Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

Title: Serverless Image Processing Workflow

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.

Introduction

Overview:

The case study focuses on developing a serverless image processing workflow using AWS services. When an image is uploaded to an S3 bucket, an AWS Lambda function is triggered to log and process the image. AWS CodePipeline automates the deployment of the Lambda function, ensuring that any updates to the function are seamlessly integrated and deployed. This setup leverages the event-driven architecture and scalability of AWS, providing an efficient and cost-effective solution for image processing.

Key Features and Applications:

Key Features:

1. Serverless Architecture: Utilizes AWS Lambda to execute code in response to image uploads without provisioning or managing servers.
2. Event-Driven Processing: Automatically triggers the Lambda function upon image upload to the S3 bucket.

3. Automated Deployment: Employs AWS CodePipeline to automate the build, test, and deployment stages of the Lambda function, ensuring continuous integration and delivery.
4. Scalability: Leverages AWS's inherent scalability to handle varying loads without manual intervention.
5. Cost-Effectiveness: Pay-as-you-go model for AWS Lambda and S3, reducing costs by eliminating the need for dedicated server resources.

Applications:

1. Real-Time Image Processing: Automatically resize, convert, or analyze images upon upload, useful for applications like photo-sharing platforms, e-commerce sites, and social media.
2. Metadata Extraction: Extract and log metadata from images for cataloging and indexing purposes.
3. Image Moderation: Automatically analyze images for inappropriate content, facilitating content moderation.
4. Thumbnail Generation: Create thumbnails for uploaded images, which can be used in web and mobile applications to improve loading times and user experience.

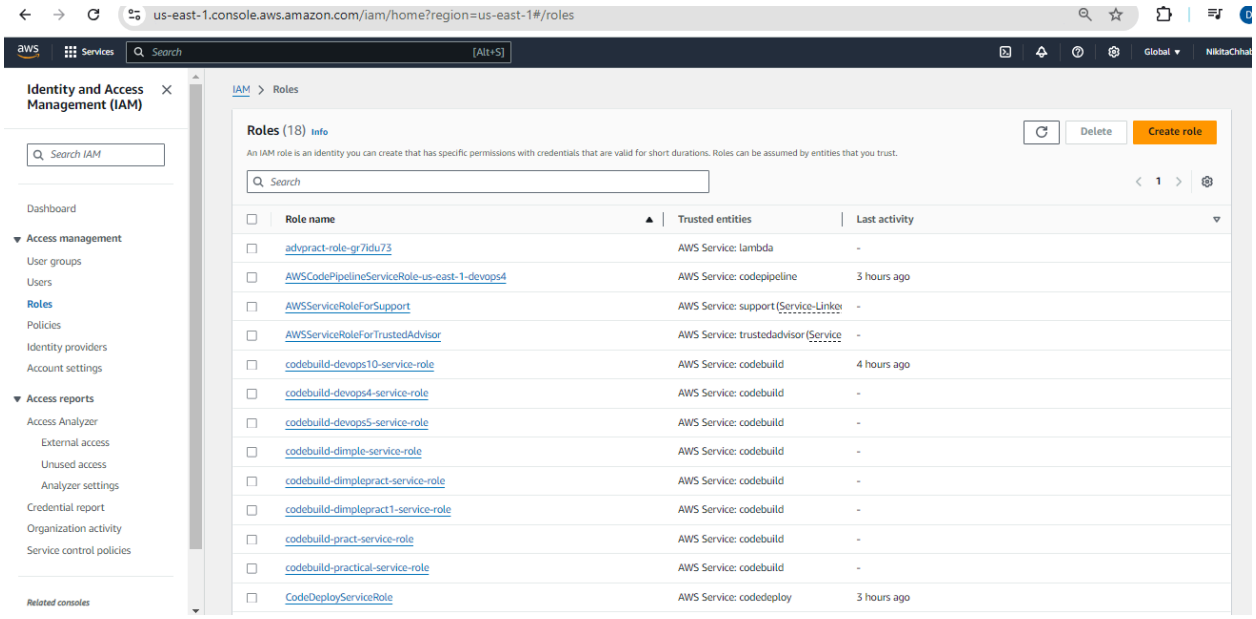
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

Step-by-Step Explanation

STEP I: Create IAM Role:

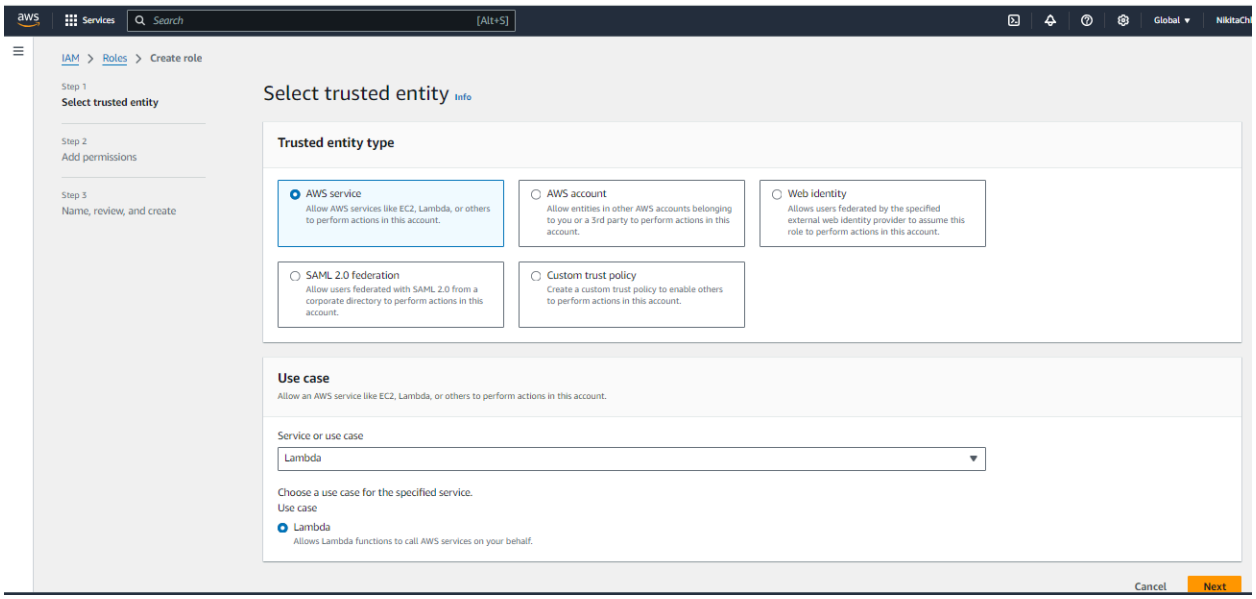
STEP 1:Login to to your AWS console account because AWS academy account doesn't give you permission ,then go on roles and click on create role.



STEP 2:In trusted entity type select “ AWS service” in use case select “lambda”and then click on next.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



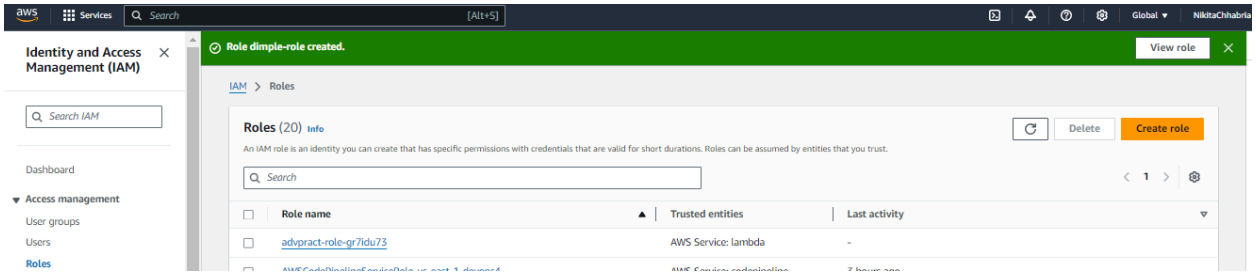
STEP 3: Attach policies such as:

- AmazonS3FullAccess
- AmazonCodeBuildAdminAcess
- AmazonCodePipeline_FullAccess
- AWSLambda_FullAccess
- CloudWatchLogsFullAccess

Then in Role name add the name of the role and then click on create role.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

