Case Study

Serverless Image Processing Workflow

Name: Rujuta Vinit Medhi

Class: D15A

Roll no.: 28

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

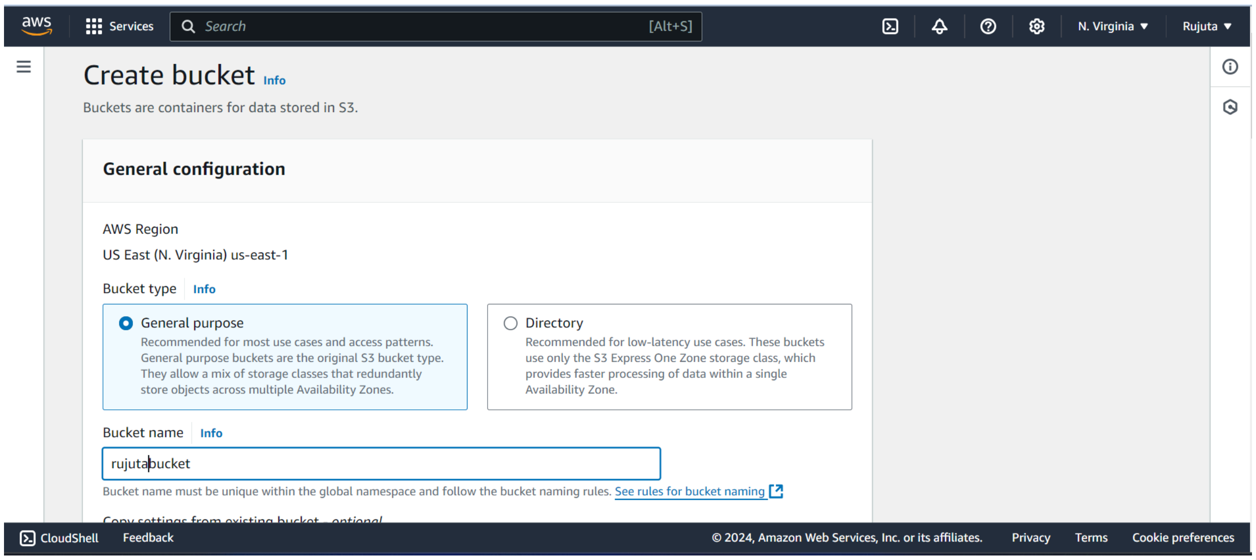
Theory:

- **AWS Lambda**: A serverless compute service that automatically runs code in response to events, managing the infrastructure needed to execute code without provisioning or managing servers. It supports multiple languages and scales automatically based on demand.
- **Amazon S3**: A highly scalable object storage service designed for storing and retrieving any amount of data from anywhere, with built-in features like data durability, security, and versioning for backups and archiving.
- **AWS CodePipeline**: A fully managed continuous integration and continuous delivery (CI/CD) service that automates the build, test, and deployment process of your code, enabling rapid software releases.

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
US East (N. Virginia) us-east-1

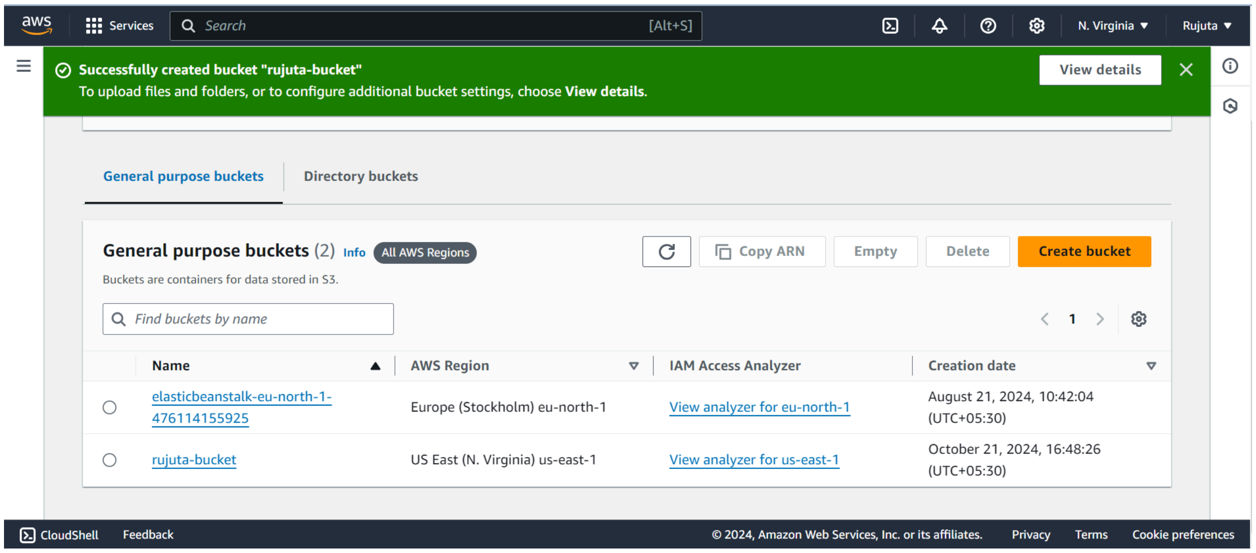
Bucket type Info

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
rujutabucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming



Successfully created bucket "rujuta-bucket"
To upload files and folders, or to configure additional bucket settings, choose **View details**.

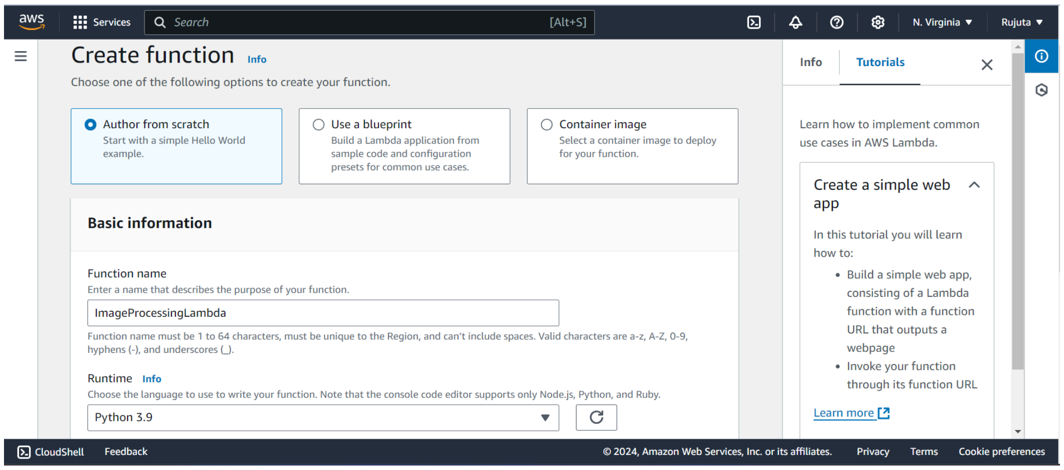
General purpose buckets (2) Info — All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
elasticbeanstalk-eu-north-1-476114155925	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	August 21, 2024, 10:42:04 (UTC+05:30)
rujuta-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	October 21, 2024, 16:48:26 (UTC+05:30)

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: Python 3.9

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

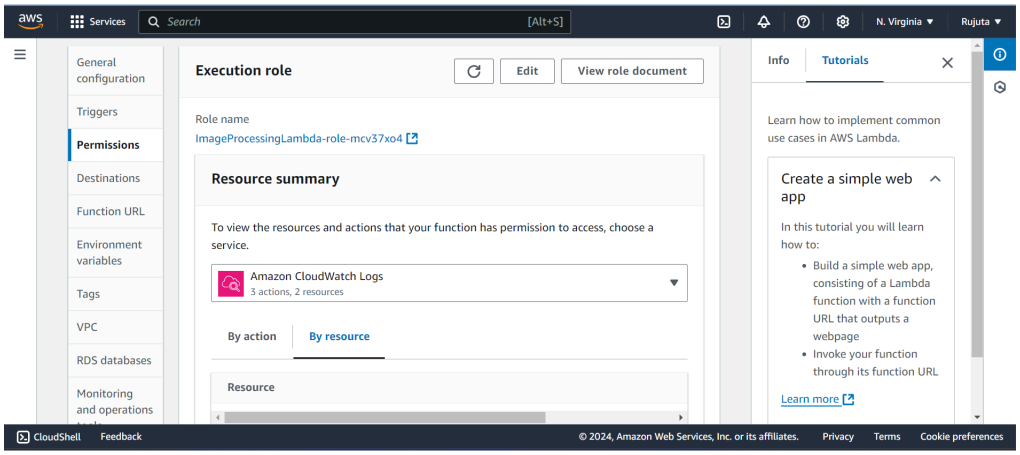
Learn how to implement common use cases in AWS Lambda.

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

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

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**.



Execution role

Role name: ImageProcessingLambda-role-mcv37xo4

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

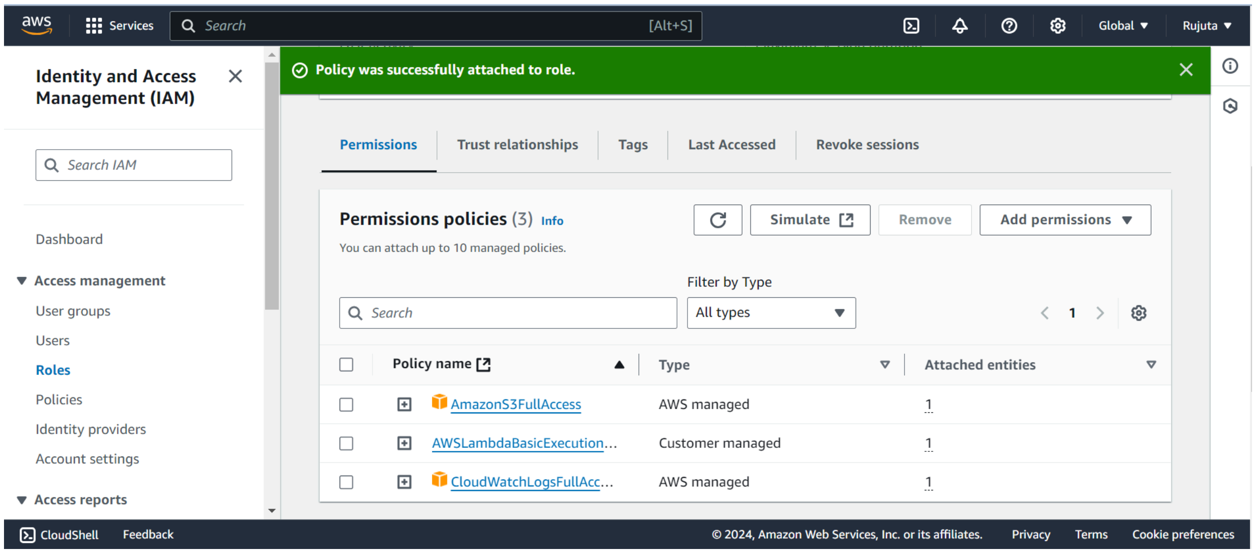
Resource

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

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



Policy was successfully attached to role.

