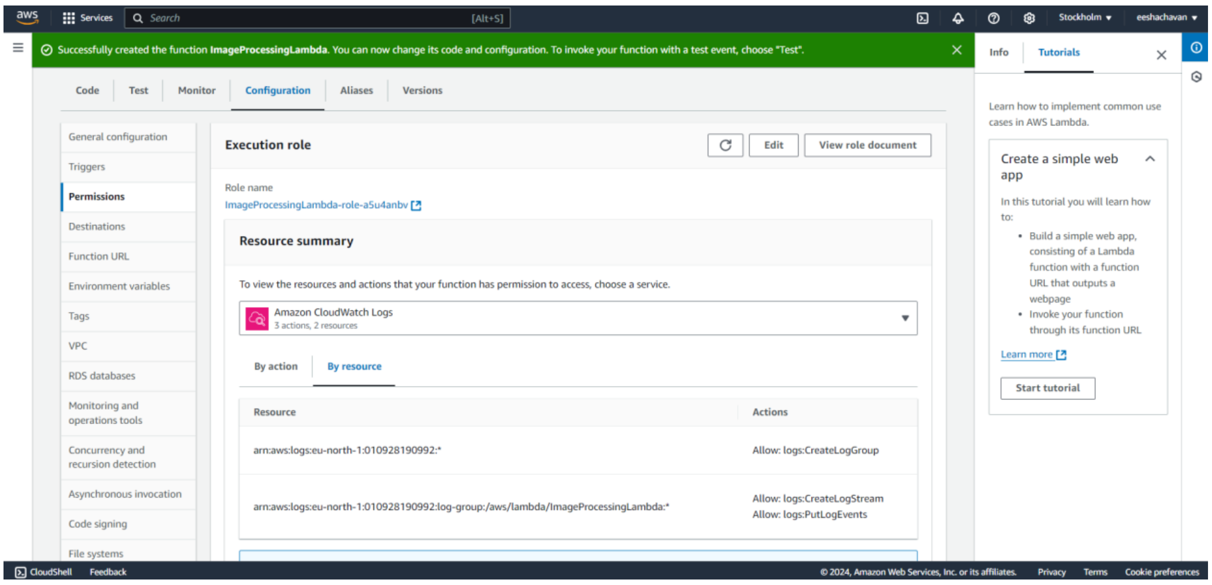
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

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**



Successfully created the function **ImageProcessingLambda**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Code | Test | Monitor | Configuration | Aliases | Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
Monitoring and operations tools
Concurrency and recursion detection
Asynchronous invocation
Code signing
File systems

Execution role

Edit | View role document

Role name
ImageProcessingLambda-role-a5u4anbv

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs
3 actions, 2 resources

By action | By resource

Resource	Actions
arn:aws:logs:eu-north-1:010928190992:*	Allow: logs:CreateLogGroup
arn:aws:logs:eu-north-1:010928190992:log-group:/aws/lambda/ImageProcessingLambda:*	Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Info | Tutorials

Learn how to implement common use cases in AWS Lambda.

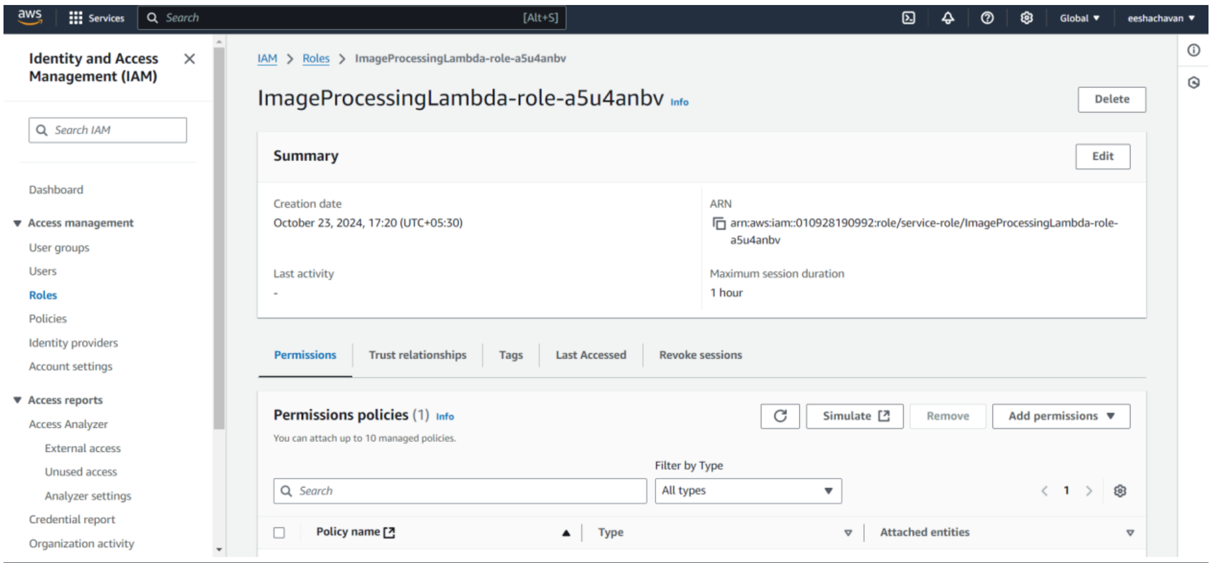
Create a simple web app

In this tutorial you will learn how to:
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

Start tutorial

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



IAM > Roles > ImageProcessingLambda-role-a5u4anbv

ImageProcessingLambda-role-a5u4anbv Info

Delete

Summary Edit

Creation date October 23, 2024, 17:20 (UTC+05:30)	ARN arn:aws:iam::010928190992:role/service-role/ImageProcessingLambda-role-a5u4anbv
Last activity -	Maximum session duration 1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

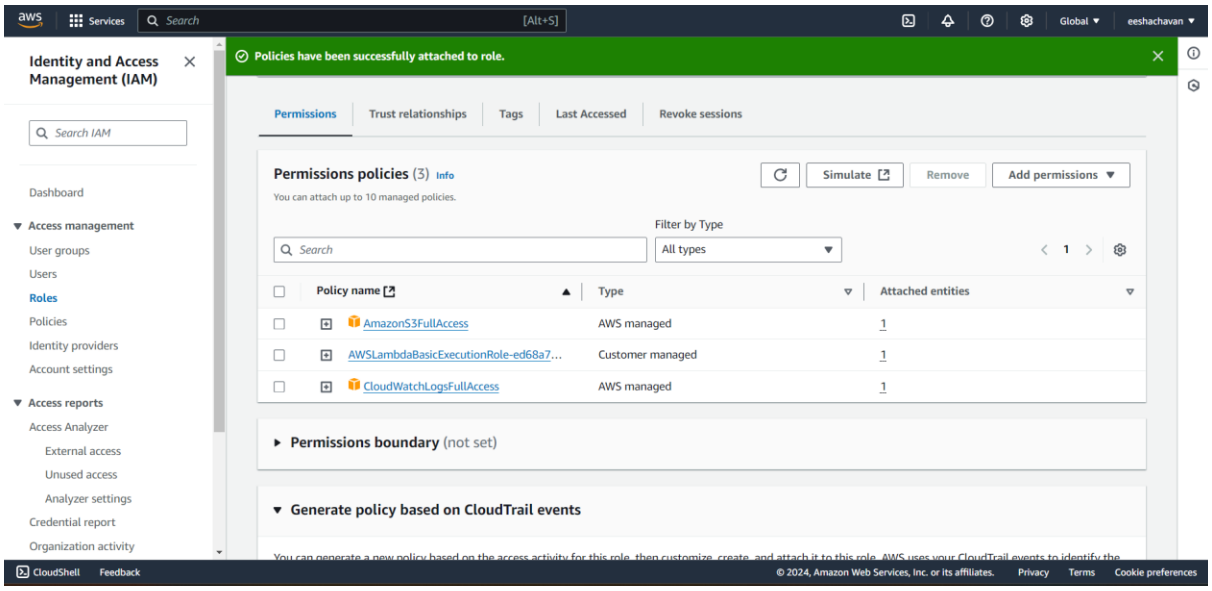
Permissions policies (1) Info

You can attach up to 10 managed policies.

Simulate | Remove | Add permissions

Filter by Type: All types

Policy name	Type	Attached entities



Policies have been successfully attached to role.

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (3) Info

You can attach up to 10 managed policies.

Simulate | Remove | Add permissions

Filter by Type: All types

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	1
AWSLambdaBasicExecutionRole-ed68a7...	Customer managed	1
CloudWatchLogsFullAccess	AWS managed	1

Permissions boundary (not set)

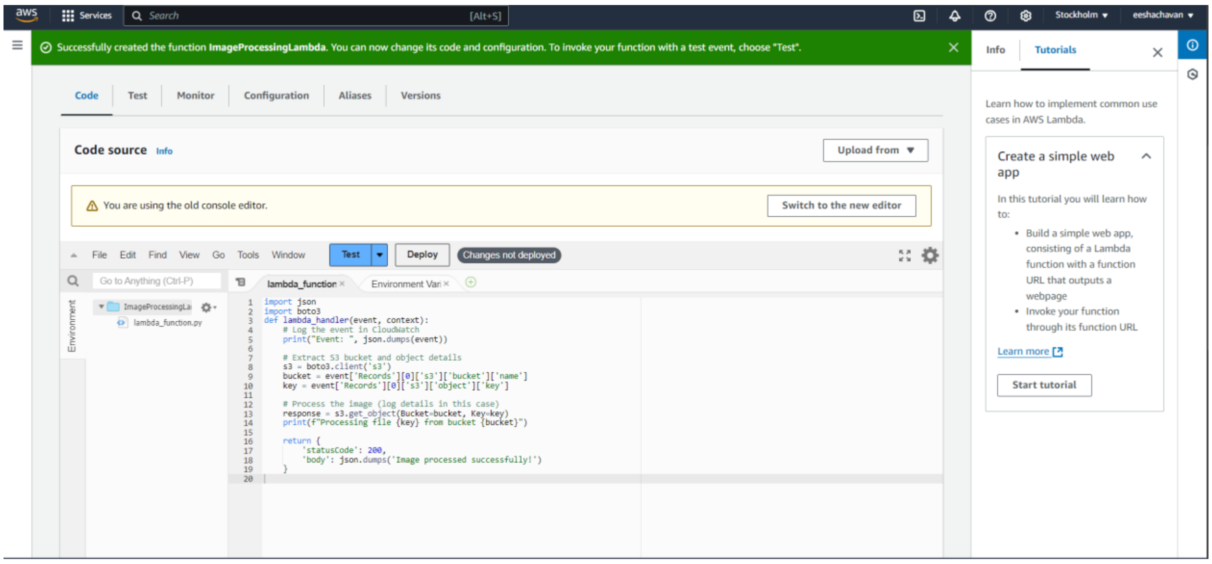
Generate policy based on CloudTrail events

You can generate a new policy based on the access activity for this role, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the

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!')
    }
```



Successfully created the function **ImageProcessingLambda**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Code source

You are using the old console editor.