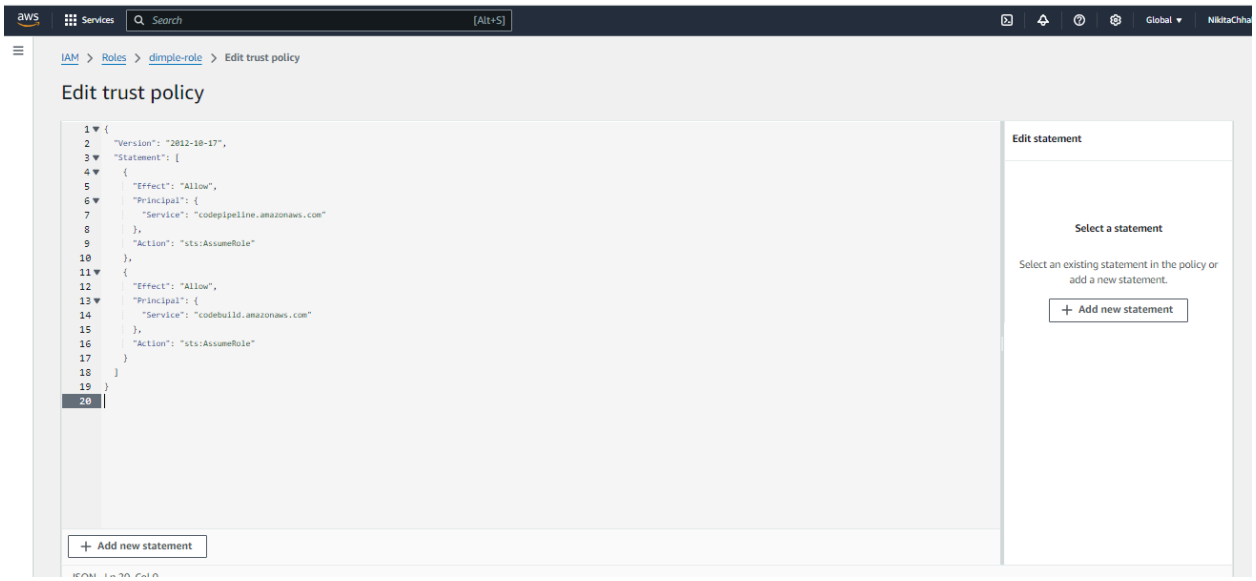


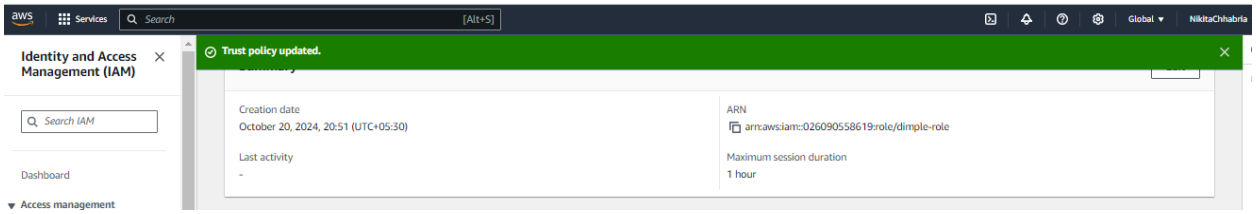
STEP 4:After that created go ob IAM >Roles >role that you have created >edit trust policies in that write this code and then click on update the policy .

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "codepipeline.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  },
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "codebuild.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**





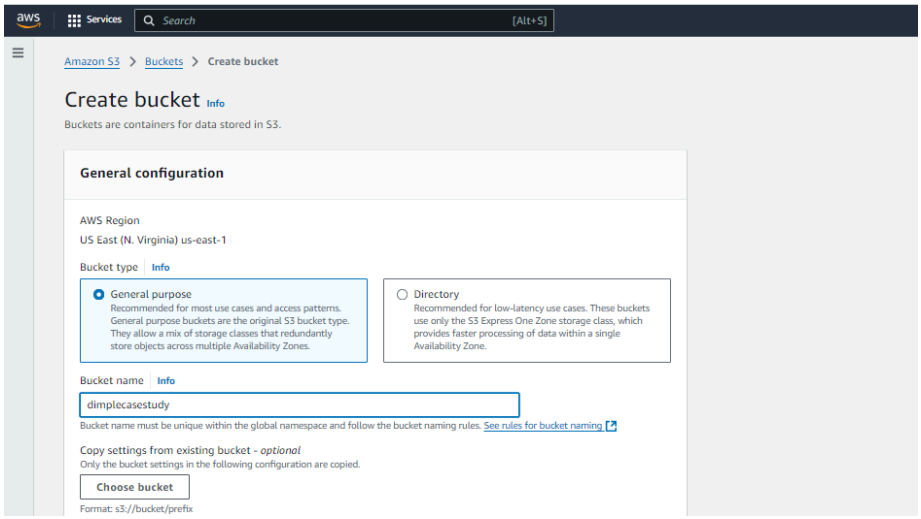
STEP II:Create an S3 bucket :

STEP 1:

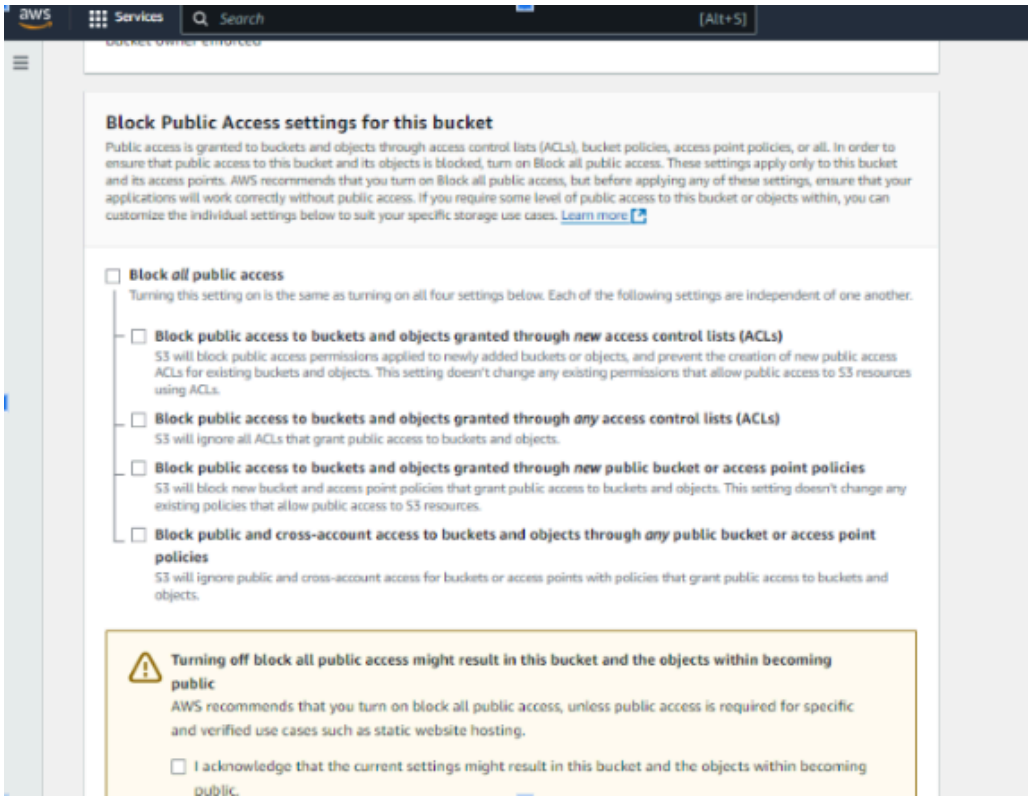
- Login to AWS Management Console.
- Go to S3 Services and forSearch for "S3" in the search bar and select it

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



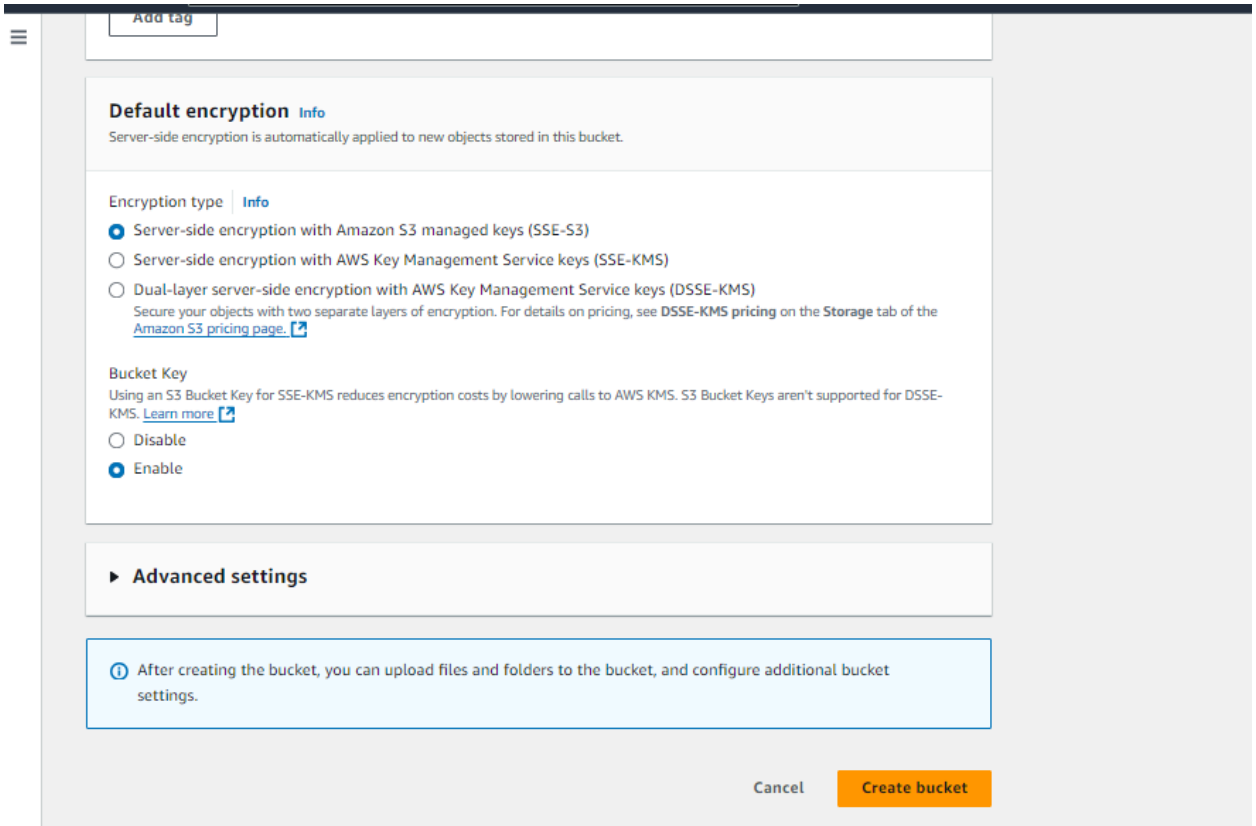
STEP 2:Enable all block public access settings for this bucket.