Permissions policies (3) Info

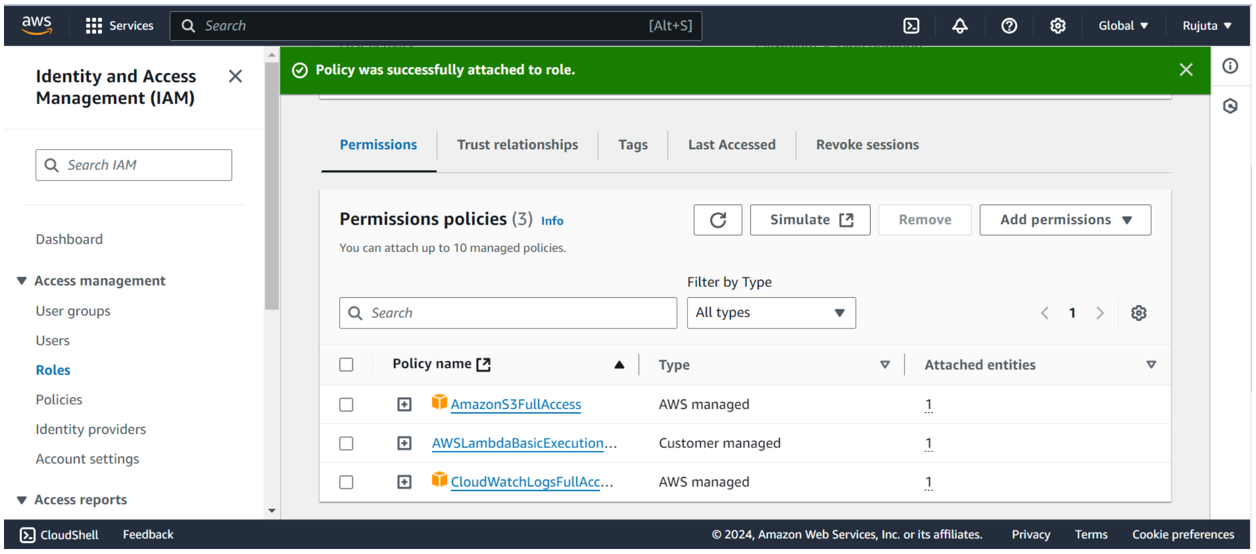
You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	1
AWSLambdaBasicExecution…	Customer managed	1
CloudWatchLogsFullAcc…	AWS managed	1

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

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**


Policy was successfully attached to role.

Permissions policies (3) Info

You can attach up to 10 managed policies.

Filter by Type: All types

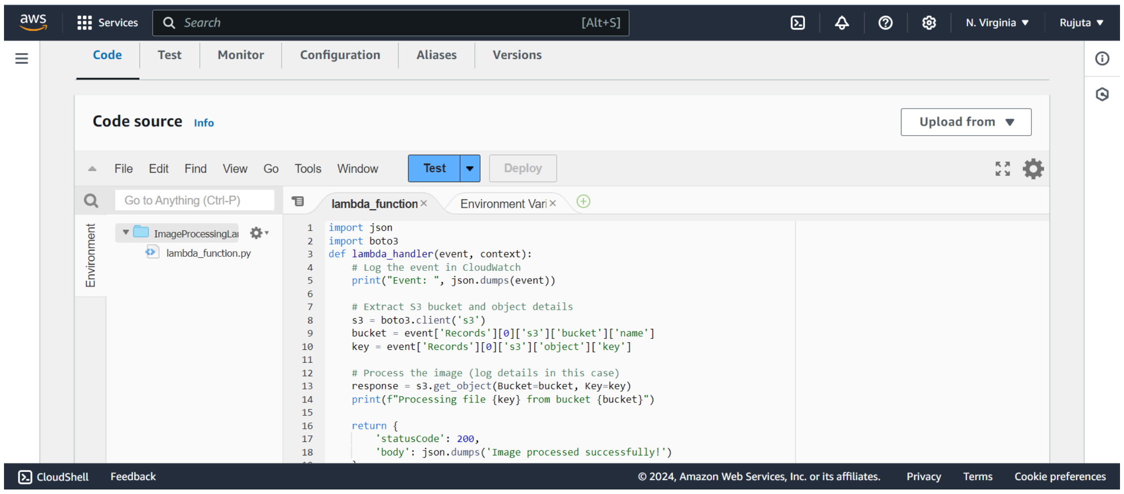
Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	1
AWSLambdaBasicExecution…	Customer managed	1
CloudWatchLogsFullAcc…	AWS managed	1

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

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!')

        }
```



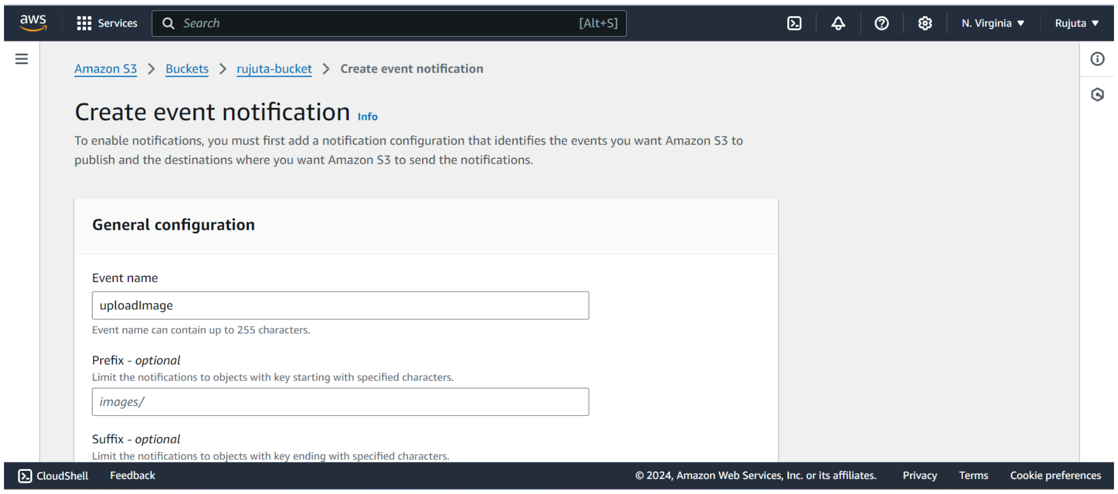
```
import json
import boto3
def lambda_handler(event, context):
    # Log the event in CloudWatch
    print("Event: ", json.dumps(event))

    # 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!')
```

- 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.



Amazon S3 > Buckets > rujuta-bucket > 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: uploadImage

Event name can contain up to 255 characters.

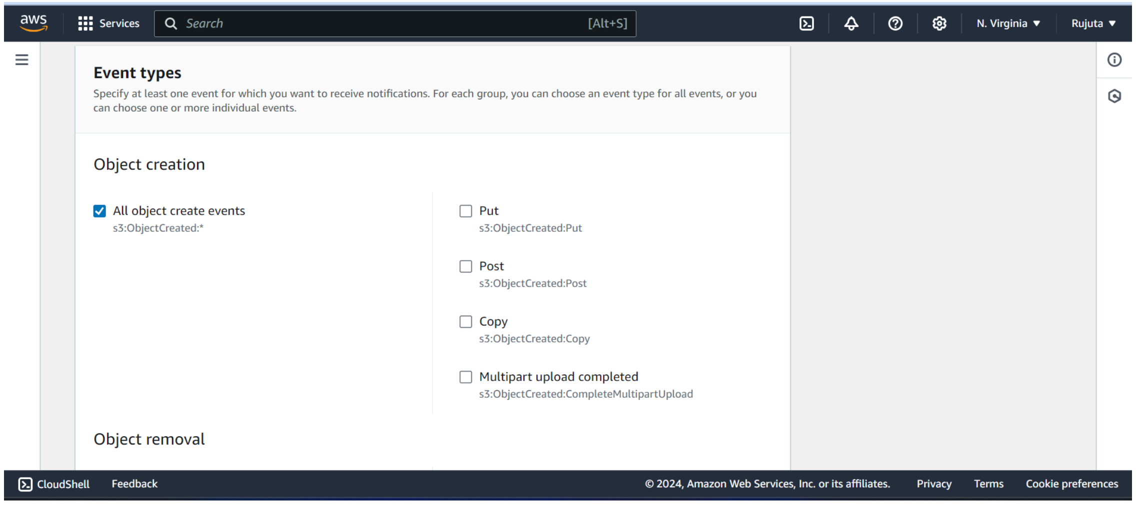
Prefix - *optional*

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.



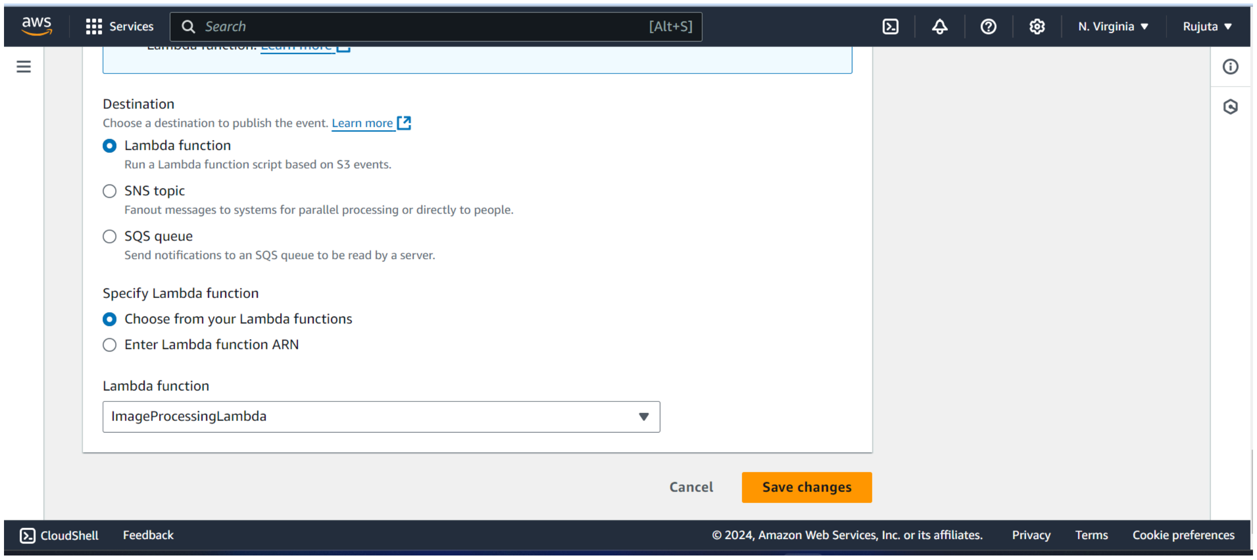
Event types

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


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

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

Click **Save Changes**.

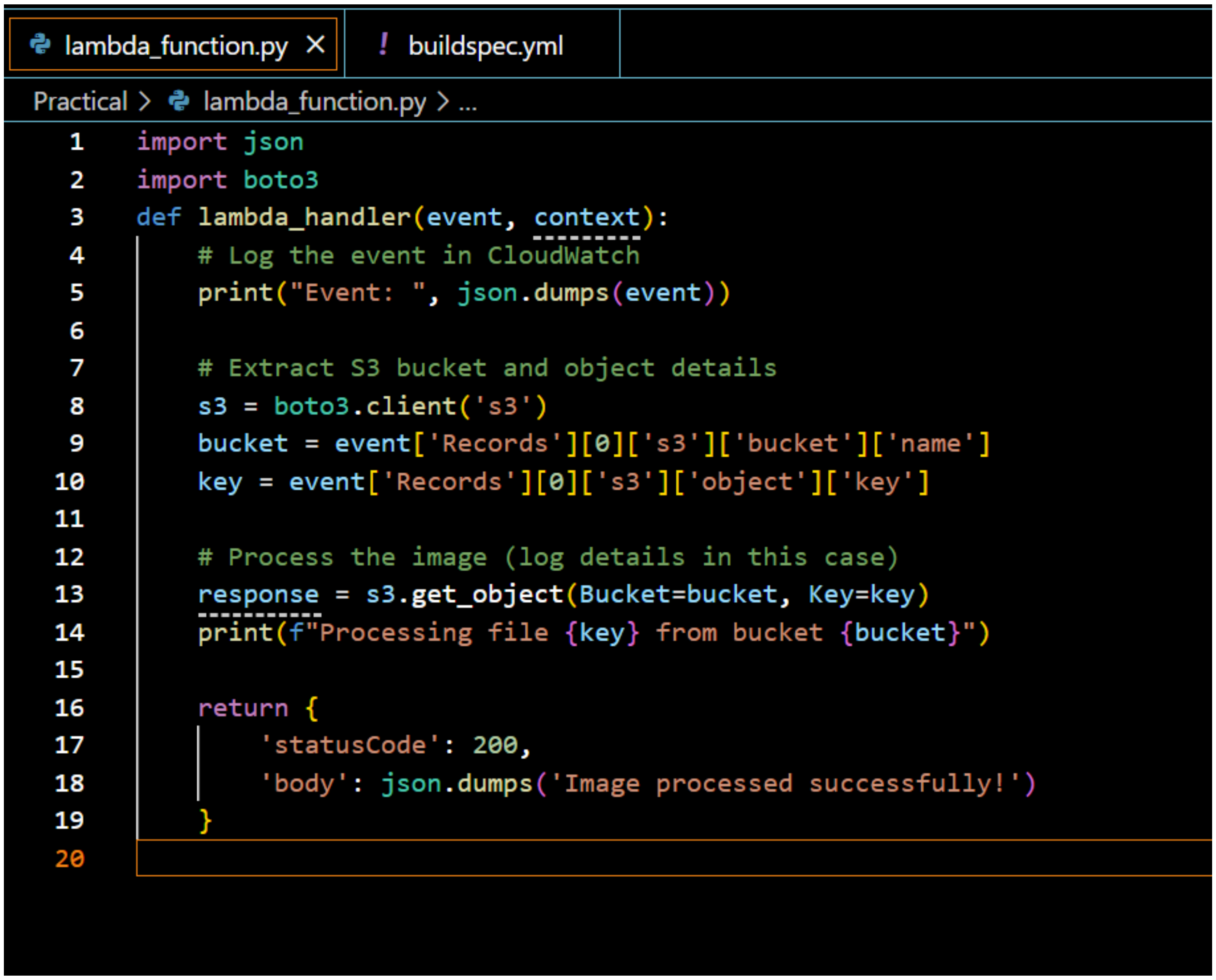
Step 4 :Step-by-Step Guide Using CodeBuild:

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:

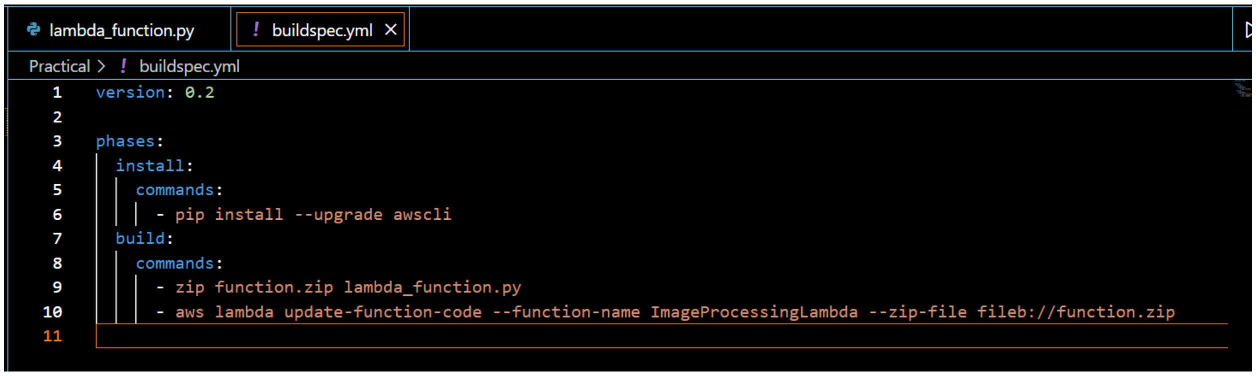
   CODE:

```
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



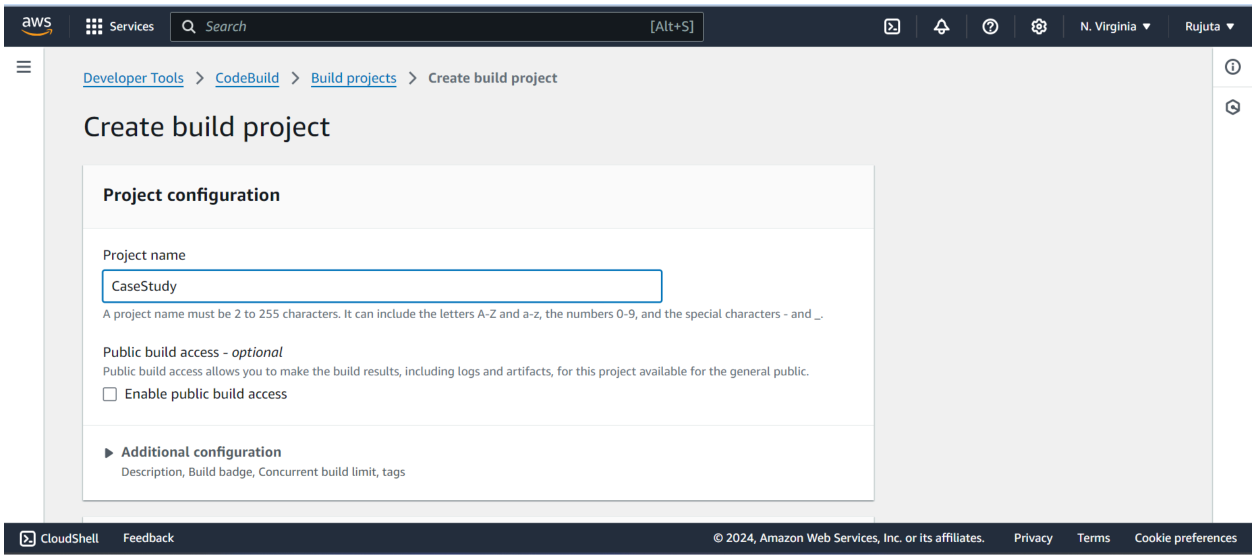
```python
import json
import boto3
def lambda_handler(event, context):
    # Log the event in CloudWatch
    print("Event: ", json.dumps(event))

    # 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!')
    }

```



```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

```

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: CaseStudy

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 _.

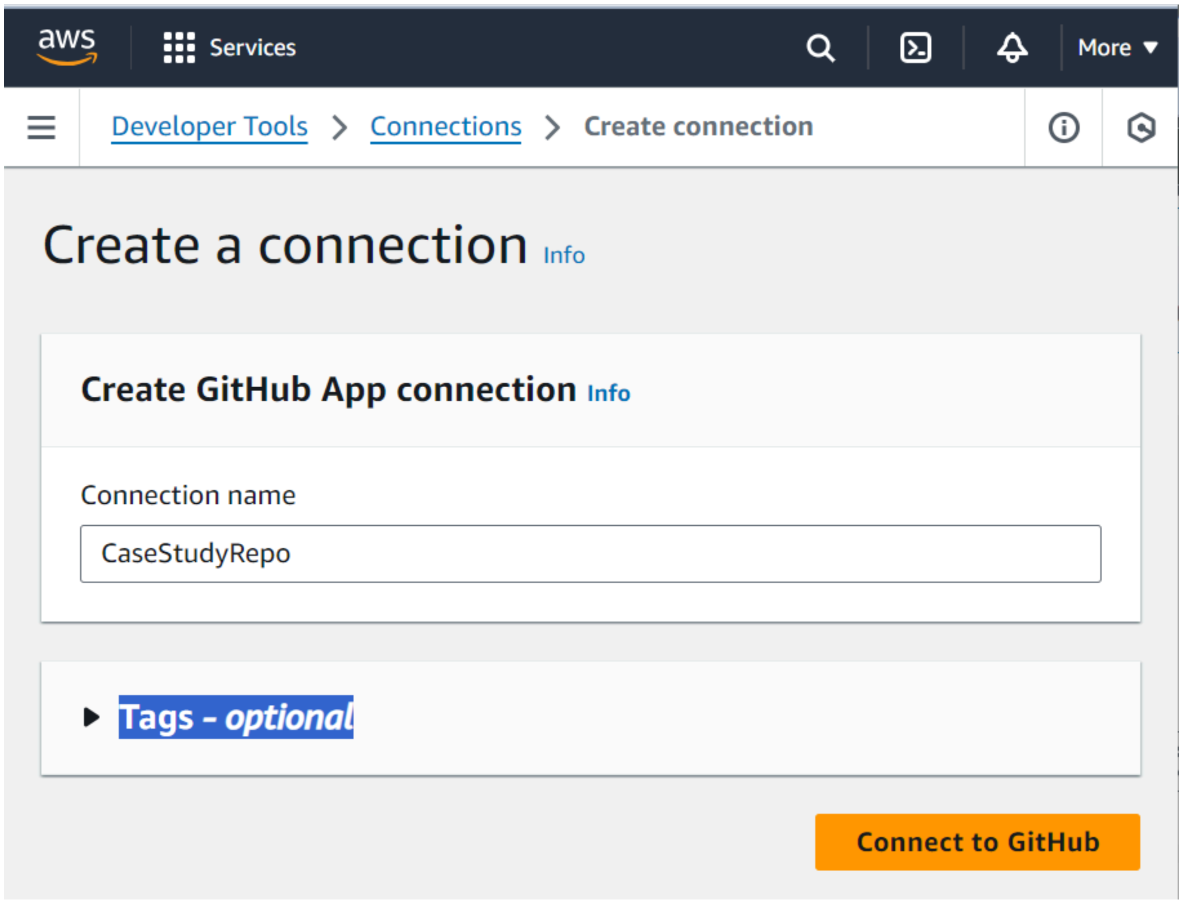
Public build access - *optional*

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



Developer Tools > Connections > Create connection

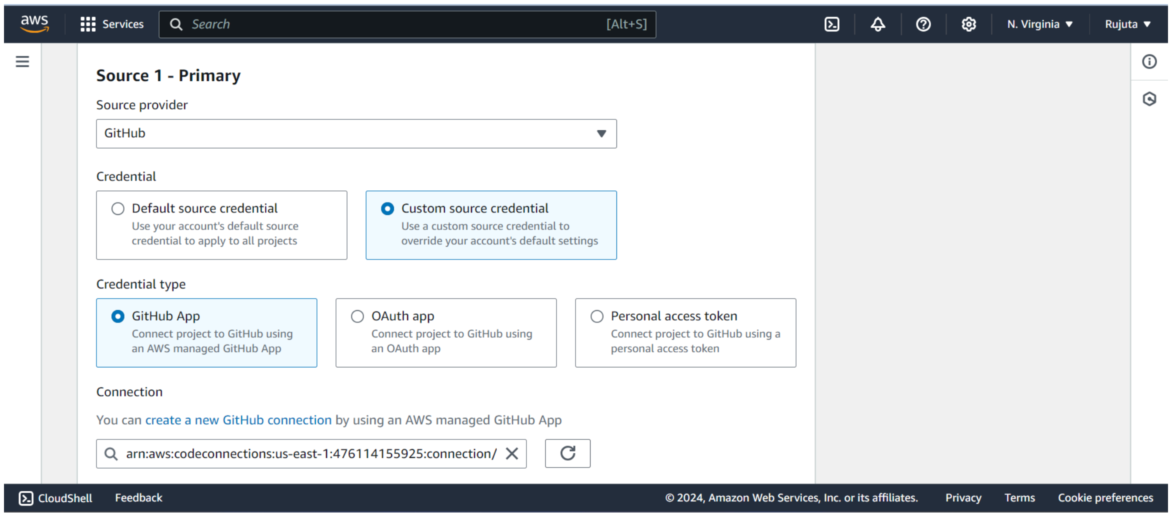
Create a connection Info

Create GitHub App connection Info

Connection name: CaseStudyRepo

Tags - *optional*

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



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]