```
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!')
    }
```

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

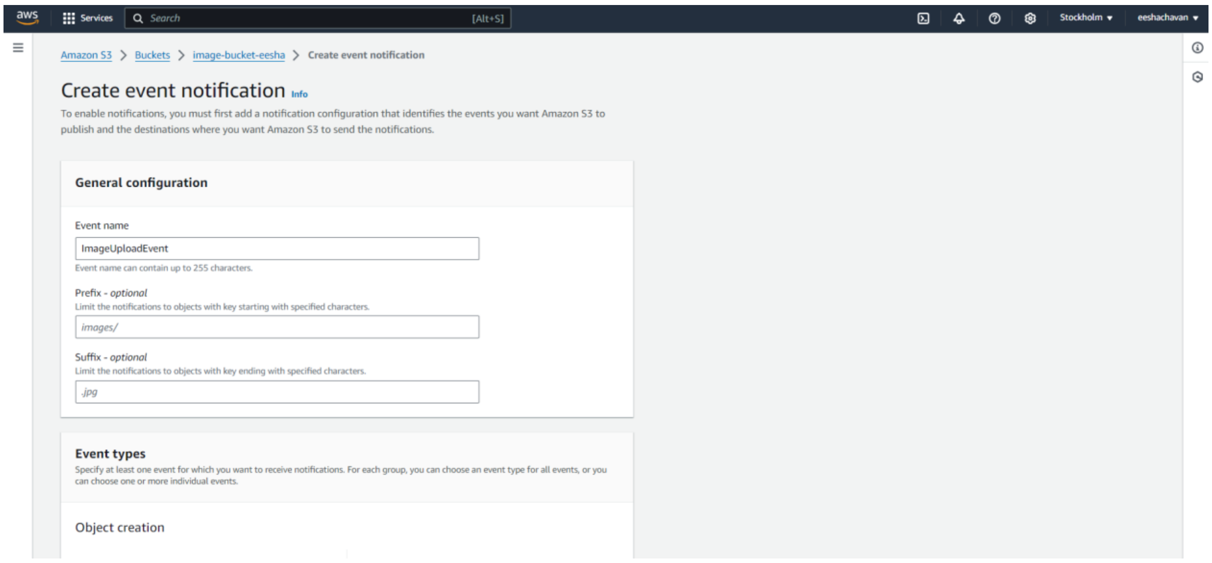
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

- 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-bucket-eesha).
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.


Amazon S3 > Buckets > image-bucket-eesha > Create event notification

Create event notification Info

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.

General configuration

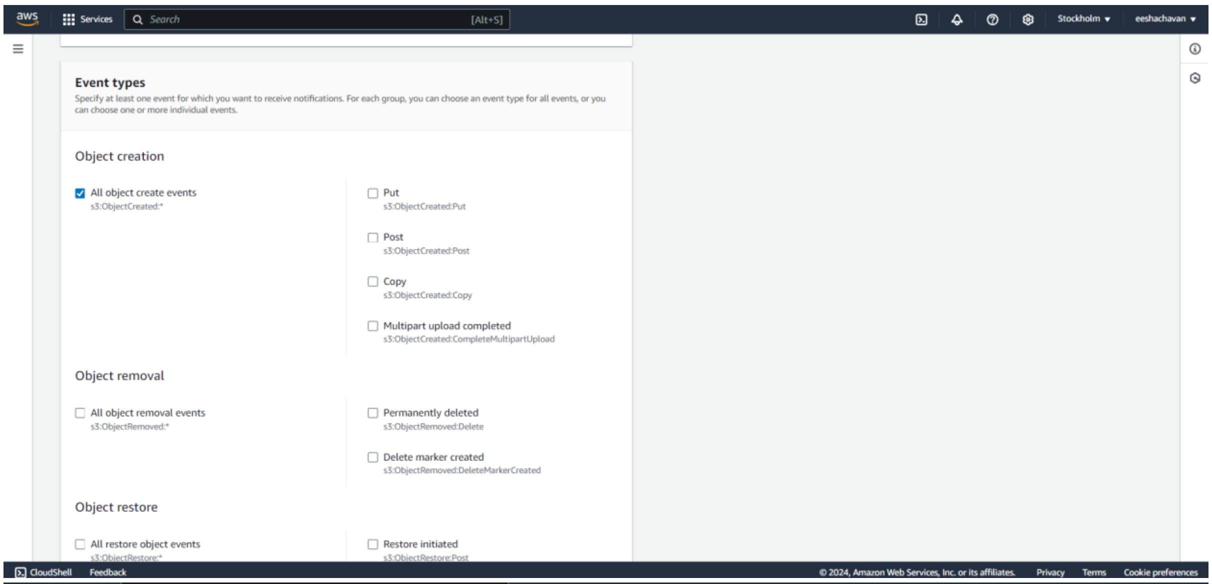
Event name
ImageUploadEvent
Event name can contain up to 255 characters.

Prefix - *optional*
Limit the notifications to objects with key starting with specified characters.
images/

Suffix - *optional*
Limit the notifications to objects with key ending with specified characters.
.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


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

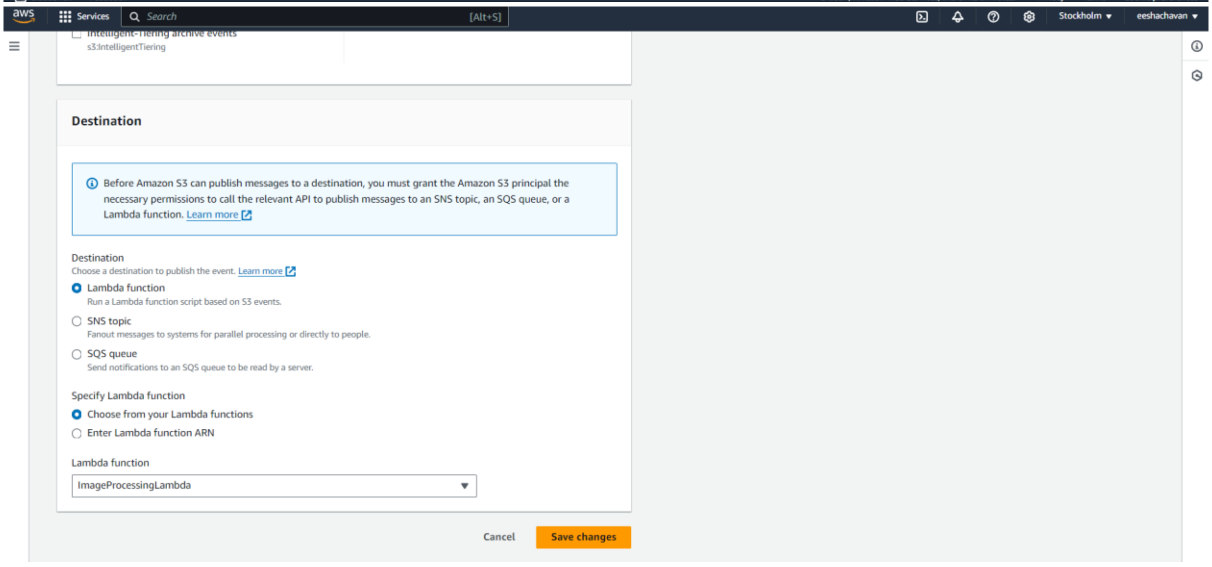
Object removal

- [ ] All object removal events — s3:ObjectRemoved:*
- [ ] Permanently deleted — s3:ObjectRemoved:Delete
- [ ] Delete marker created — s3:ObjectRemoved:DeleteMarkerCreated

Object restore

- [ ] All restore object events — s3:ObjectRestore:*
- [ ] Restore initiated — s3:ObjectRestore:Post

© 2024, Amazon Web Services, Inc. or its affiliates.


- [ ] Intelligent-Tiering archive events — s3:IntelligentTiering

Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. Learn more

Destination
Choose a destination to publish the event. Learn more
- [●] Lambda function — Run a Lambda function script based on S3 events.
- [ ] SNS topic — Fanout messages to systems for parallel processing or directly to people.
- [ ] SQS queue — Send notifications to an SQS queue to be read by a server.

Specify Lambda function
- [●] Choose from your Lambda functions
- [ ] Enter Lambda function ARN

Lambda function
ImageProcessingLambda

Cancel | Save changes

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:

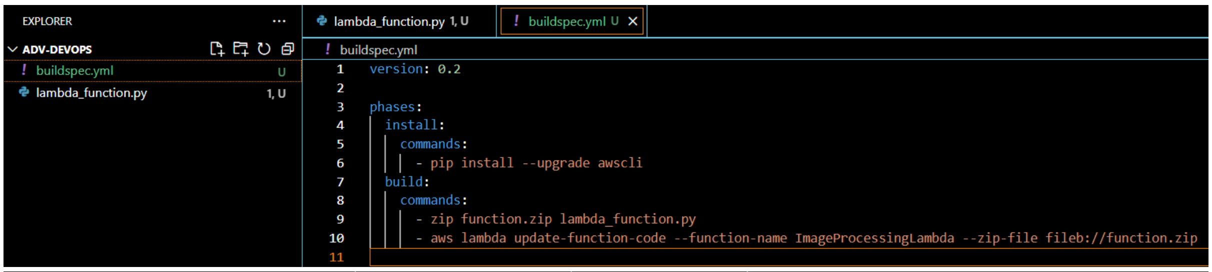
         - pip install --upgrade awscli

     build:

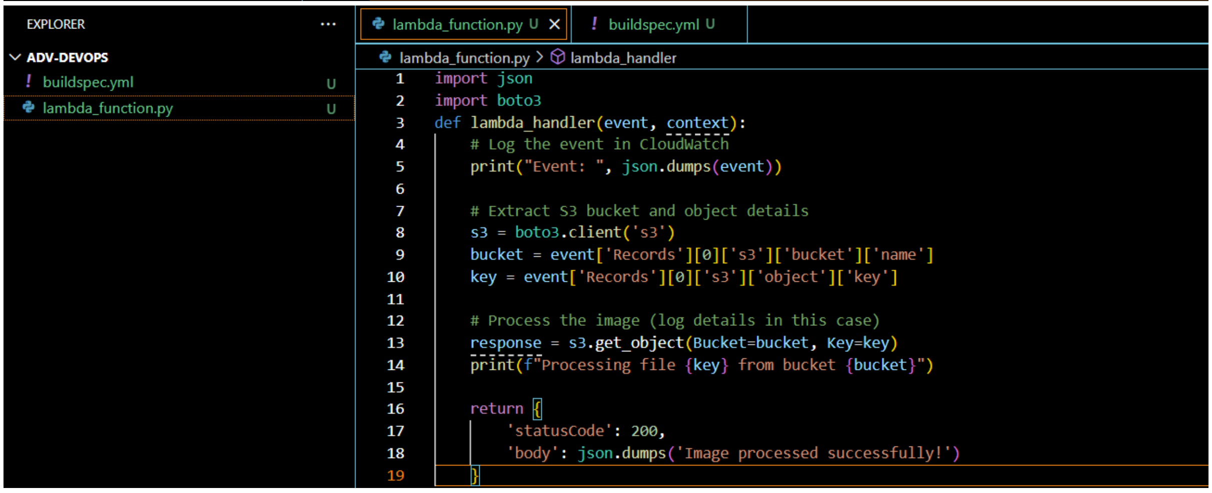
       commands:

         - zip function.zip lambda_function.py

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



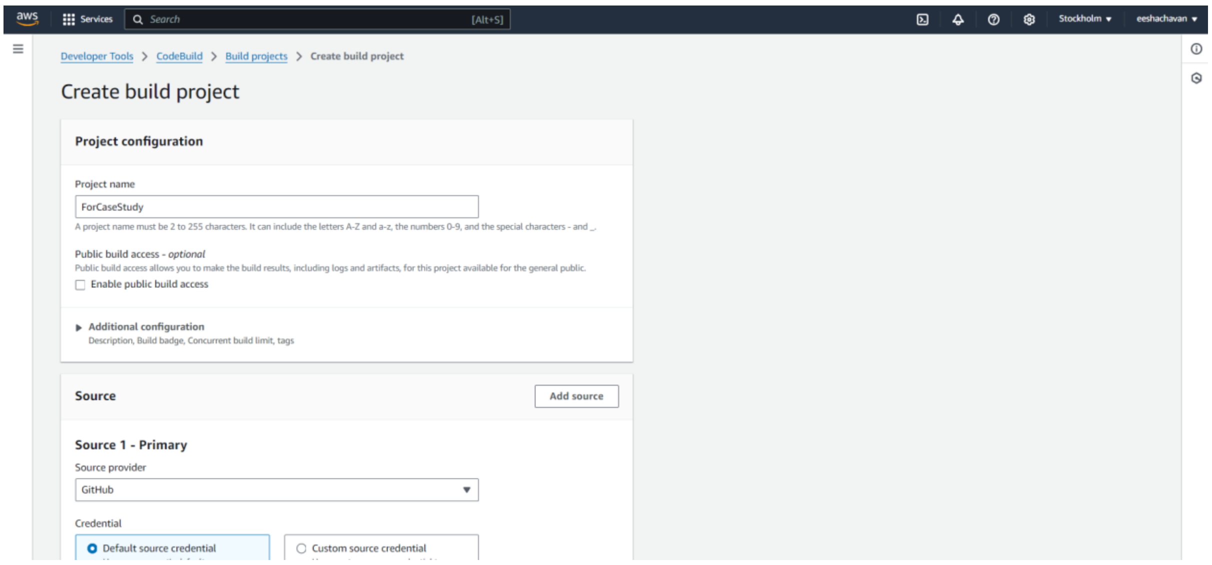
```yaml
version: 0.2