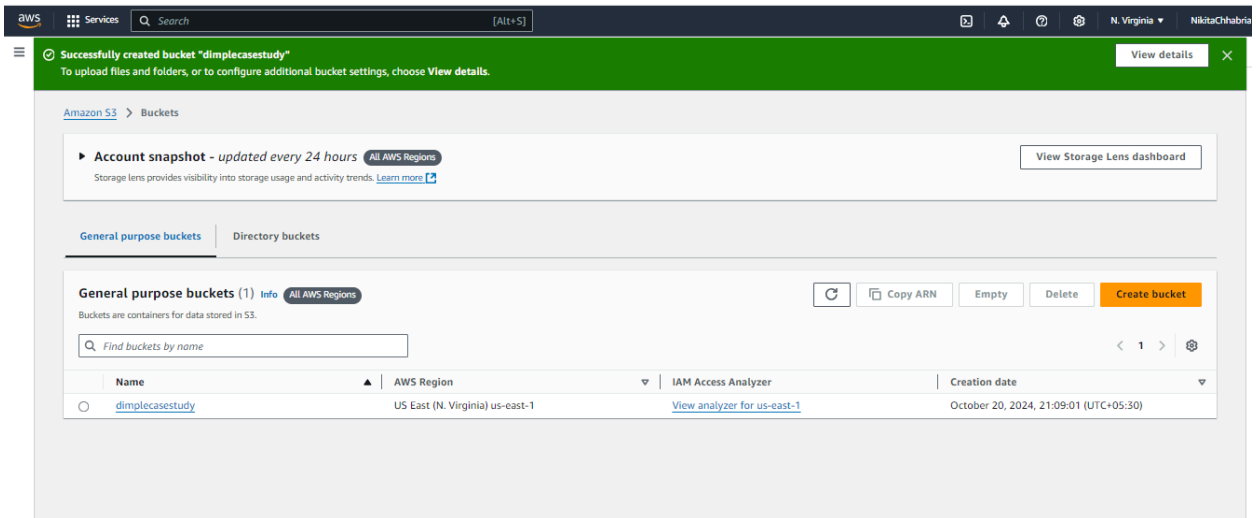
STEP 3:Let all the default settings remain same and then Click on create bucket .

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



STEP 4: The bucket gets created successfully.



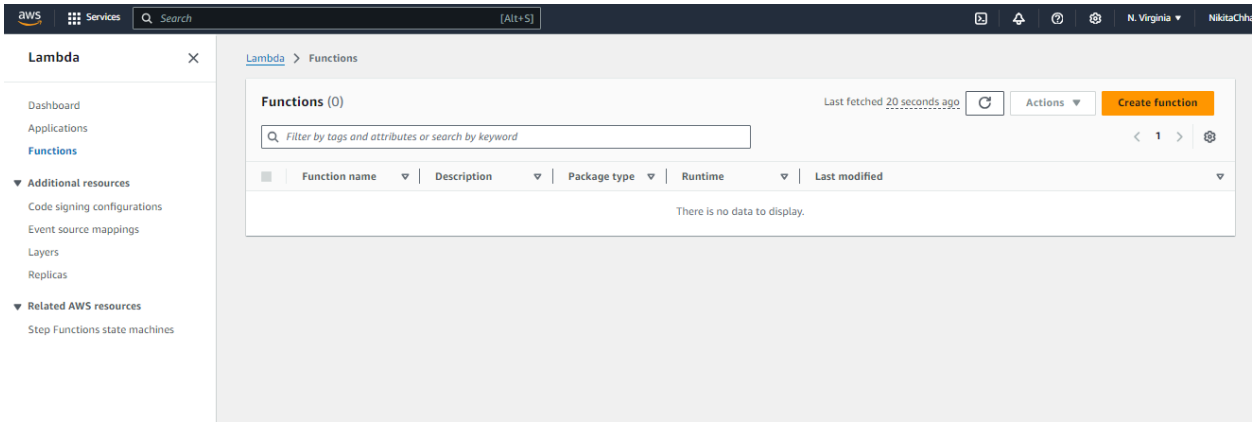
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