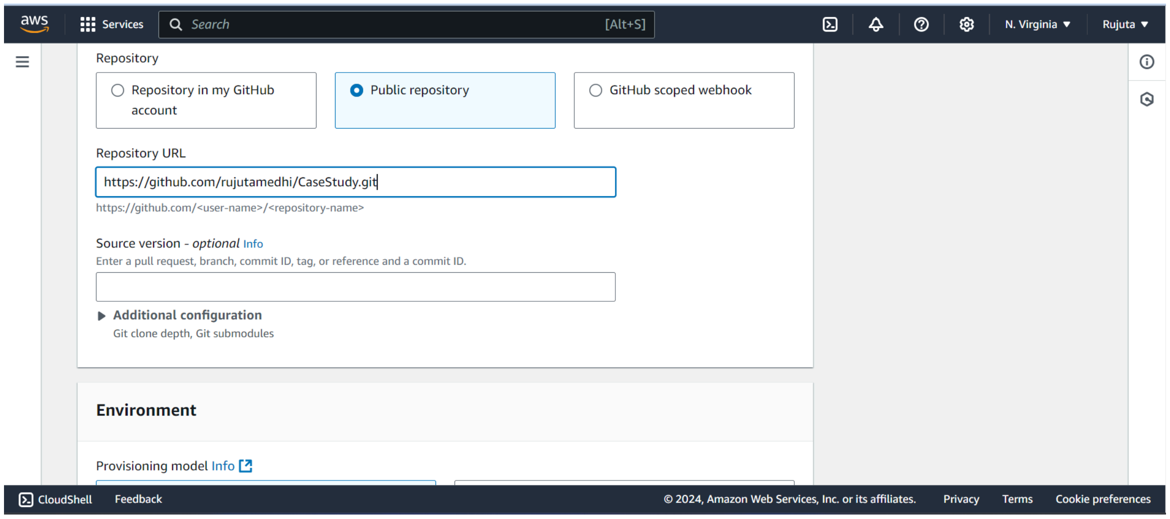
Credential type: [✓ GitHub App — Connect project to GitHub using an AWS managed GitHub App]; OAuth app — Connect project to GitHub using an OAuth app; Personal access token — Connect project to GitHub using a personal access token

Connection

You can create a new GitHub connection by using an AWS managed GitHub App

arn:aws:codeconnections:us-east-1:476114155925:connection/

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



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

Repository URL

https://github.com/rujutamedhi/CaseStudy.git

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.

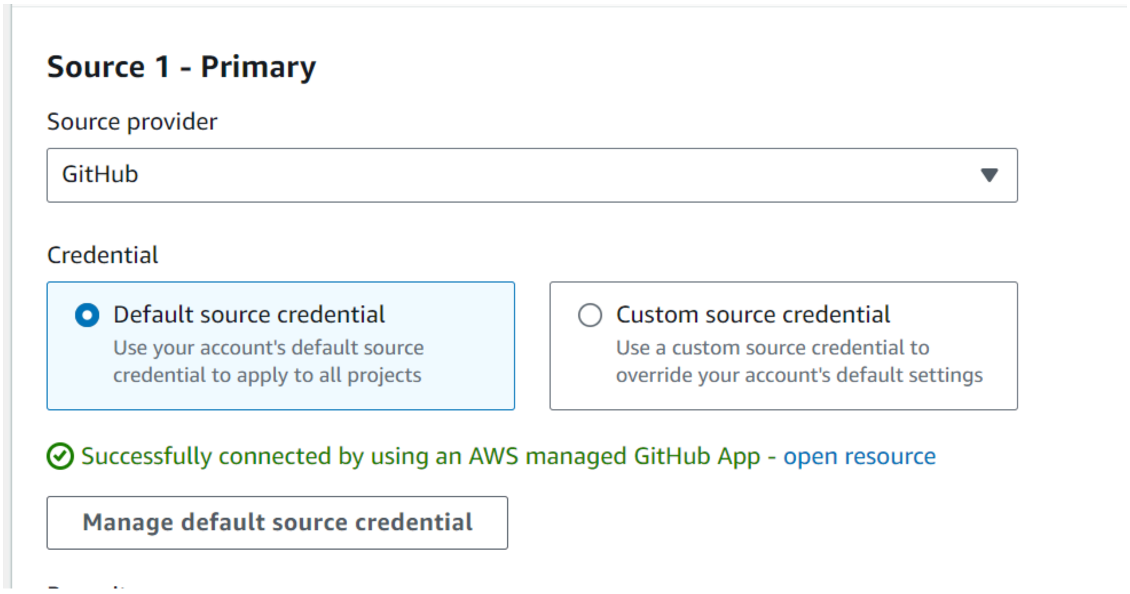
Additional configuration

Git clone depth, Git submodules

Environment

Provisioning model Info

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



Source 1 - Primary

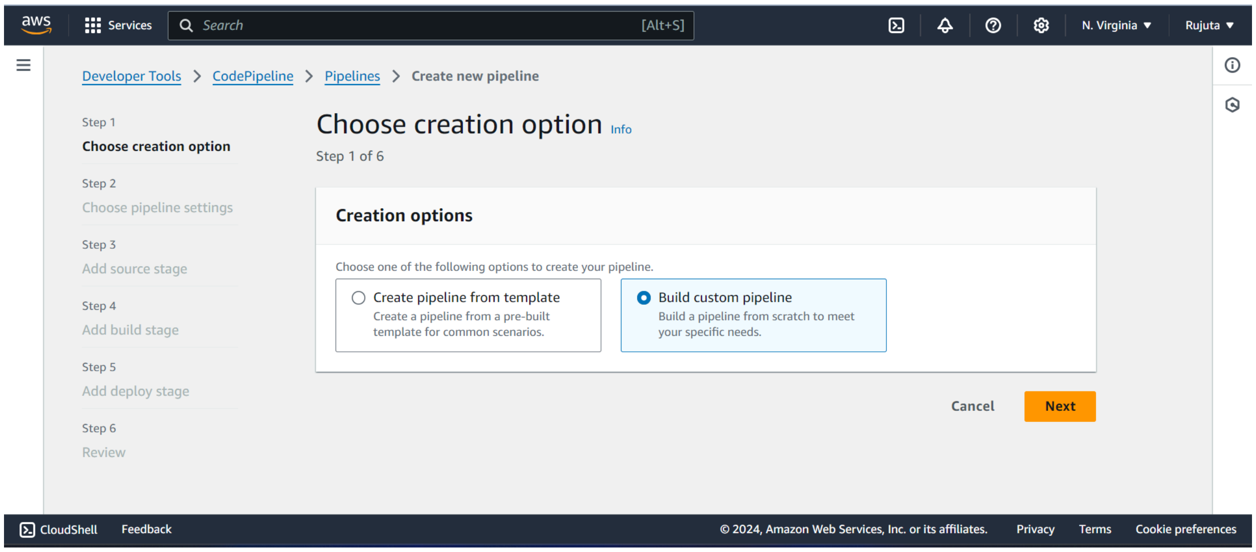
Source provider: GitHub

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

Successfully connected by using an AWS managed GitHub App - open resource

Step 5: Set Up AWS CodePipeline to Automate Lambda Deployment

1. Go to the **CodePipeline** service and click **Create Pipeline**.



Choose creation option

Step 1 of 6

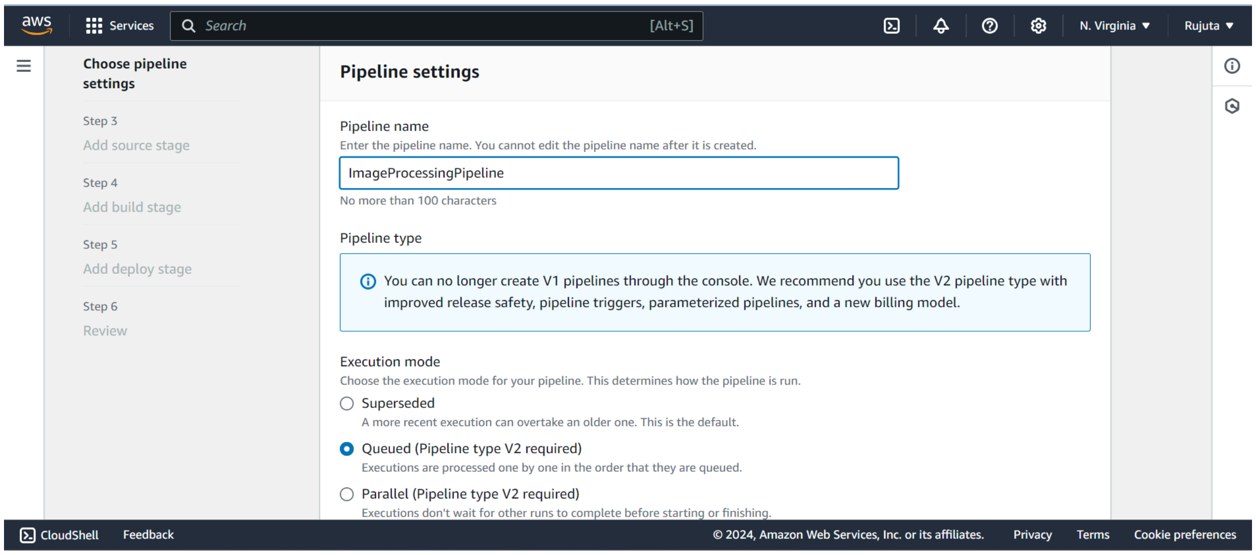
Creation options

Choose one of the following options to create your pipeline.

( ) Create pipeline from template
Create a pipeline from a pre-built template for common scenarios.

[✓ Build custom pipeline]
Build a pipeline from scratch to meet your specific needs.

2. **Pipeline Settings**:
   - **Pipeline Name**: ImageProcessingPipeline
   - **Service Role**: Allow CodePipeline to create a new role.



Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add deploy stage

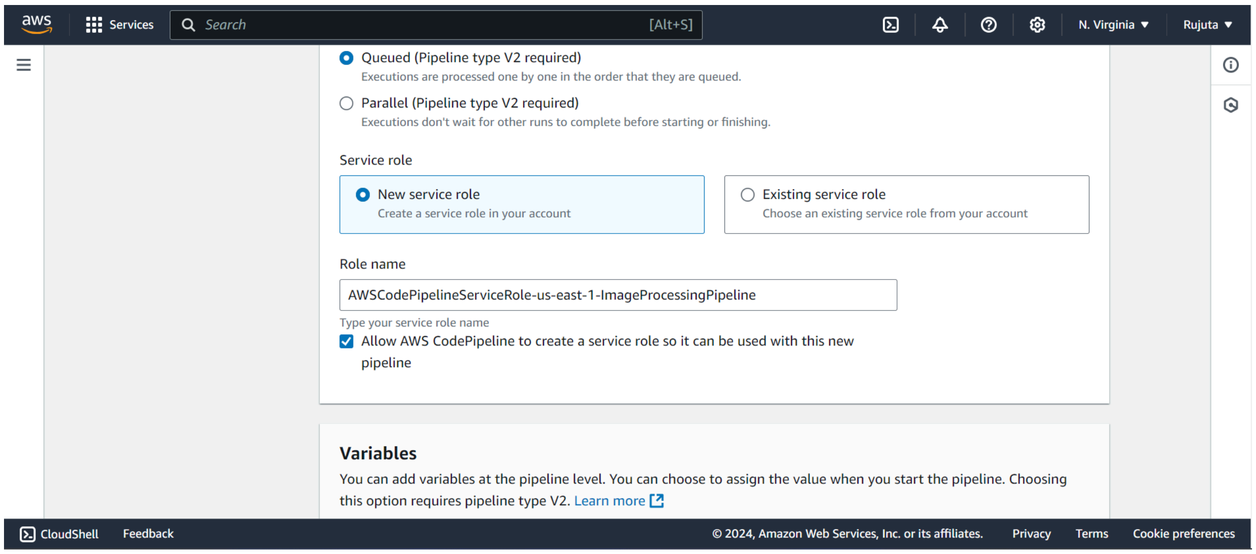
Step 6
Review

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
  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.



- ● 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.

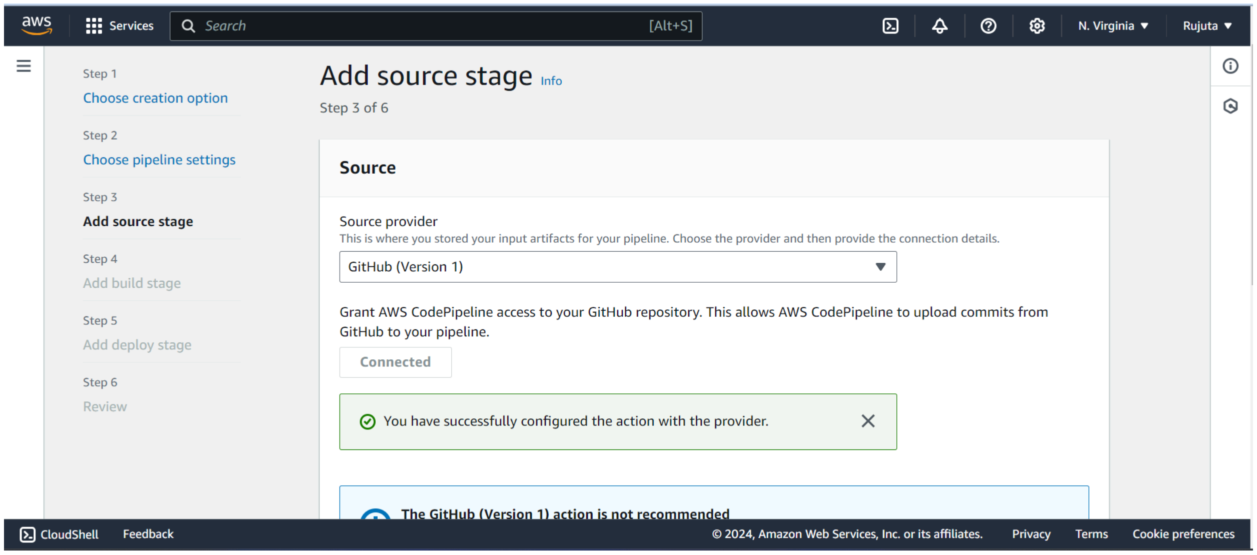
Service role
- ● New service role
  Create a service role in your account
- ○ Existing service role
  Choose an existing service role from your account

Role name
AWSCodePipelineServiceRole-us-east-1-ImageProcessingPipeline
Type your service role name
☑ Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

Variables
You can add variables at the pipeline level. You can choose to assign the value when you start the pipeline. Choosing this option requires pipeline type V2. Learn more

3. Source Stage (Code Repository):

- ○ For **Source Provider**, choose **GitHub** or **AWS CodeCommit** based on your code repository.



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)

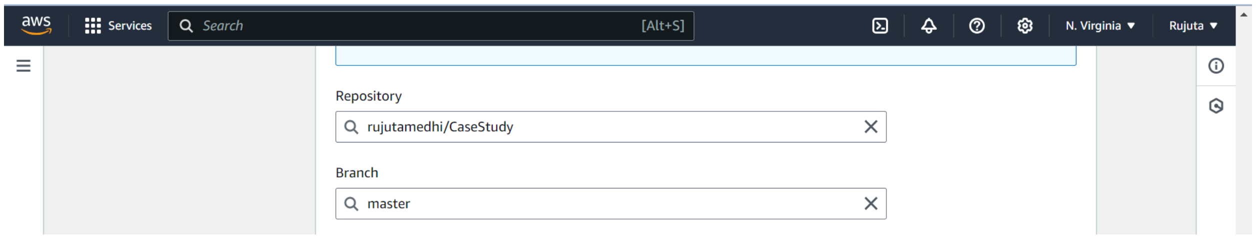
Grant AWS CodePipeline access to your GitHub repository. This allows AWS CodePipeline to upload commits from GitHub to your pipeline.

Connected

You have successfully configured the action with the provider.

The GitHub (Version 1) action is not recommended

Connect your repository that contains the Lambda code



Repository

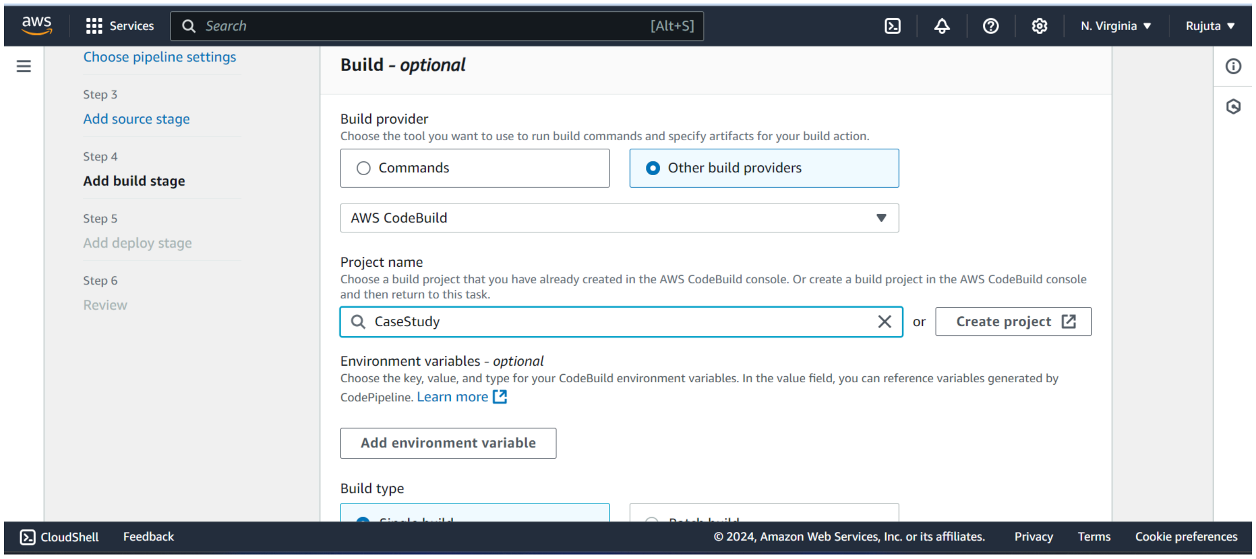
rujutamedhi/CaseStudy

Branch

master

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.

CaseStudy

or Create project

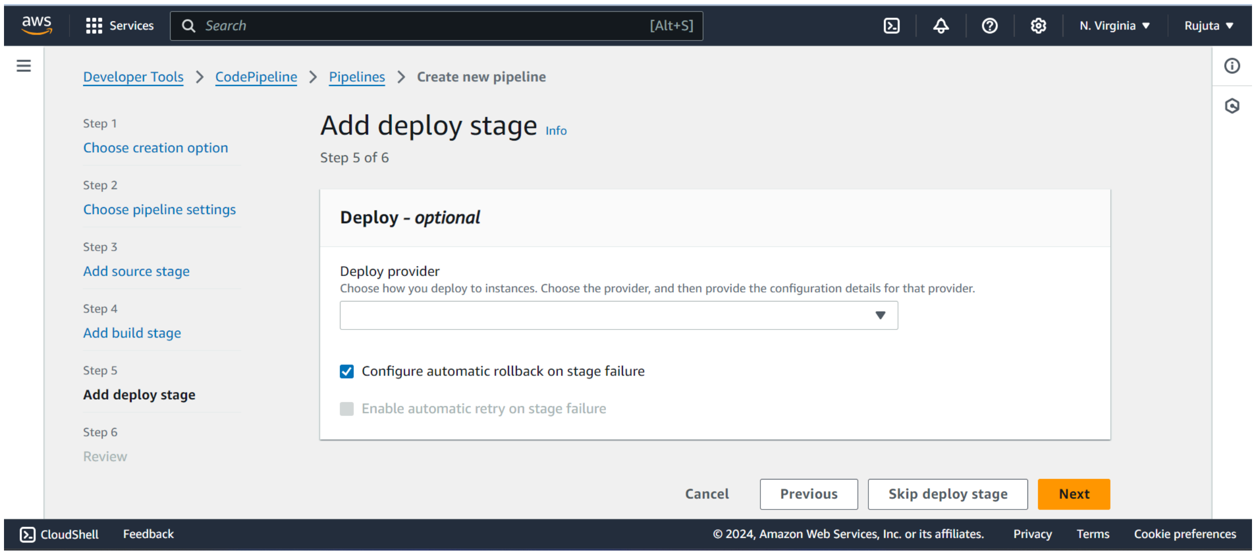
Environment variables - *optional*
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more