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
```



```python
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.



Developer Tools > CodeBuild > Build projects > Create build project

Create build project

Project configuration

Project name

ForCaseStudy

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.

Public build access - *optional*

Public build access allows you to make the build results, including logs and artifacts, for this project available for the general public.

☐ Enable public build access

Additional configuration

Description, Build badge, Concurrent build limit, tags

Source

Add source

Source 1 - Primary

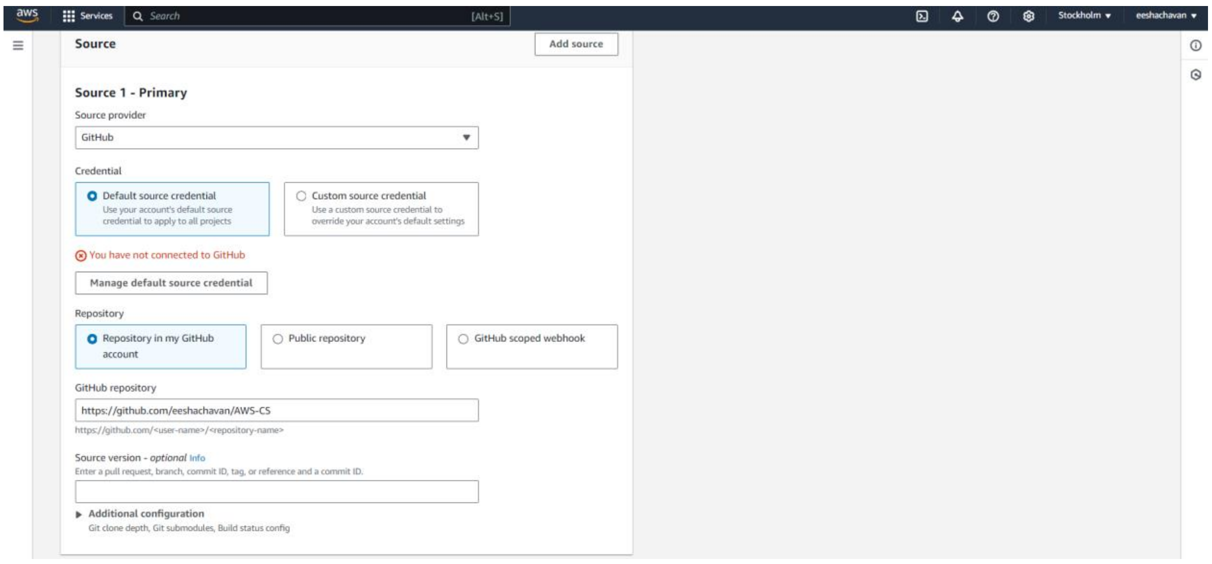
Source provider

GitHub

Credential

[✓ Default source credential] Custom source credential

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



Source

Add source

Source 1 - Primary

Source provider

GitHub

Credential

[✓ Default source credential
Use your account's default source credential to apply to all projects]

Custom source credential
Use a custom source credential to override your account's default settings

You have not connected to GitHub

Manage default source credential

Repository

[✓ Repository in my GitHub account] Public repository GitHub scoped webhook

GitHub repository

https://github.com/eeshachavan/AWS-CS

https://github.com/<user-name>/<repository-name>

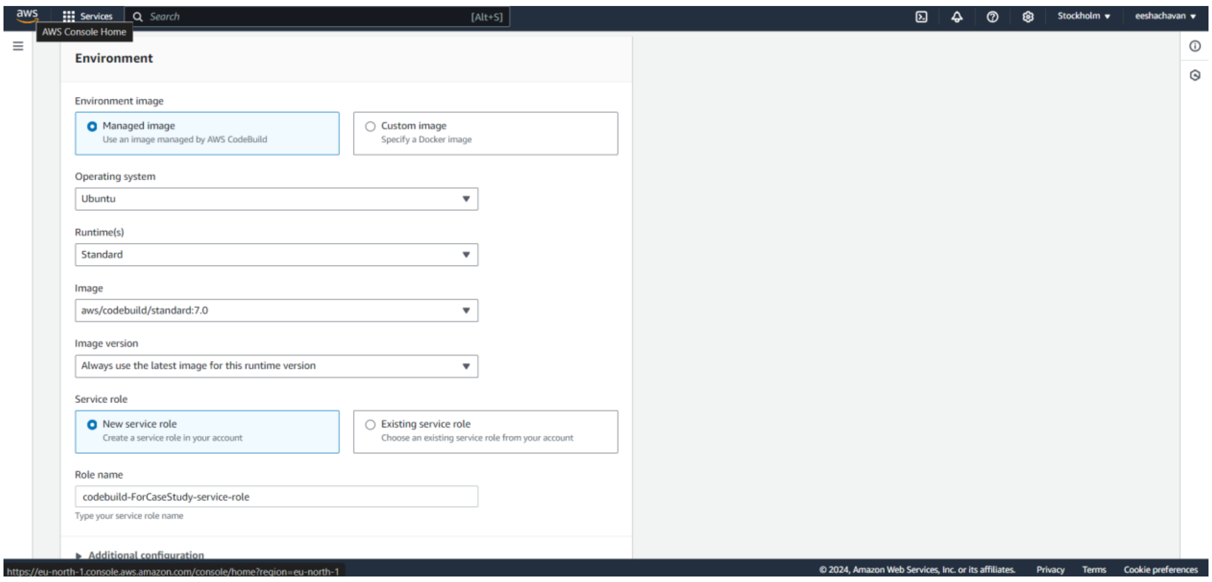
Source version - *optional* Info

Enter a pull request, branch, commit ID, tag, or reference and a commit ID.

Additional configuration

Git clone depth, Git submodules, Build status config

- For the **Environment**, select a managed image (e.g., Ubuntu with standard runtimes).


Environment

Environment image

[✓ Managed image — Use an image managed by AWS CodeBuild] Custom image — Specify a Docker image

Operating system: Ubuntu

Runtime(s): Standard

Image: aws/codebuild/standard:7.0

Image version: Always use the latest image for this runtime version

Service role

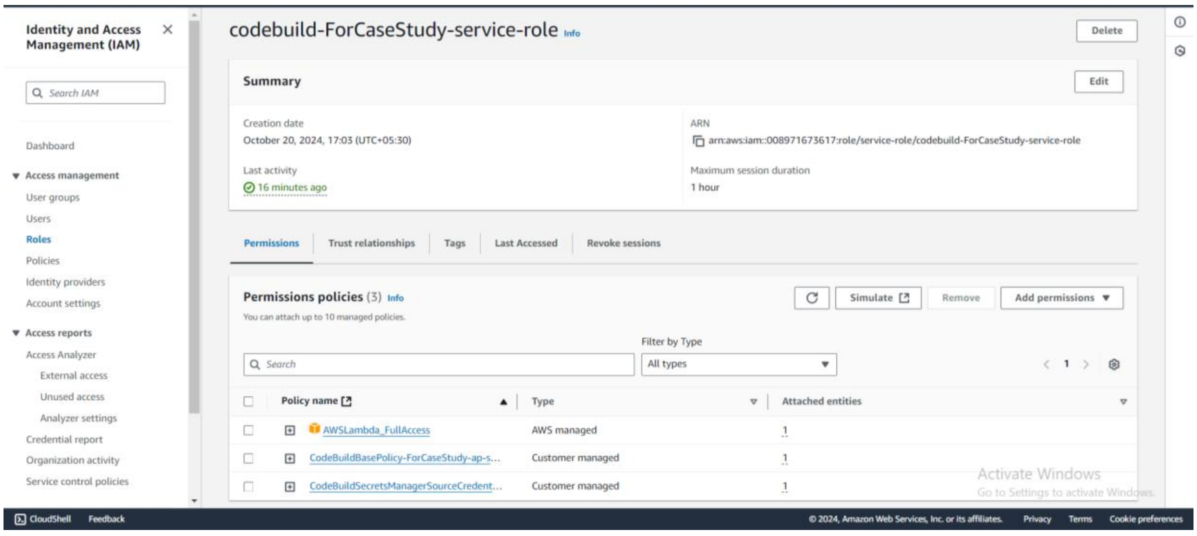
[✓ New service role — Create a service role in your account] Existing service role — Choose an existing service role from your account

Role name: codebuild-ForCaseStudy-service-role

Type your service role name

▸ Additional configuration

- Ensure that the environment has the correct permissions to update the Lambda function (using a role with AWSLambdaFullAccess or similar).


codebuild-ForCaseStudy-service-role

Summary

Creation date: October 20, 2024, 17:03 (UTC+05:30)

ARN: arn:aws:iam::008971673617:role/service-role/codebuild-ForCaseStudy-service-role