STEP III:Create a Lambda Function

STEP1:Go to Lambda Service and search for "Lambda" in the search bar and select it.

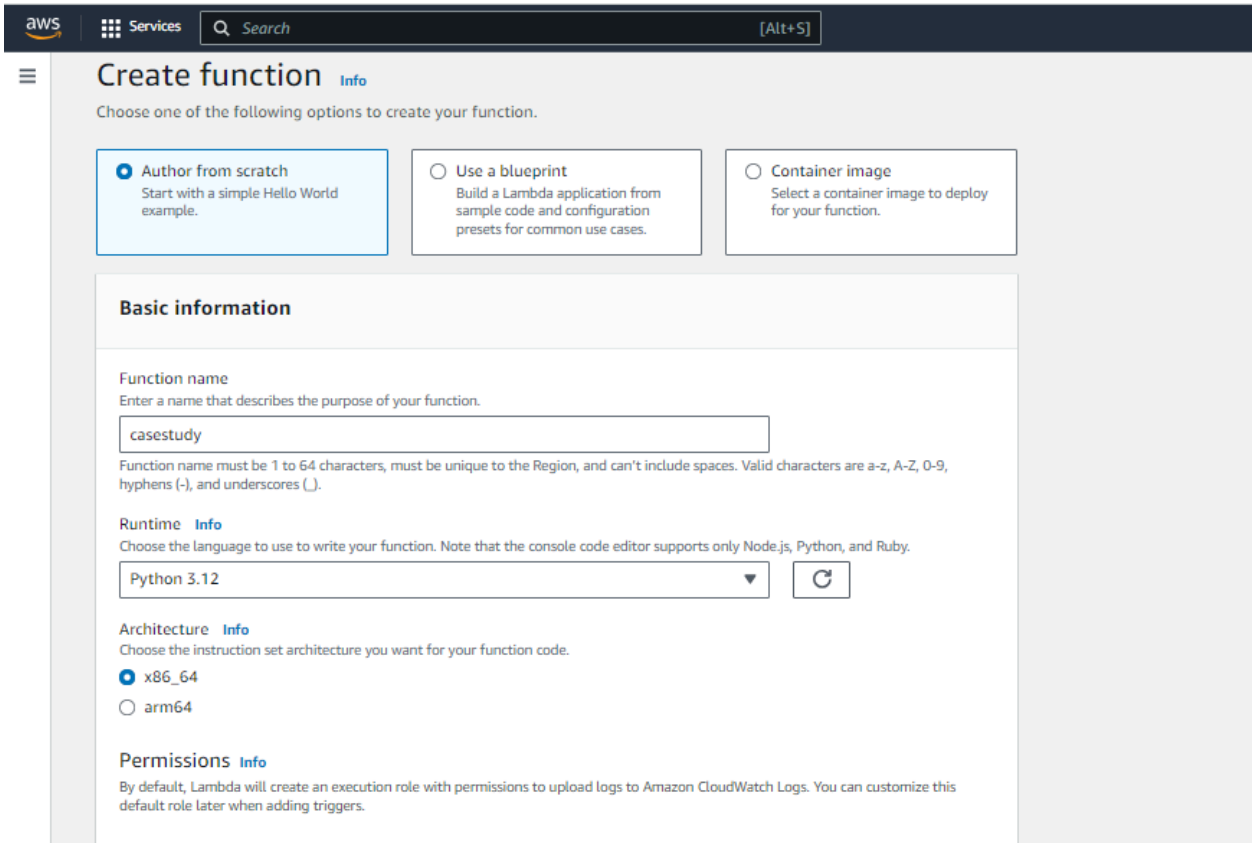
STEP 2: Create a New Lambda Function:Click on create new