Add environment variable

Build type

5. Deploy Stage (Deploy to Lambda):

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



Add deploy stage Info

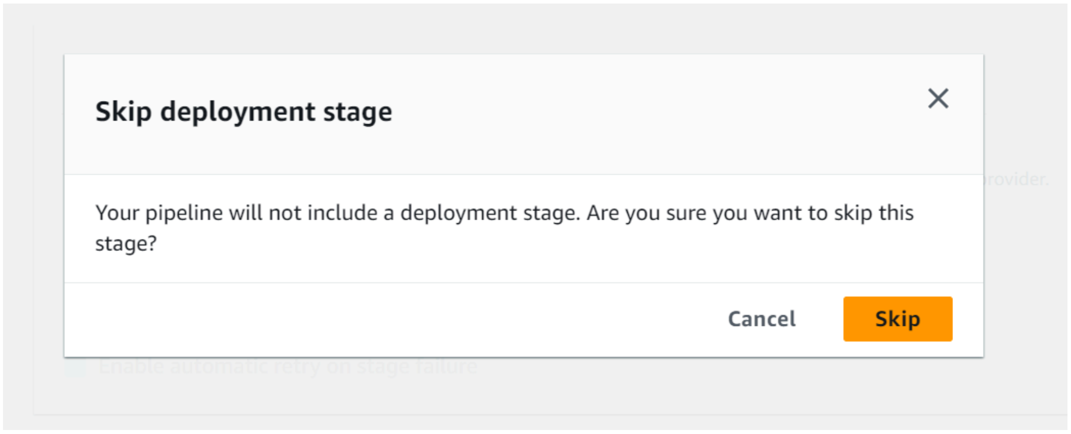
Step 5 of 6

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

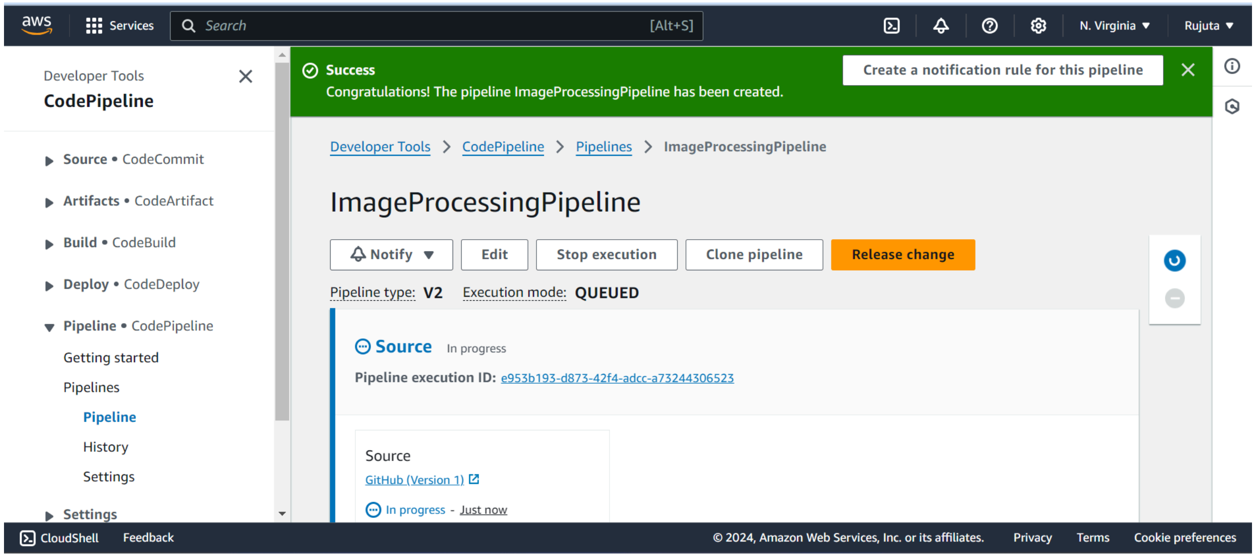
Enable automatic retry on stage failure



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.

Developer Tools > CodePipeline > Pipelines > ImageProcessingPipeline

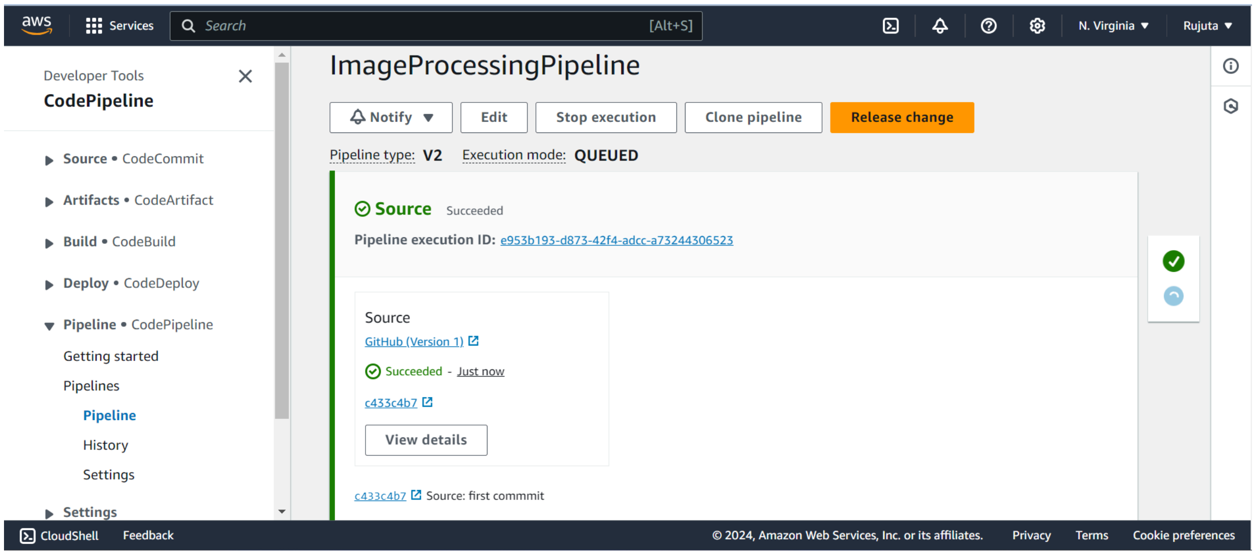
ImageProcessingPipeline

Pipeline type: V2 Execution mode: QUEUED

Source In progress
Pipeline execution ID: e953b193-d873-42f4-adcc-a73244306523

Source
GitHub (Version 1)
In progress - Just now

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



ImageProcessingPipeline

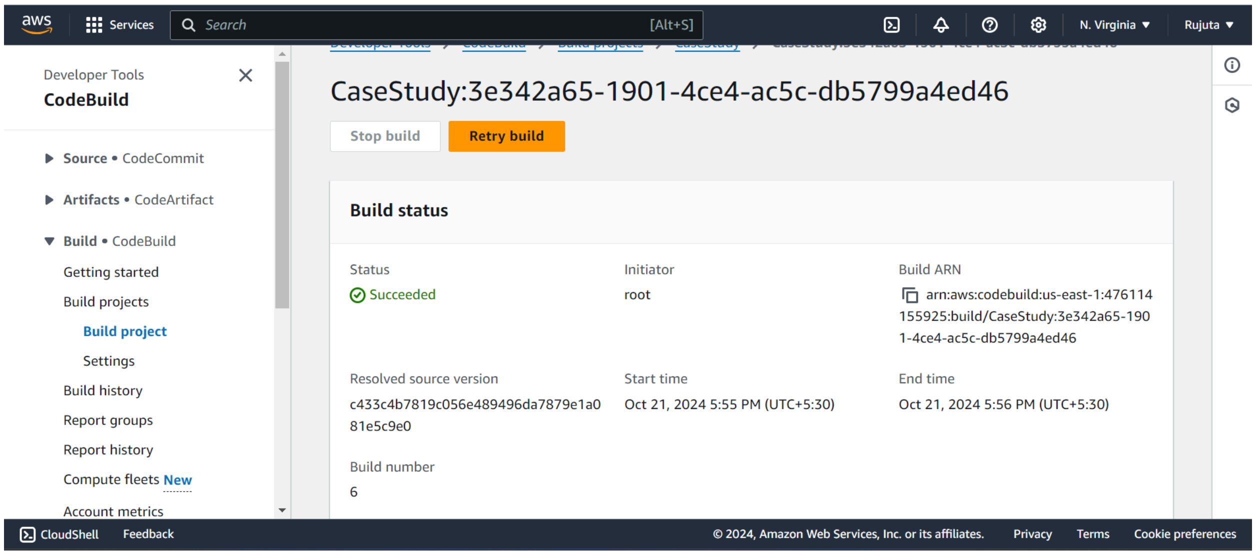
Pipeline type: V2 Execution mode: QUEUED

Source Succeeded
Pipeline execution ID: e953b193-d873-42f4-adcc-a73244306523

Source
GitHub (Version 1)
Succeeded - Just now
c433c4b7

c433c4b7 Source: first commmit

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



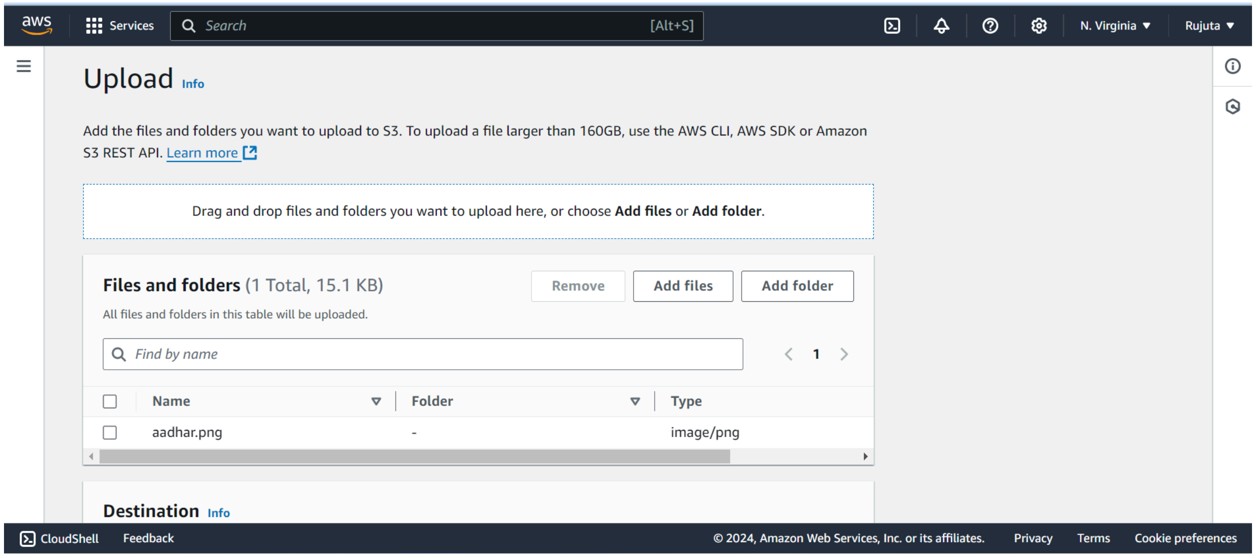
CaseStudy:3e342a65-1901-4ce4-ac5c-db5799a4ed46

Build status

Status	Initiator	Build ARN
Succeeded	root	arn:aws:codebuild:us-east-1:476114155925:build/CaseStudy:3e342a65-1901-4ce4-ac5c-db5799a4ed46
Resolved source version	**Start time**	**End time**
c433c4b7819c056e489496da7879e1a081e5c9e0	Oct 21, 2024 5:55 PM (UTC+5:30)	Oct 21, 2024 5:56 PM (UTC+5:30)
Build number		
6		