Last activity: 16 minutes ago

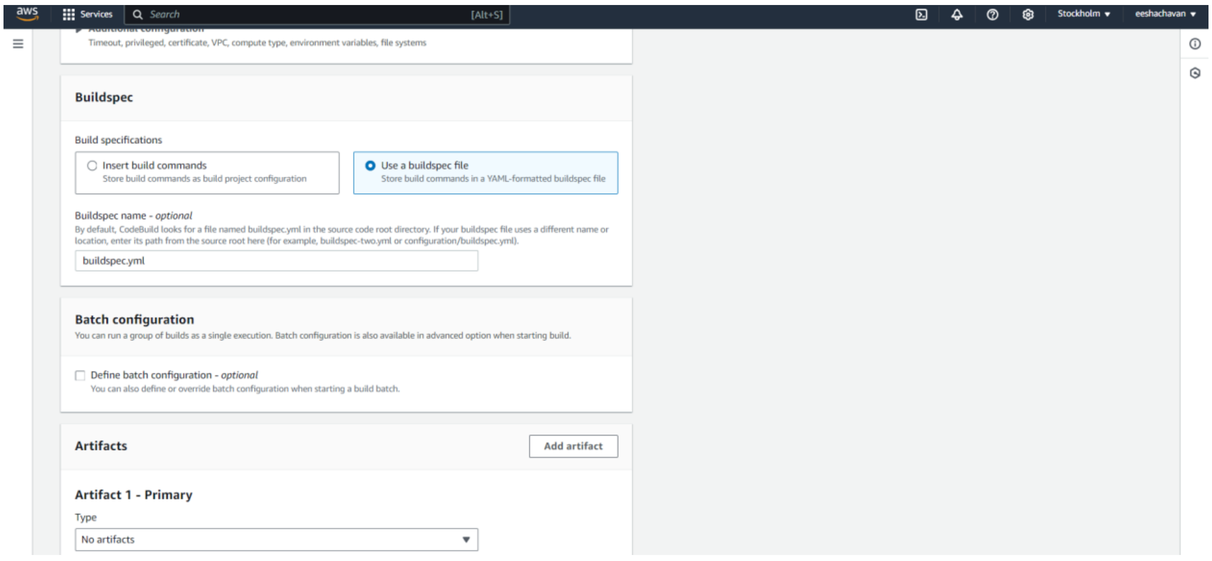
Maximum session duration: 1 hour

Permissions policies (3)

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AWSLambda_FullAccess	AWS managed	1
CodeBuildBasePolicy-ForCaseStudy-ap-s...	Customer managed	1
CodeBuildSecretsManagerSourceCredent...	Customer managed	1

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


Timeout, privileged, certificate, VPC, compute type, environment variables, file systems

Buildspec

Build specifications

Insert build commands [✓ Use a buildspec file]

Insert build commands: Store build commands as build project configuration

Use a buildspec file: Store build commands in a YAML-formatted buildspec file

Buildspec name - *optional*

By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

buildspec.yml

Batch configuration

You can run a group of builds as a single execution. Batch configuration is also available in advanced option when starting build.

☐ Define batch configuration - *optional*

You can also define or override batch configuration when starting a build batch.

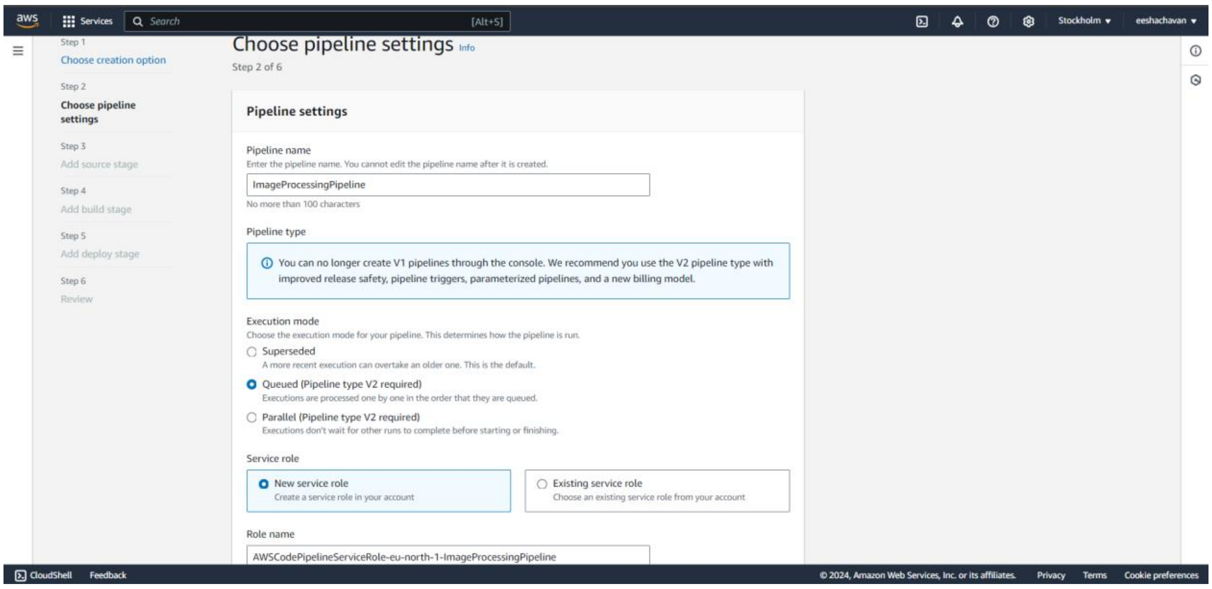
Artifacts

Artifact 1 - Primary

Type: No artifacts

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.


Choose pipeline settings

Step 2 of 6

Pipeline settings

Pipeline name

Enter the pipeline name. You cannot edit the pipeline name after it is created.

ImageProcessingPipeline

No more than 100 characters

Pipeline type

You can no longer create V1 pipelines through the console. We recommend you use the V2 pipeline type with improved release safety, pipeline triggers, parameterized pipelines, and a new billing model.

Execution mode

Choose the execution mode for your pipeline. This determines how the pipeline is run.

Superseded [✓ Queued (Pipeline type V2 required)] Parallel (Pipeline type V2 required)

Superseded: A more recent execution can overtake an older one. This is the default.

Queued (Pipeline type V2 required): Executions are processed one by one in the order that they are queued.

Parallel (Pipeline type V2 required): Executions don't wait for other runs to complete before starting or finishing.

Service role

[✓ New service role] Existing service role

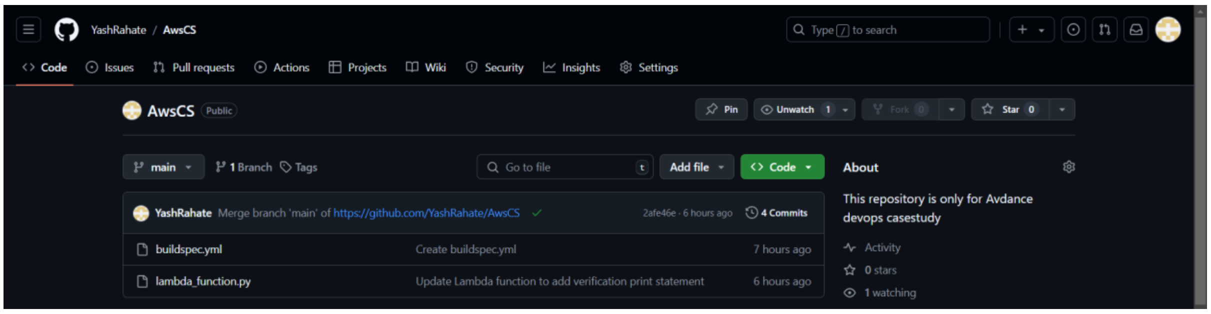
New service role: Create a service role in your account

Existing service role: Choose an existing service role from your account

Role name

AWSCodePipelineServiceRole-eu-north-1-ImageProcessingPipeline

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


YashRahate / AwsCS

AwsCS Public

main · 1 Branch · Tags

YashRahate Merge branch 'main' of https://github.com/YashRahate/AwsCS ✓ 2afe46e · 6 hours ago · 4 Commits

buildspec.yml	Create buildspec.yml	7 hours ago
lambda_function.py	Update Lambda function to add verification print statement	6 hours ago

About

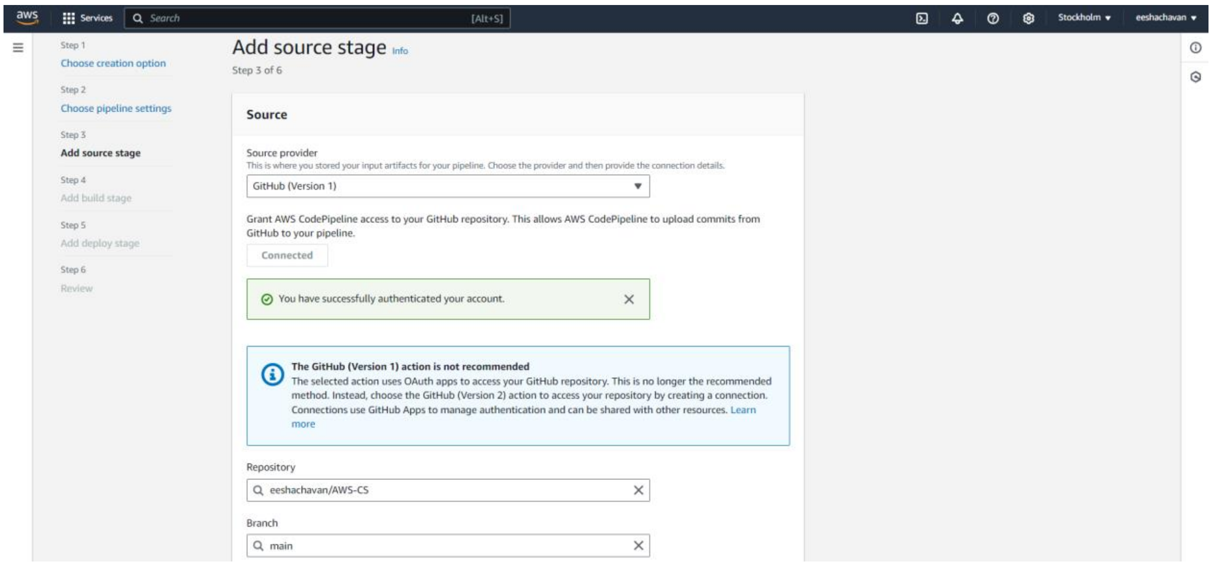
This repository is only for Avdance devops casestudy

Activity

0 stars

1 watching

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


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 source stage Info

Step 3 of 6

Source

Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (Version 1)

Grant AWS CodePipeline access to your GitHub repository. This allows AWS CodePipeline to upload commits from GitHub to your pipeline.

Connected

You have successfully authenticated your account.