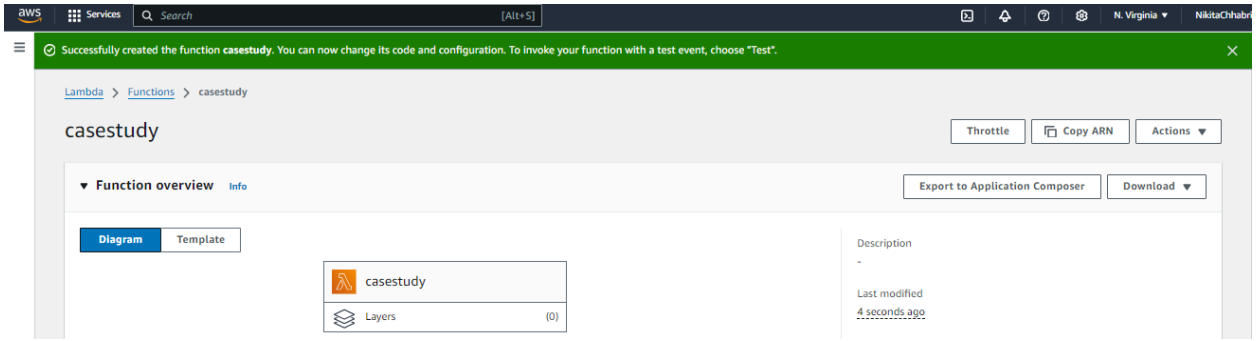
STEP 3:Select “Author from scratch” and then add then add the name of the function name and then select the latest python version and then click on create function.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



STEP 4:The lambda function is created successfully.



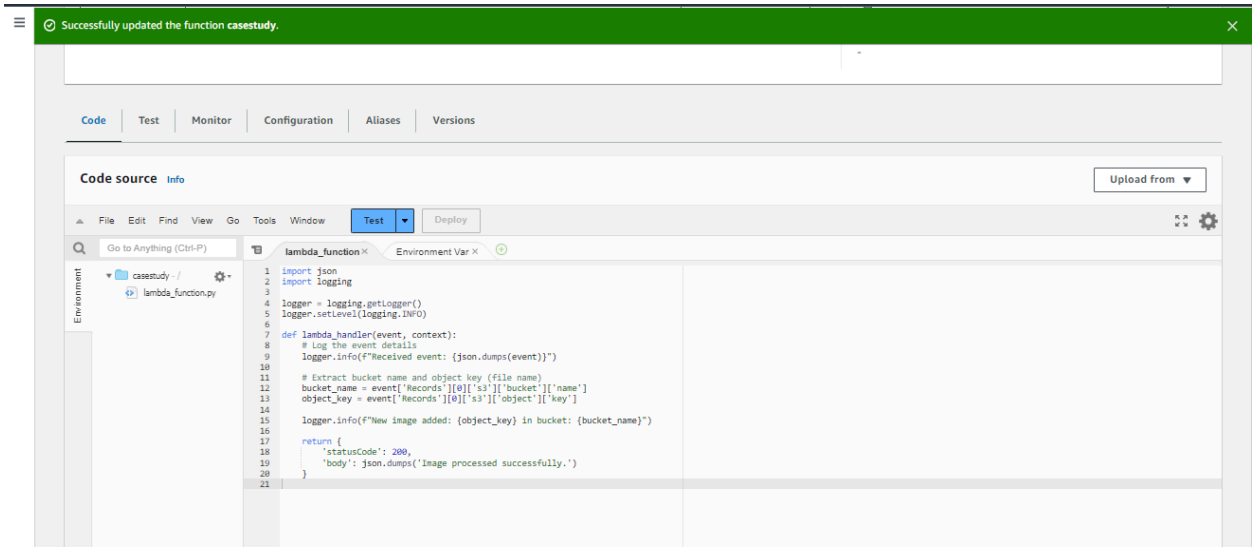
STEP 5:

Write the Lambda Function Code:

In the "Function code" section, replace the default code with the following Python code:

```
import json

import logging
```

```
logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    # Log the event details
    logger.info(f"Received event: {json.dumps(event)}")
    # Extract bucket name and object key (file name)
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    object_key = event['Records'][0]['s3']['object']['key']
    logger.info(f"New image added: {object_key} in bucket: {bucket_name}") return {
        'statusCode': 200,
        'body': json.dumps('Image processed successfully.')
    }
```