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



Upload

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

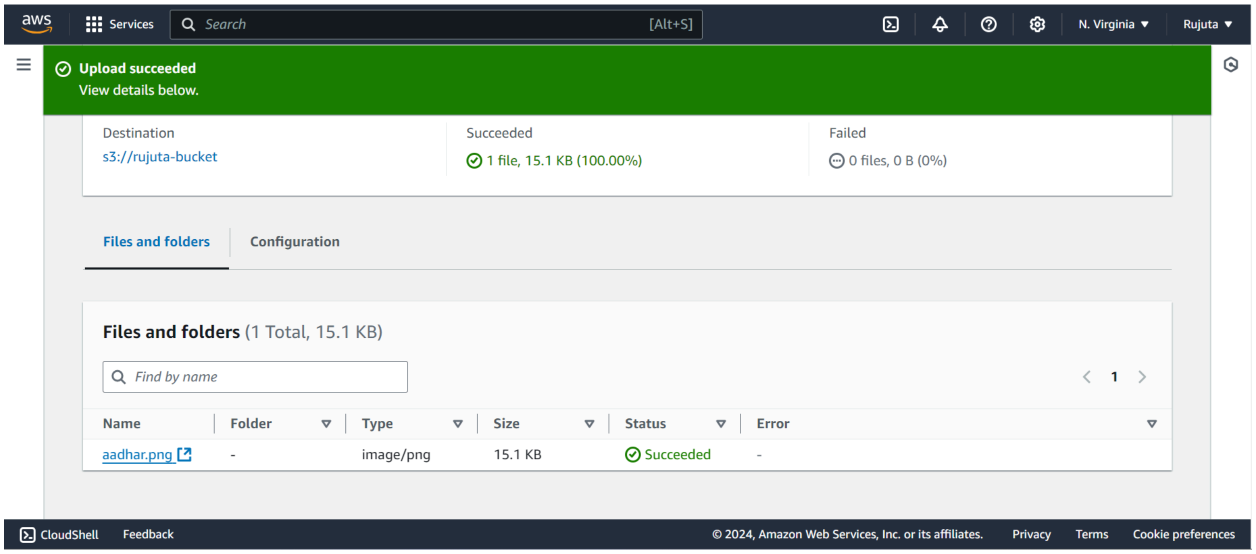
Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 15.1 KB)

All files and folders in this table will be uploaded.

Name	Folder	Type
aadhar.png	-	image/png

Destination



Upload succeeded
View details below.

Destination
s3://rujuta-bucket

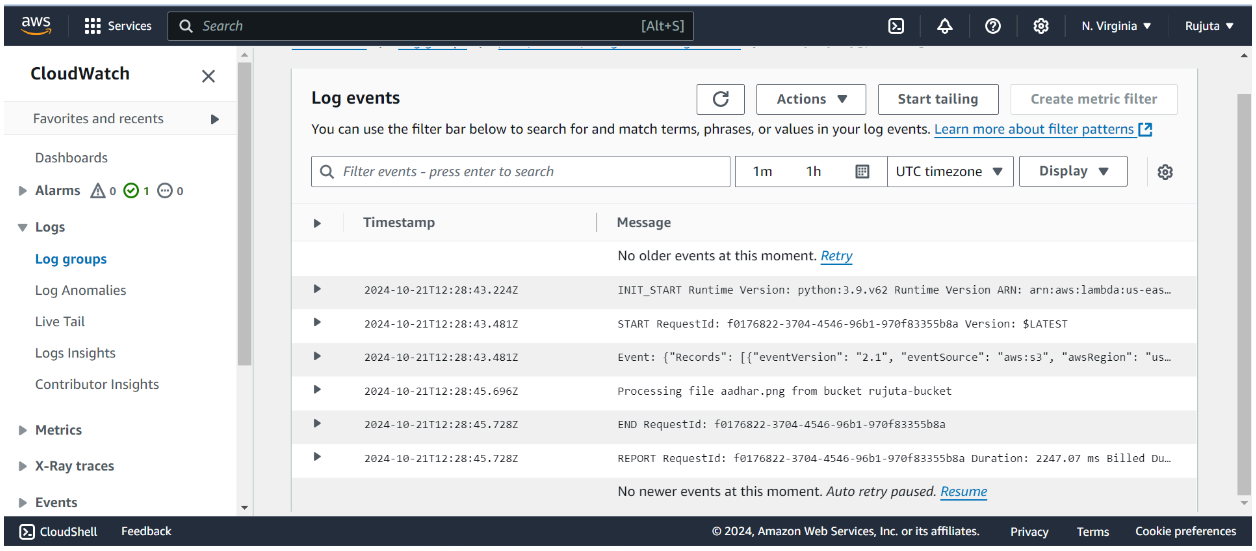
Succeeded
1 file, 15.1 KB (100.00%)

Failed
0 files, 0 B (0%)

Files and folders (1 Total, 15.1 KB)

Name	Folder	Type	Size	Status	Error
aadhar.png	-	image/png	15.1 KB	Succeeded	-

1. **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).



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
2024-10-21T12:28:43.224Z   INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:us-eas…
2024-10-21T12:28:43.481Z   START RequestId: f0176822-3704-4546-96b1-970f83355b8a Version: $LATEST
2024-10-21T12:28:43.481Z   Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us…
2024-10-21T12:28:45.696Z   Processing file aadhar.png from bucket rujuta-bucket
2024-10-21T12:28:45.728Z   END RequestId: f0176822-3704-4546-96b1-970f83355b8a
2024-10-21T12:28:45.728Z   REPORT RequestId: f0176822-3704-4546-96b1-970f83355b8a Duration: 2247.07 ms Billed Du…
                           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).

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 {
```

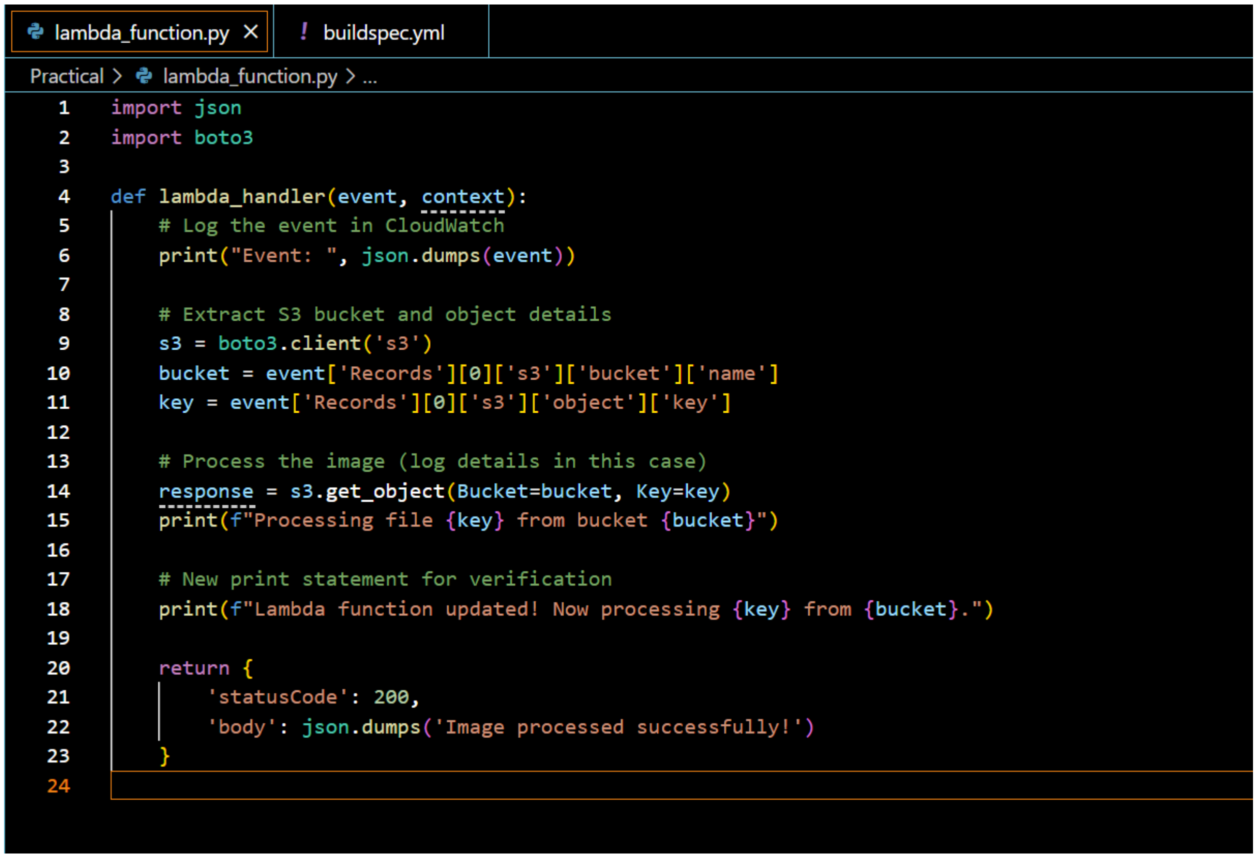
'statusCode': 200,

'body': json.dumps('Image processed successfully!')

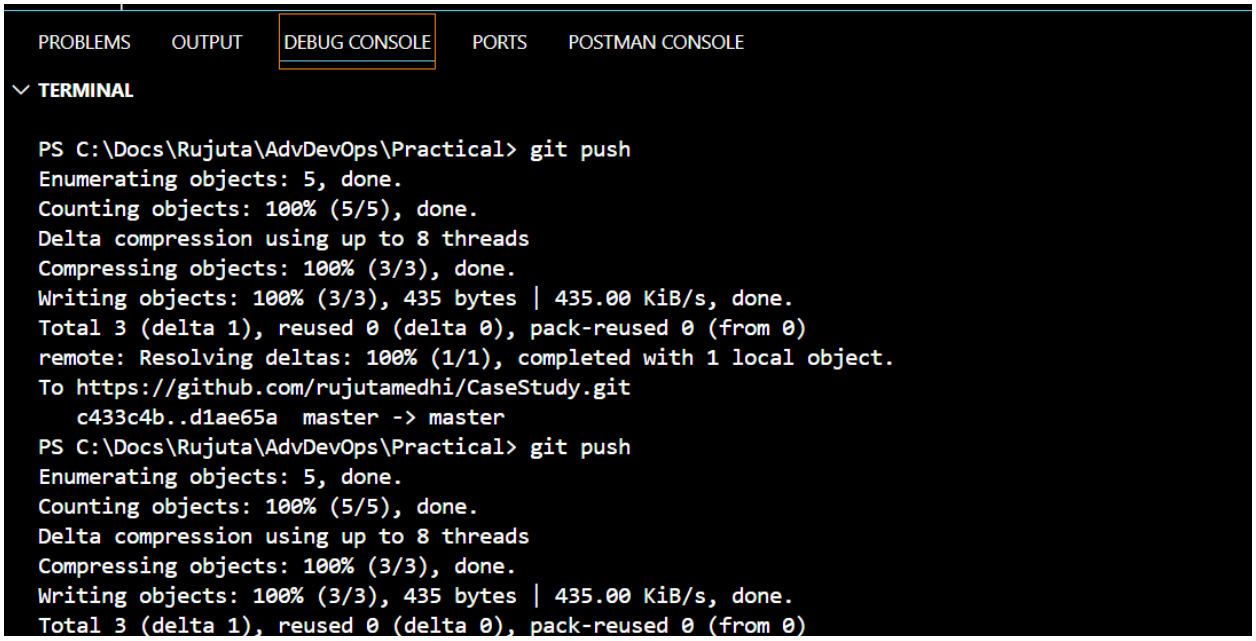
}

1. **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
```