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

Repository

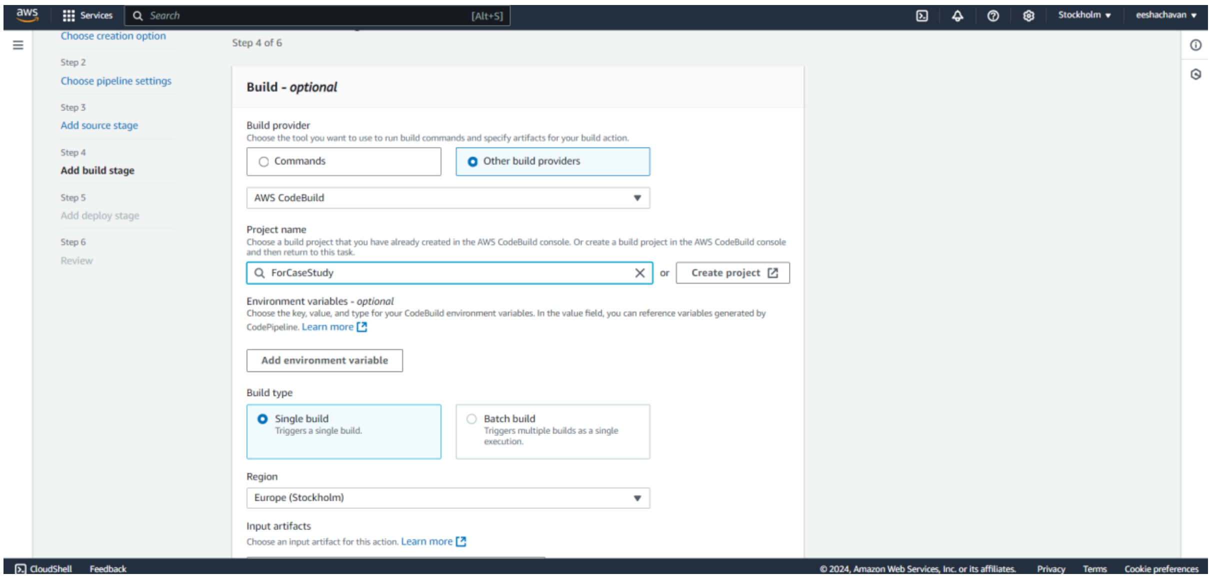
eeshachavan/AWS-CS

Branch

main

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.



Build - *optional*

Build provider
Choose the tool you want to use to run build commands and specify artifacts for your build action.

Commands [✓ Other build providers]

AWS CodeBuild

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

ForCaseStudy

Environment variables - *optional*
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more

Build type

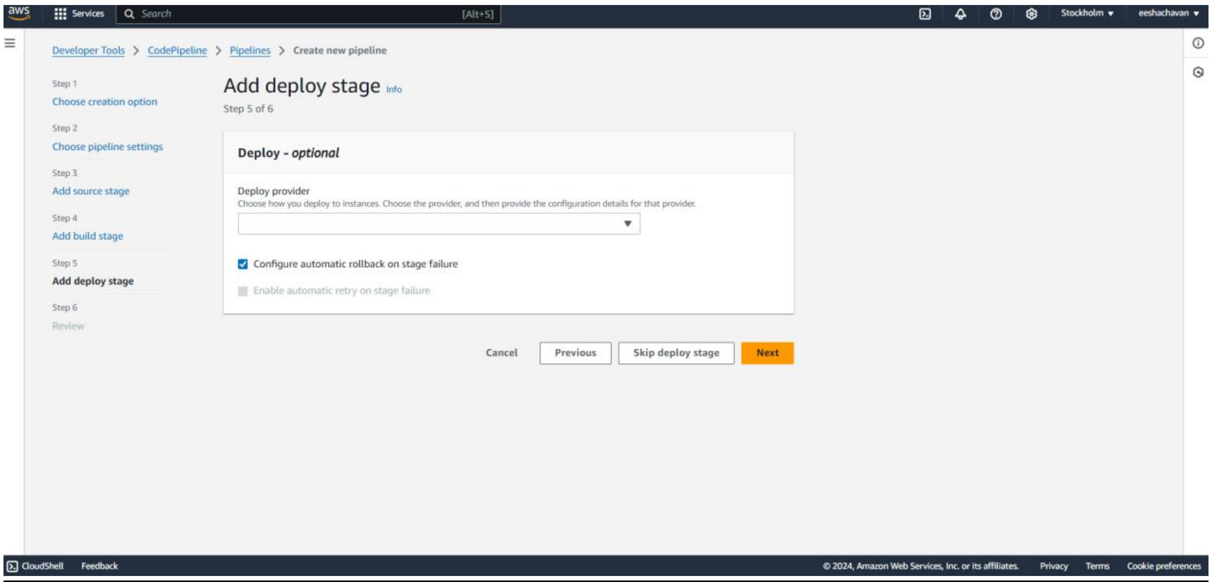
[✓ Single build] Triggers a single build.

Batch build Triggers multiple builds as a single execution.

Region
Europe (Stockholm)

Input artifacts
Choose an input artifact for this action. Learn more

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



Add deploy stage Info

Step 5 of 6

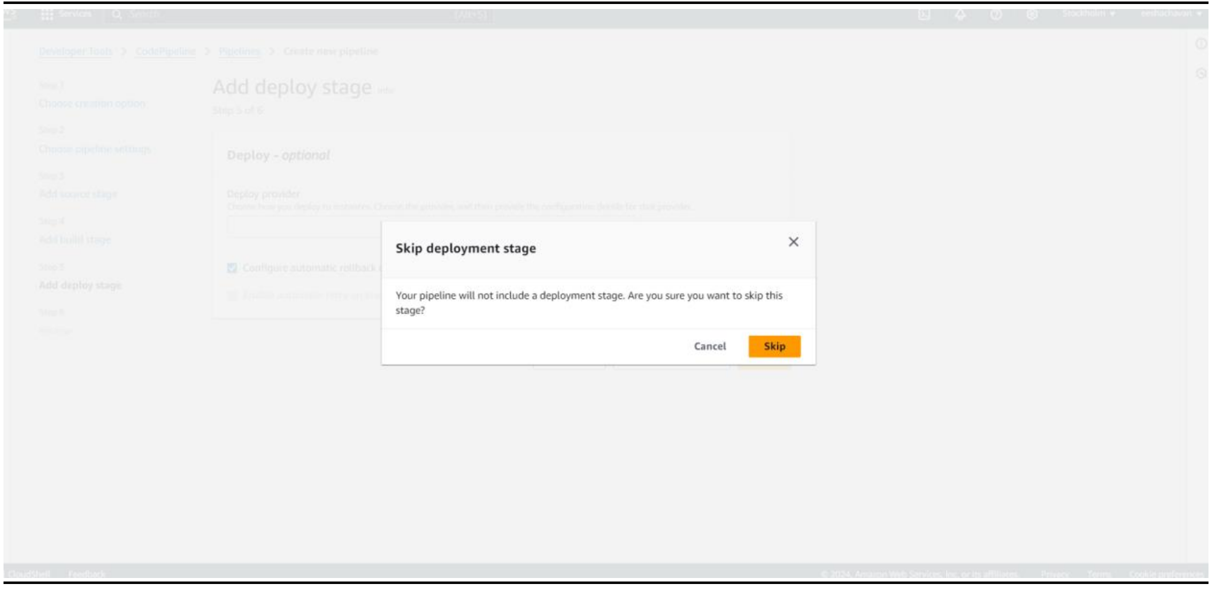
Deploy - *optional*

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

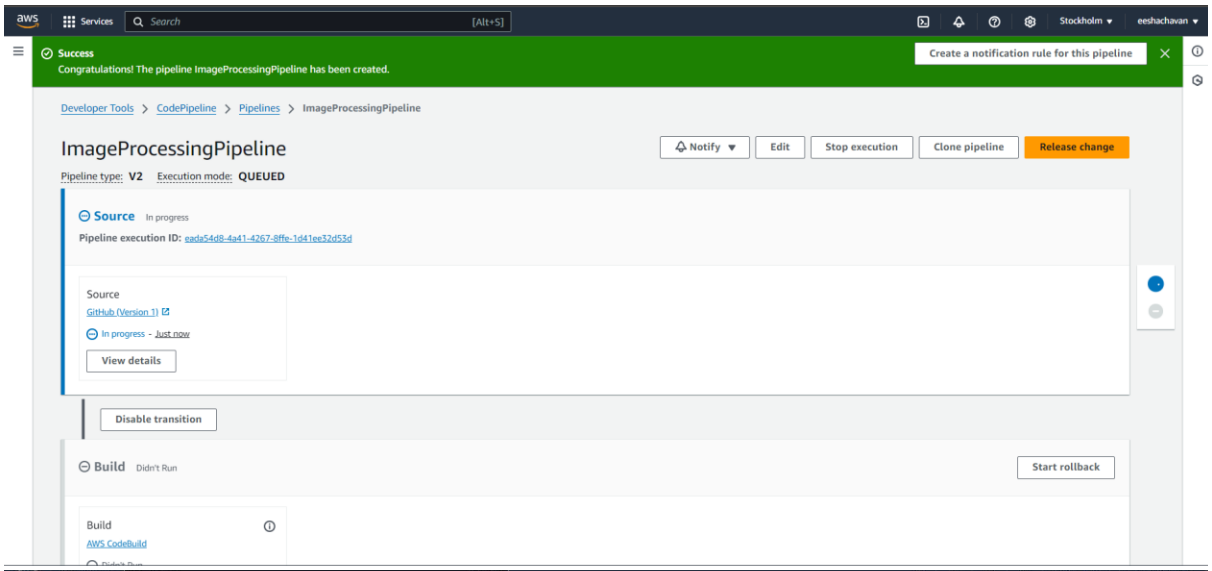
☐ Enable automatic retry on stage failure



Skip deployment stage

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

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



Success

Congratulations! The pipeline ImageProcessingPipeline has been created.

Create a notification rule for this pipeline
Developer Tools > CodePipeline > Pipelines > ImageProcessingPipeline

ImageProcessingPipeline

Pipeline type: V2 Execution mode: QUEUED

Source In progress

Pipeline execution ID: eada54d8-4a41-4267-8ffe-1d41ee32d53d

Source

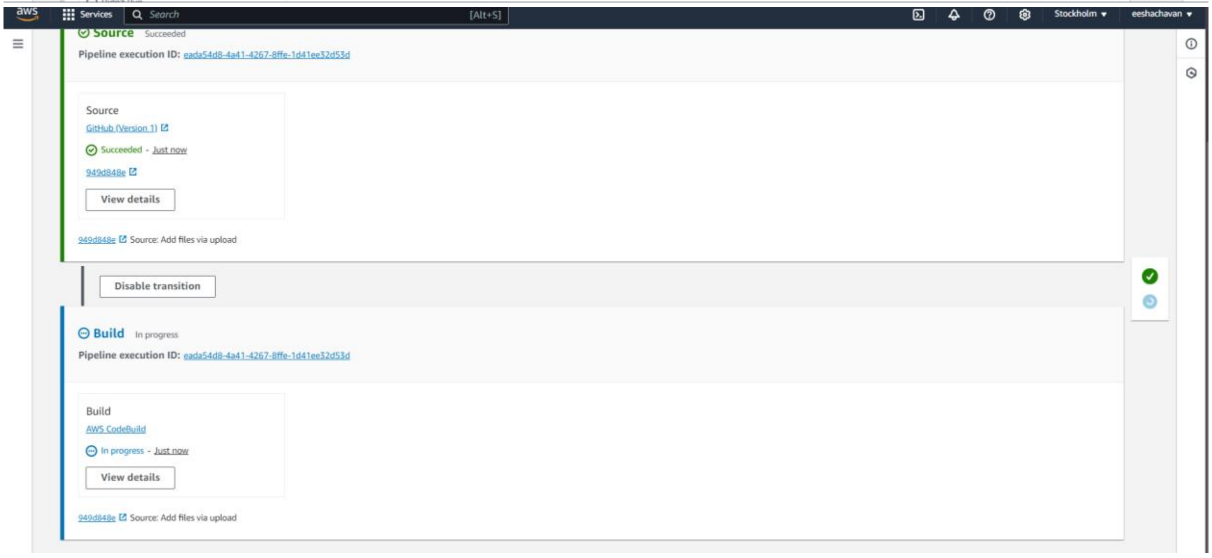
GitHub (Version 1)