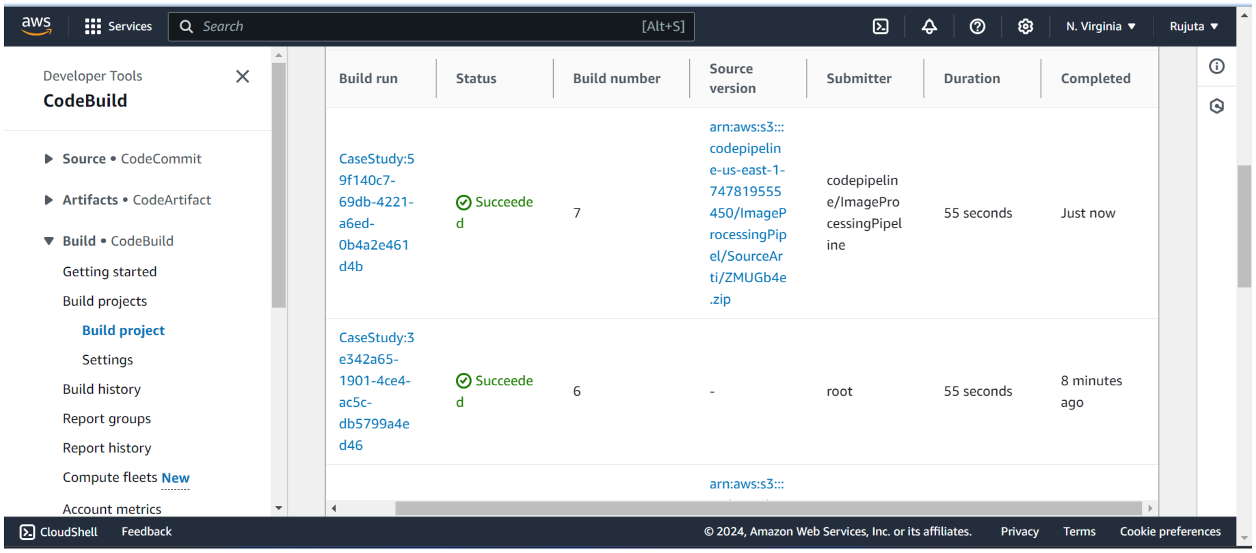


```python
import json
import boto3

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!')
    }

```



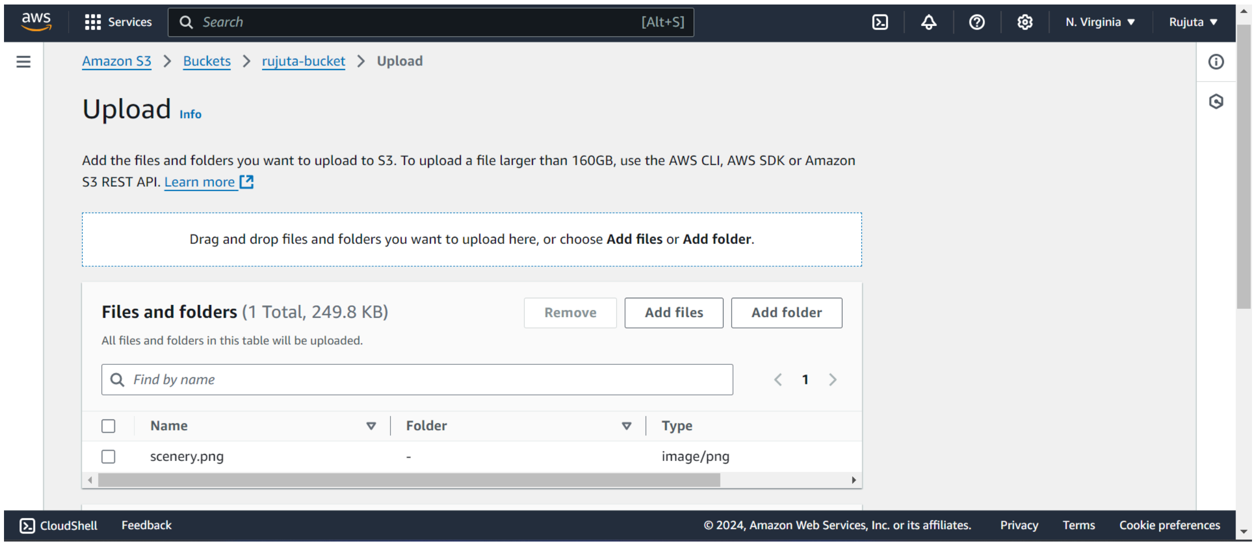
```
PS C:\Docs\Rujuta\AdvDevOps\Practical> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 435 bytes | 435.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/rujutamedhi/CaseStudy.git
   c433c4b..d1ae65a  master -> master
PS C:\Docs\Rujuta\AdvDevOps\Practical> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 435 bytes | 435.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
```

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



Build run	Status	Build number	Source version	Submitter	Duration	Completed
CaseStudy:59f140c7-69db-4221-a6ed-0b4a2e461d4b	Succeeded	7	arn:aws:s3:::codepipeline-us-east-1-747819555450/ImageProcessingPipel/SourceArti/ZMUGb4e.zip	codepipeline/ImageProcessingPipeline	55 seconds	Just now
CaseStudy:3e342a65-1901-4ce4-ac5c-db5799a4ed46	Succeeded	6	-	root	55 seconds	8 minutes ago
			arn:aws:s3:::			

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



Amazon S3 > Buckets > rujuta-bucket > Upload

Upload Info

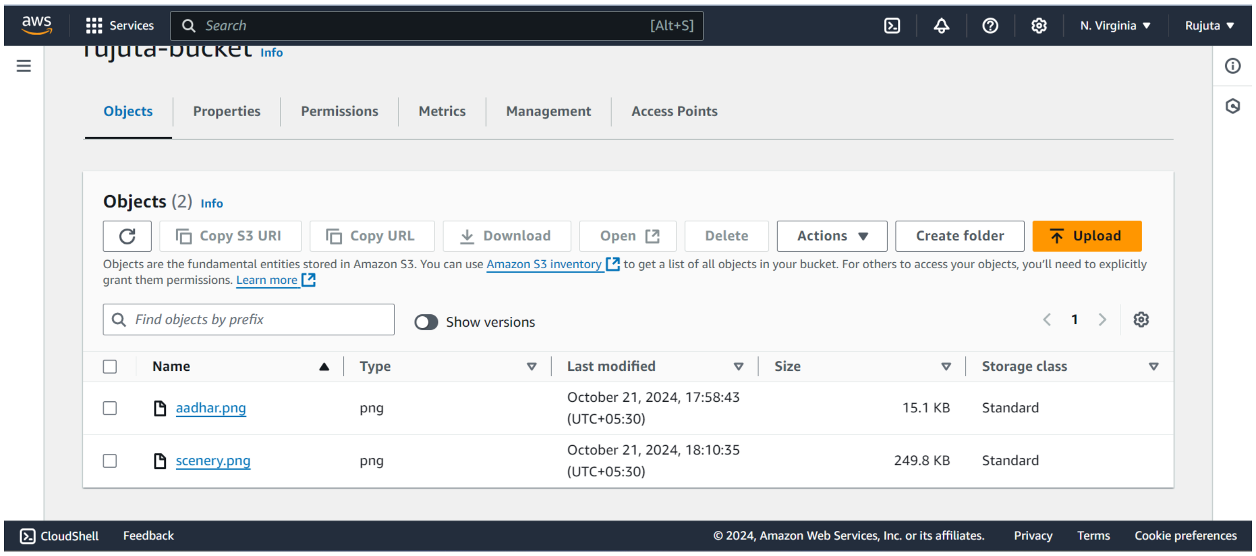
Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more

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

Files and folders (1 Total, 249.8 KB)

All files and folders in this table will be uploaded.

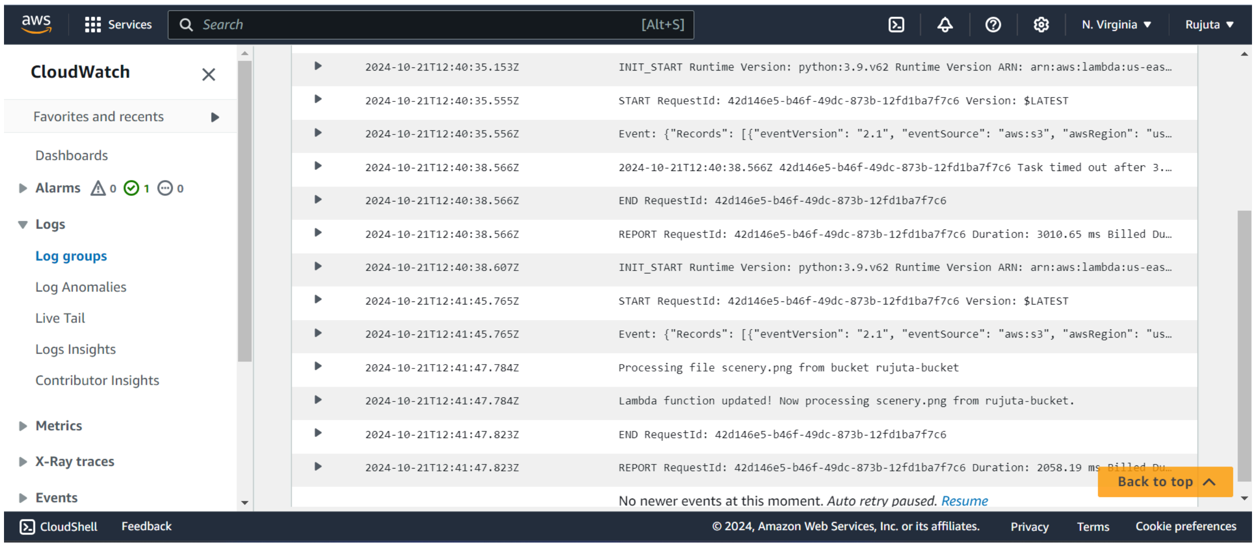
Name	Folder	Type
scenery.png	-	image/png



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
aadhar.png	png	October 21, 2024, 17:58:43 (UTC+05:30)	15.1 KB	Standard
scenery.png	png	October 21, 2024, 18:10:35 (UTC+05:30)	249.8 KB	Standard



```
2024-10-21T12:40:35.153Z    INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:us-eas…
2024-10-21T12:40:35.555Z    START RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6 Version: $LATEST
2024-10-21T12:40:35.556Z    Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us…
2024-10-21T12:40:38.566Z    2024-10-21T12:40:38.566Z 42d146e5-b46f-49dc-873b-12fd1ba7f7c6 Task timed out after 3.…
2024-10-21T12:40:38.566Z    END RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6
2024-10-21T12:40:38.566Z    REPORT RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6 Duration: 3010.65 ms Billed Du…
2024-10-21T12:40:38.607Z    INIT_START Runtime Version: python:3.9.v62 Runtime Version ARN: arn:aws:lambda:us-eas…
2024-10-21T12:41:45.765Z    START RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6 Version: $LATEST
2024-10-21T12:41:45.765Z    Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us…
2024-10-21T12:41:47.784Z    Processing file scenery.png from bucket rujuta-bucket
2024-10-21T12:41:47.784Z    Lambda function updated! Now processing scenery.png from rujuta-bucket.
2024-10-21T12:41:47.823Z    END RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6
2024-10-21T12:41:47.823Z    REPORT RequestId: 42d146e5-b46f-49dc-873b-12fd1ba7f7c6 Duration: 2058.19 ms Billed Du…
                            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.