In progress - Just now

Build Didn't Run

Build

AWS CodeBuild



Source Succeeded

Pipeline execution ID: eada54d8-4a41-4267-8ffe-1d41ee32d53d

Source

GitHub (Version 1)

Succeeded - Just now

949d848e

949d848e Source: Add files via upload

Build In progress

Pipeline execution ID: eada54d8-4a41-4267-8ffe-1d41ee32d53d

Build

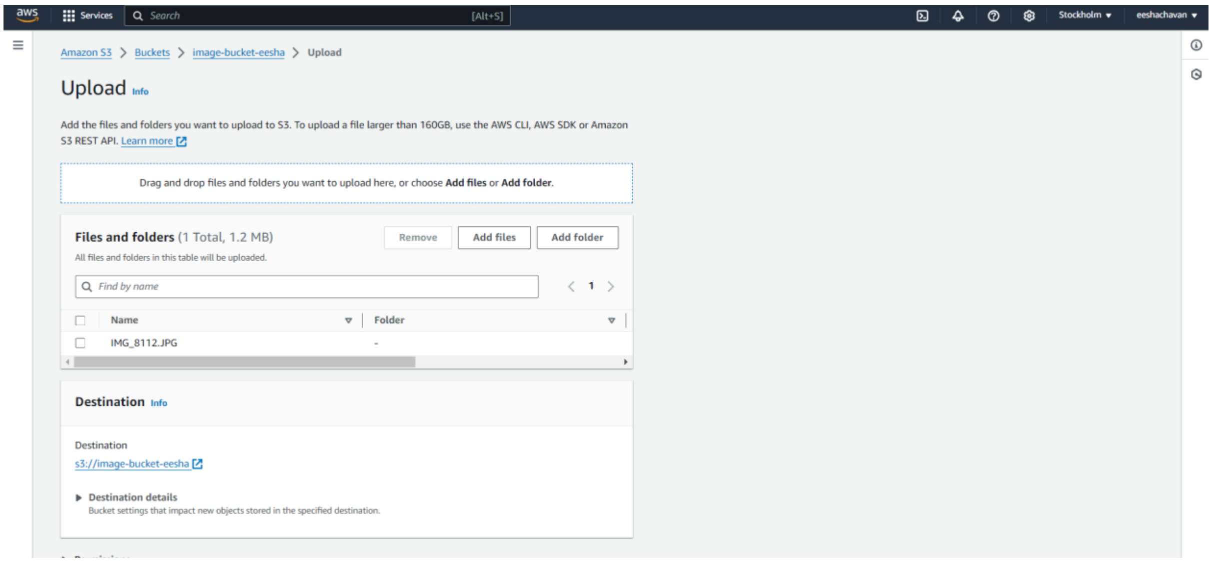
AWS CodeBuild

In progress - Just now

949d848e Source: Add files via upload

Step 6: Test the Serverless Workflow

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



Amazon S3 > Buckets > image-bucket-eesha > Upload

Upload Info

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

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 1.2 MB)

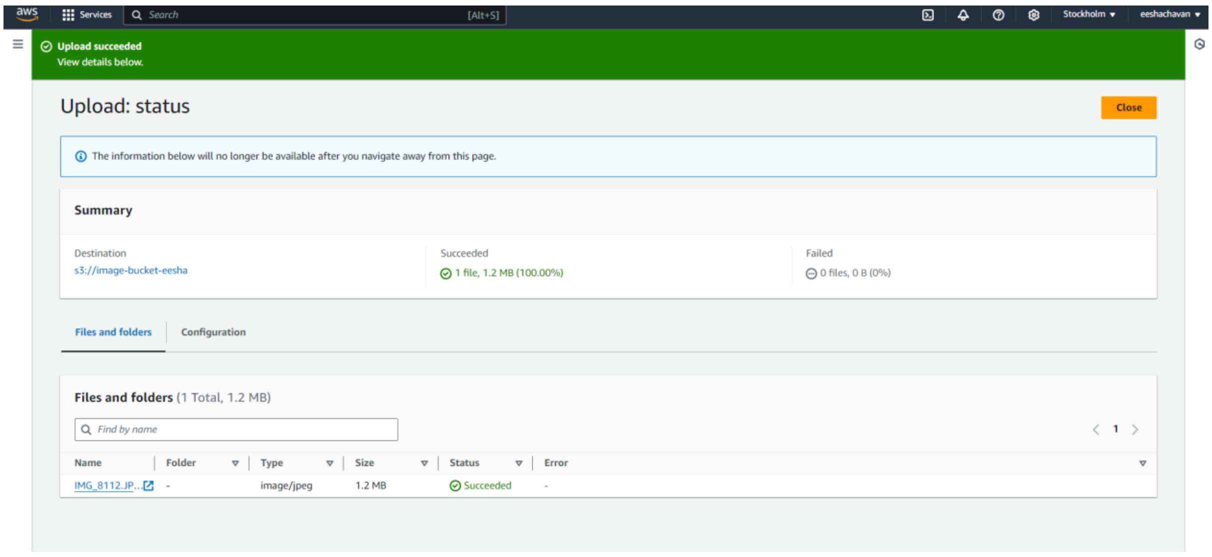
All files and folders in this table will be uploaded.

Name	Folder
IMG_8112.JPG	-

Destination Info

Destination
s3://image-bucket-eesha

Destination details
Bucket settings that impact new objects stored in the specified destination.



Upload succeeded
View details below.

Upload: status

The information below will no longer be available after you navigate away from this page.

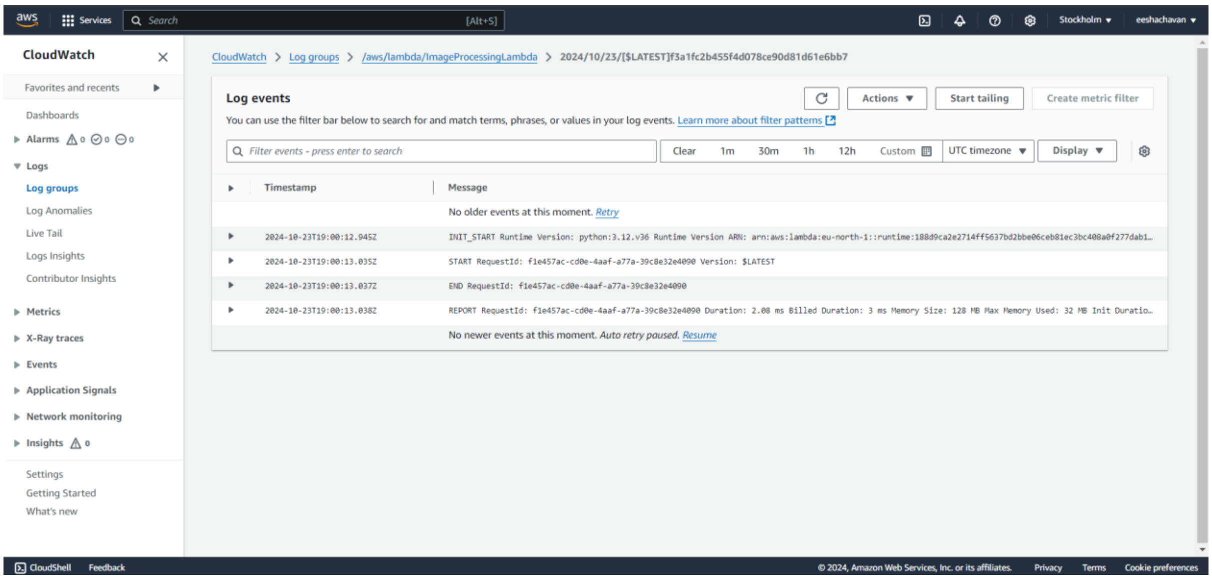
Summary

Destination	Succeeded	Failed
s3://image-bucket-eesha	1 file, 1.2 MB (100.00%)	0 files, 0 B (0%)

Files and folders (1 Total, 1.2 MB)

Name	Folder	Type	Size	Status	Error
IMG_8112.JP...	-	image/jpeg	1.2 MB	Succeeded	-

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).



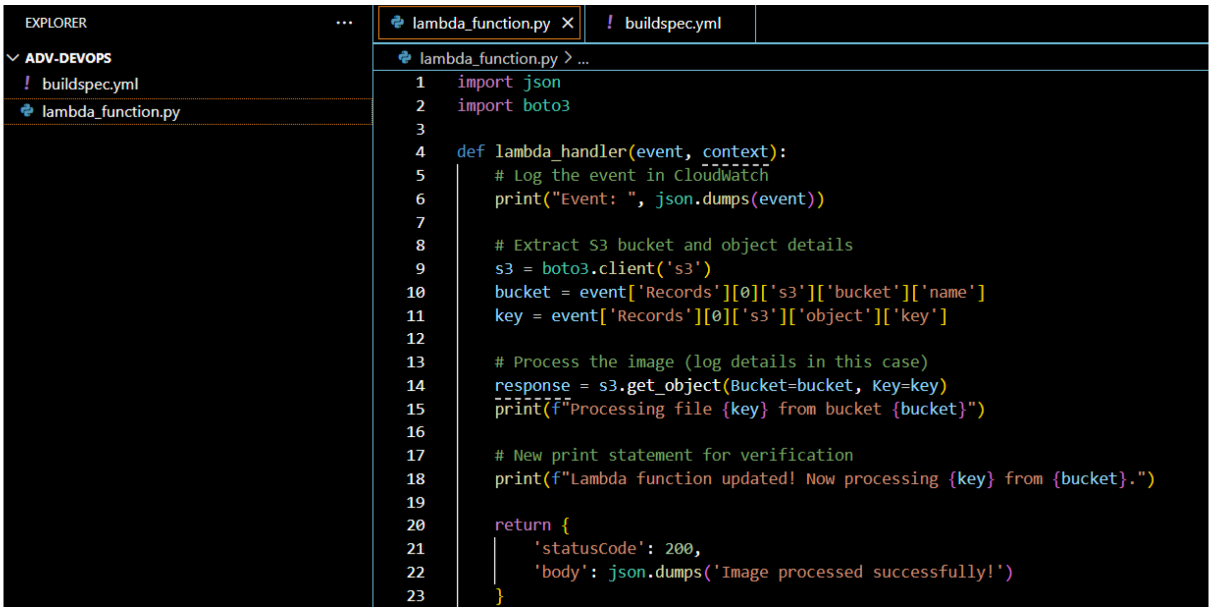
Log events

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
                           No older events at this moment. Retry
2024-10-23T19:00:12.945Z   INIT_START Runtime Version: python:3.12.v36 Runtime Version ARN: arn:aws:lambda:eu-north-1::runtime:188d9ca2e2714ff5637bd2bbe06ceb81ec3bc408a0f277dab1…
2024-10-23T19:00:13.035Z   START RequestId: f1e457ac-cd0e-4aaf-a77a-39c8e32e4090 Version: $LATEST
2024-10-23T19:00:13.037Z   END RequestId: f1e457ac-cd0e-4aaf-a77a-39c8e32e4090
2024-10-23T19:00:13.038Z   REPORT RequestId: f1e457ac-cd0e-4aaf-a77a-39c8e32e4090 Duration: 2.08 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 32 MB Init Duratio…
                           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).



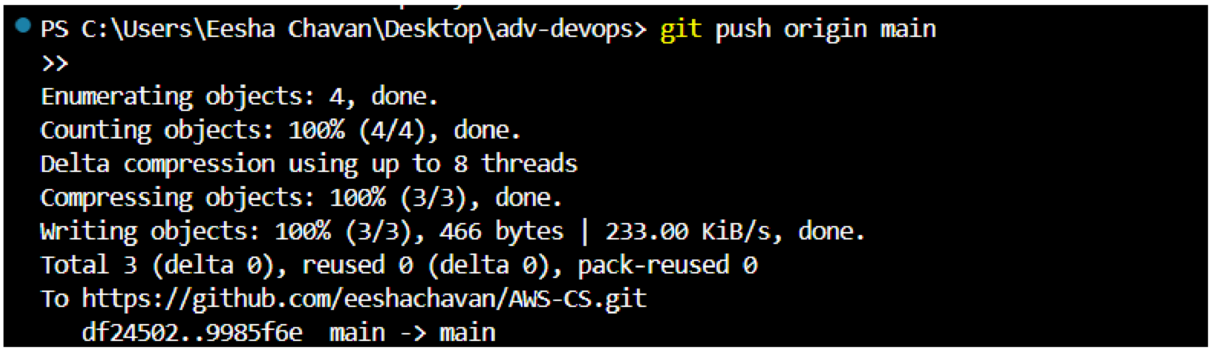
```python
import json
import boto3

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!')
    }
```

New Code:

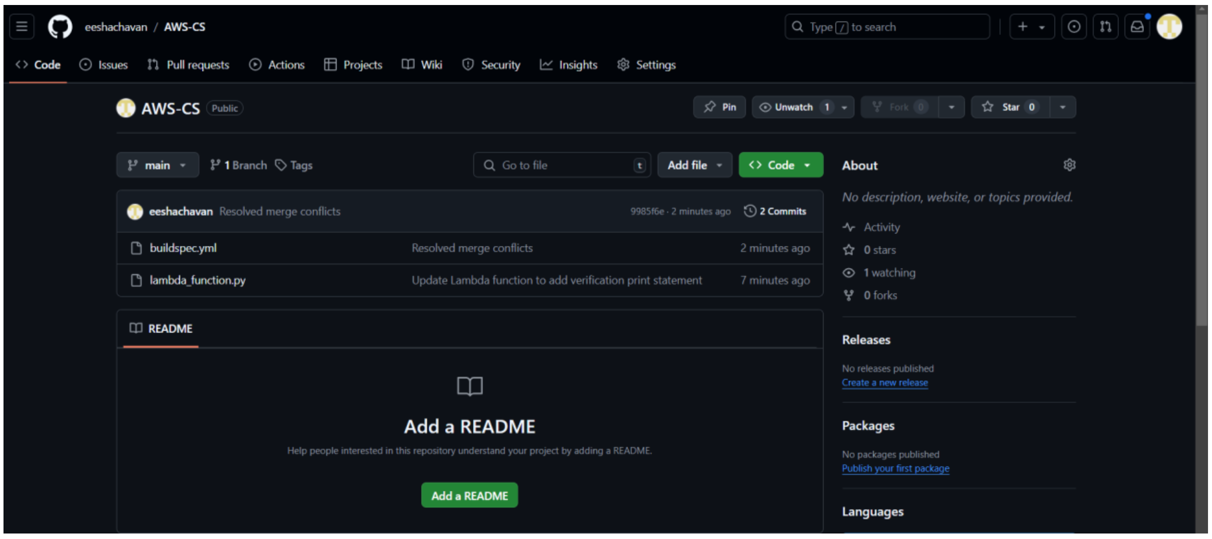
```
import json
import boto3

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\Eesha Chavan\Desktop\adv-devops> git push origin main
>>
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 466 bytes | 233.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/eeshachavan/AWS-CS.git
   df24502..9985f6e  main -> main
```



eeshachavan / AWS-CS

AWS-CS Public

main · 1 Branch · Tags

eeshachavan Resolved merge conflicts · 9985f6e · 2 minutes ago · 2 Commits

buildspec.yml	Resolved merge conflicts	2 minutes ago
lambda_function.py	Update Lambda function to add verification print statement	7 minutes ago

README

Add a README

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

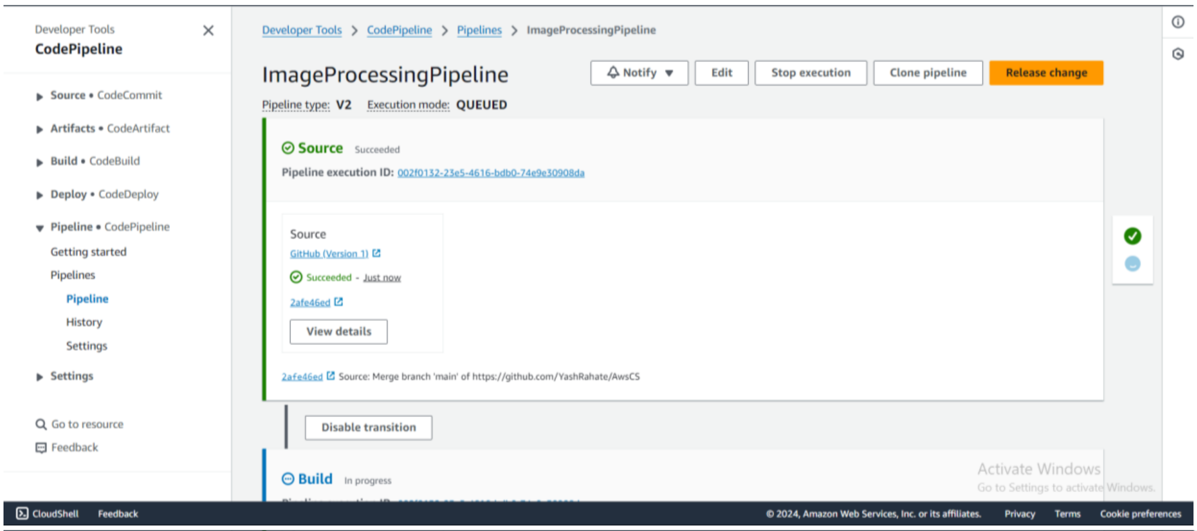
Activity · 0 stars · 1 watching · 0 forks

Releases: No releases published · Create a new release

Packages: No packages published · Publish your first package

Languages

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



Developer Tools > CodePipeline > Pipelines > ImageProcessingPipeline

ImageProcessingPipeline

Pipeline type: V2 Execution mode: QUEUED

Source Succeeded

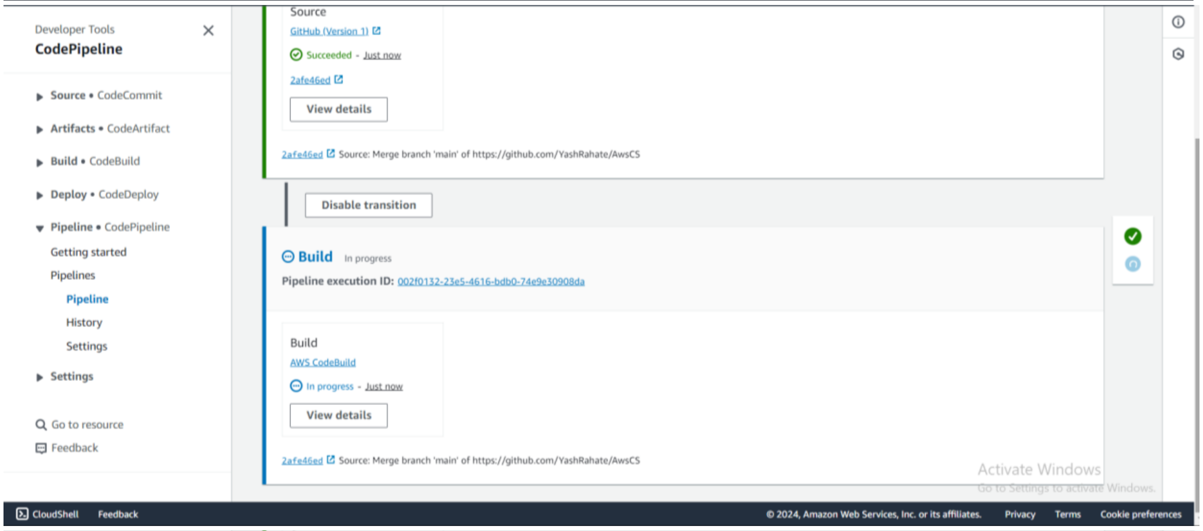
Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da

Source
GitHub (Version 1)
Succeeded - Just now
2afe46ed

2afe46ed Source: Merge branch 'main' of https://github.com/YashRahate/AwsCS

Build In progress

© 2024, Amazon Web Services, Inc. or its affiliates.



Source
GitHub (Version 1)
Succeeded - Just now
2afe46ed

2afe46ed Source: Merge branch 'main' of https://github.com/YashRahate/AwsCS

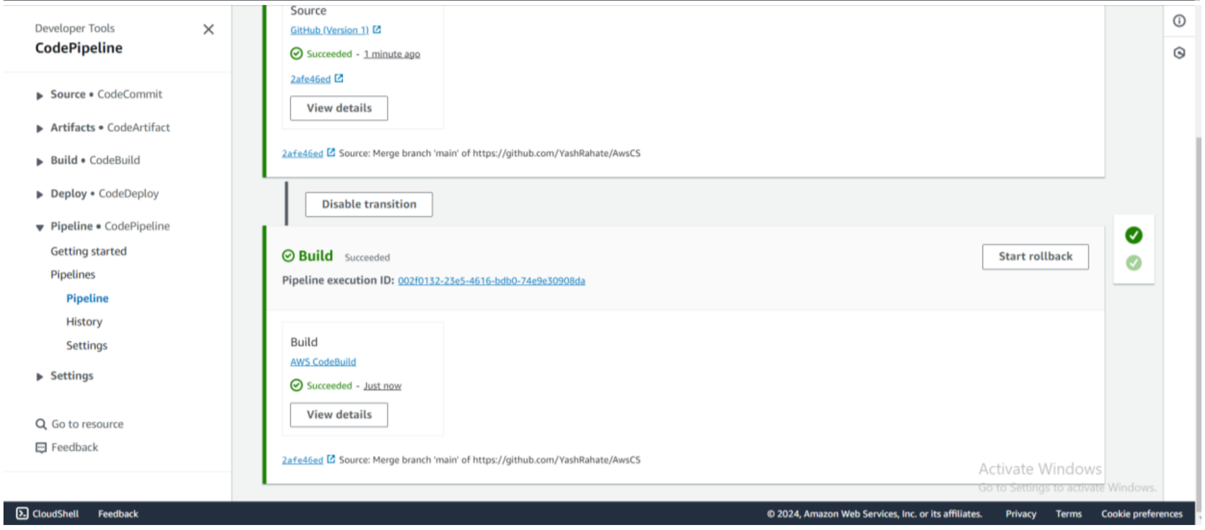
Build In progress

Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da

Build
AWS CodeBuild
In progress - Just now

2afe46ed Source: Merge branch 'main' of https://github.com/YashRahate/AwsCS

© 2024, Amazon Web Services, Inc. or its affiliates.



Source
GitHub (Version 1)
Succeeded - 1 minute ago
2afe46ed

2afe46ed Source: Merge branch 'main' of https://github.com/YashRahate/AwsCS

Build Succeeded

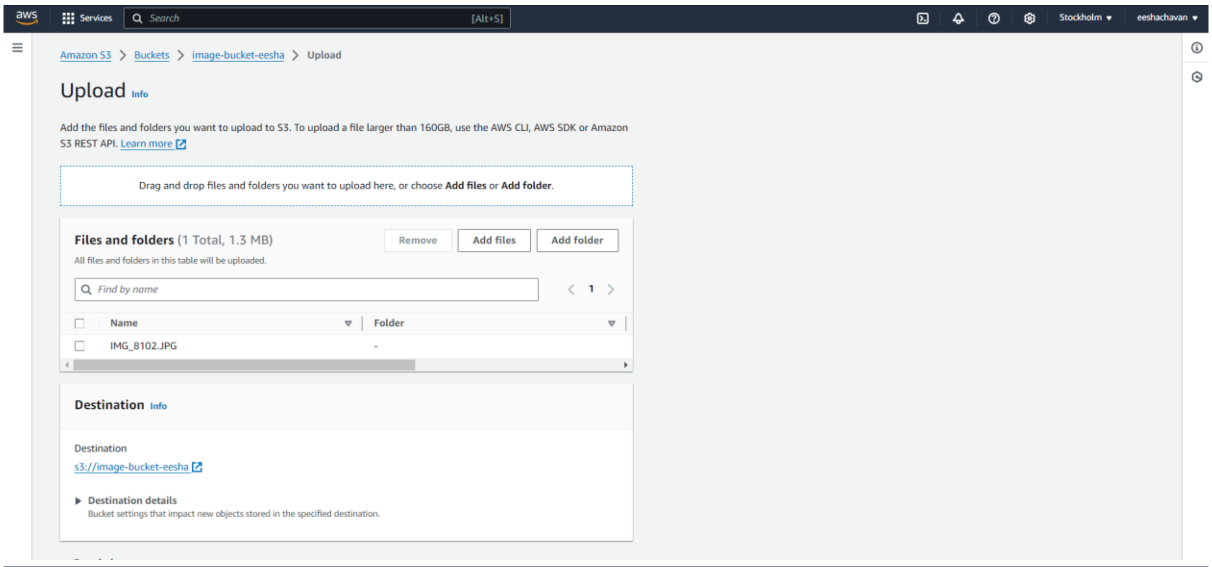
Pipeline execution ID: 002f0132-23e5-4616-bdb0-74e9e30908da

Build
AWS CodeBuild
Succeeded - Just now

2afe46ed Source: Merge branch 'main' of https://github.com/YashRahate/AwsCS

© 2024, Amazon Web Services, Inc. or its affiliates.

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


Amazon S3 > Buckets > image-bucket-eesha > Upload

Upload Info

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

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 1.3 MB)

All files and folders in this table will be uploaded.

Name	Folder
IMG_8102.JPG	-

Destination Info

Destination
s3://image-bucket-eesha

Destination details
Bucket settings that impact new objects stored in the specified destination.

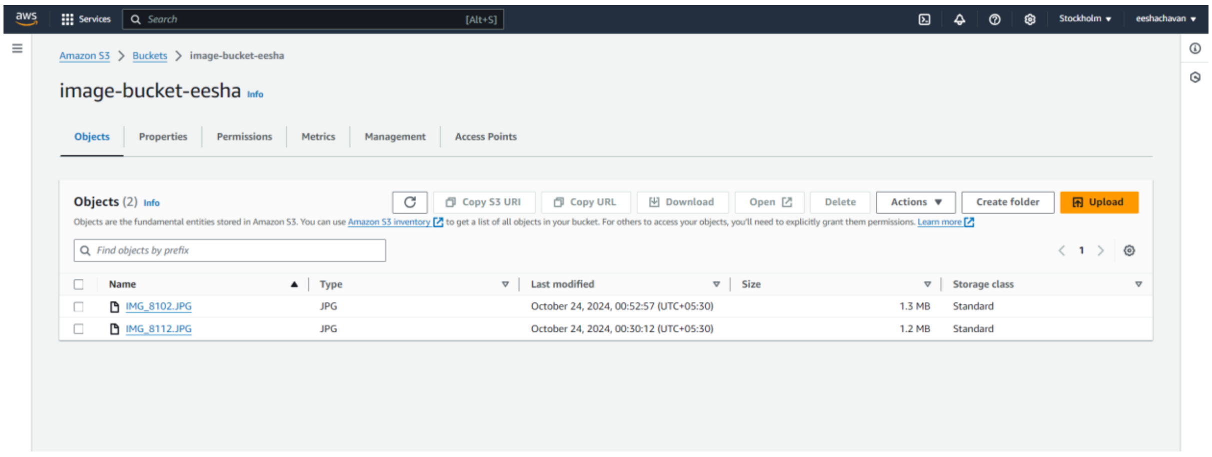
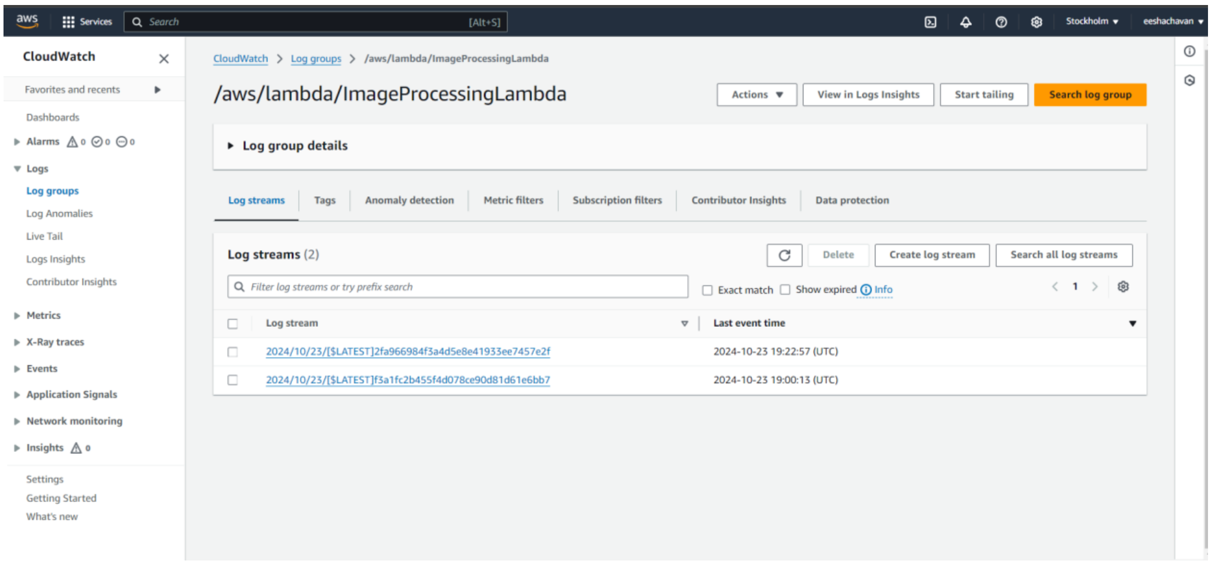

Amazon S3 > Buckets > image-bucket-eesha

image-bucket-eesha Info

Objects (2) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

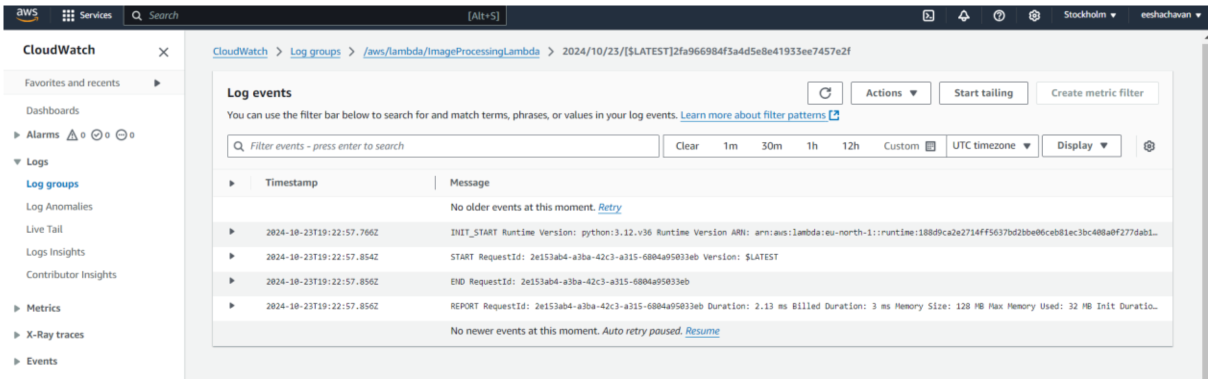
Name	Type	Last modified	Size	Storage class
IMG_8102.JPG	JPG	October 24, 2024, 00:52:57 (UTC+05:30)	1.3 MB	Standard
IMG_8112.JPG	JPG	October 24, 2024, 00:30:12 (UTC+05:30)	1.2 MB	Standard


CloudWatch > Log groups > /aws/lambda/ImageProcessingLambda

/aws/lambda/ImageProcessingLambda

Log streams (2)

Log stream	Last event time
2024/10/23/[$LATEST]2fa966984f3a4d5e8e41933ee7457e2f	2024-10-23 19:22:57 (UTC)
2024/10/23/[$LATEST]f3a1fc2b455f4d078ce90d81d61e6bb7	2024-10-23 19:00:13 (UTC)


CloudWatch > Log groups > /aws/lambda/ImageProcessingLambda > 2024/10/23/[$LATEST]2fa966984f3a4d5e8e41933ee7457e2f

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
                            No older events at this moment. Retry
2024-10-23T19:22:57.766Z    INIT_START Runtime Version: python:3.12.v36 Runtime Version ARN: arn:aws:lambda:eu-north-1::runtime:188d9ca2e2714ff5637bd2bbe06ceb81ec3bc408a0f277dab1…
2024-10-23T19:22:57.854Z    START RequestId: 2e153ab4-a3ba-42c3-a315-6804a95033eb Version: $LATEST
2024-10-23T19:22:57.856Z    END RequestId: 2e153ab4-a3ba-42c3-a315-6804a95033eb
2024-10-23T19:22:57.856Z    REPORT RequestId: 2e153ab4-a3ba-42c3-a315-6804a95033eb Duration: 2.13 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 32 MB Init Duratio…
                            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.