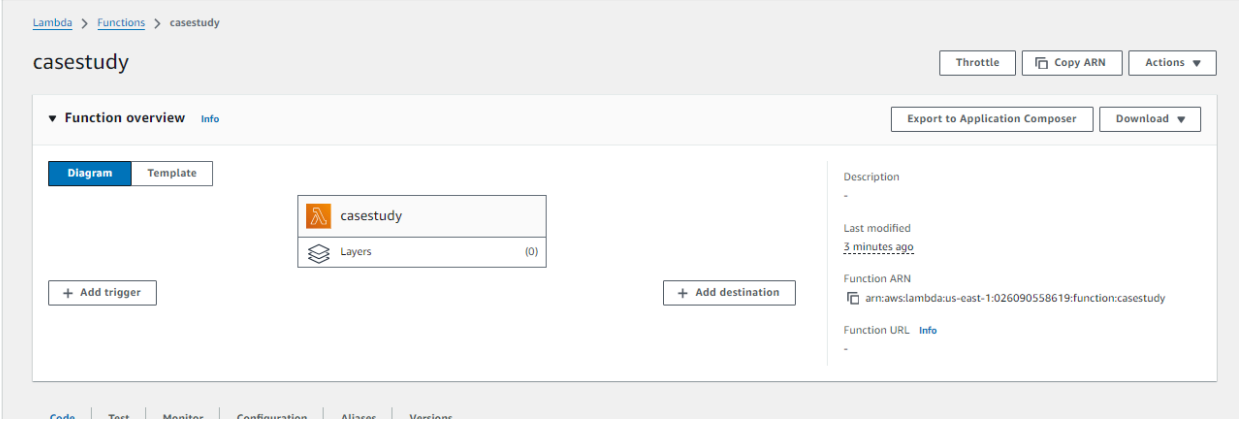
STEP 6:

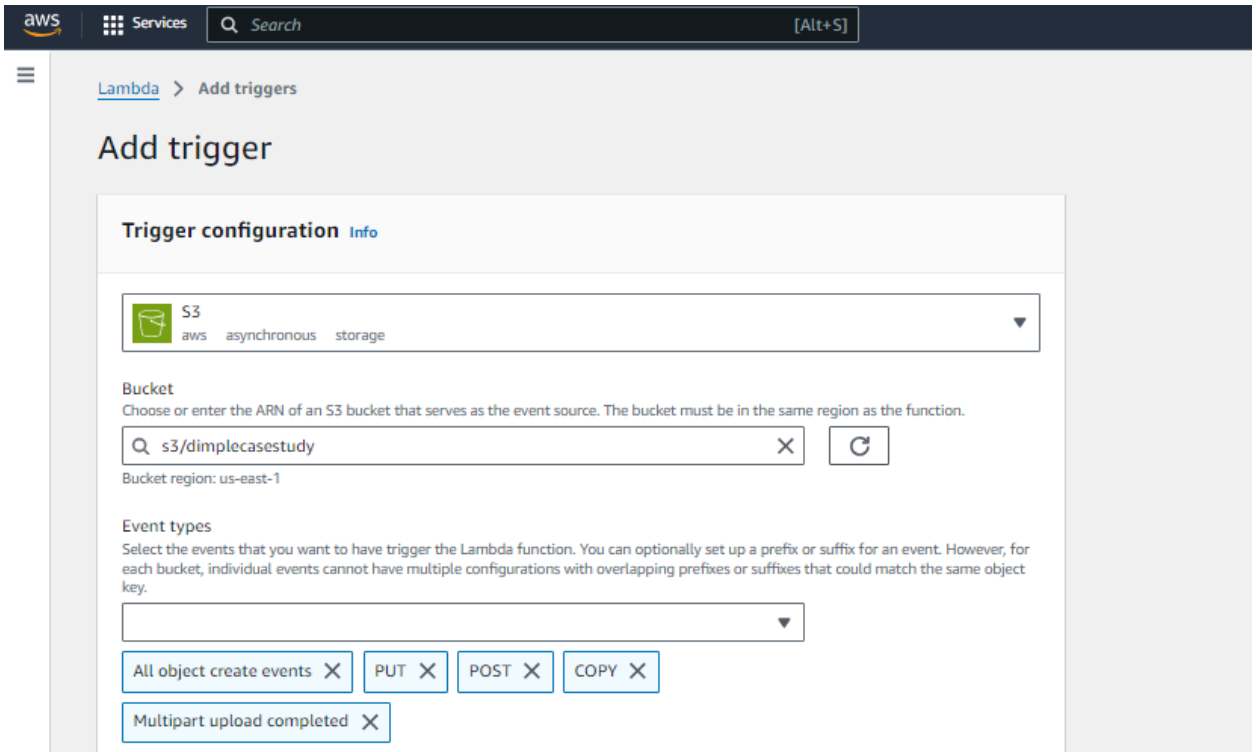
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

Add S3 Trigger to Lambda Function:

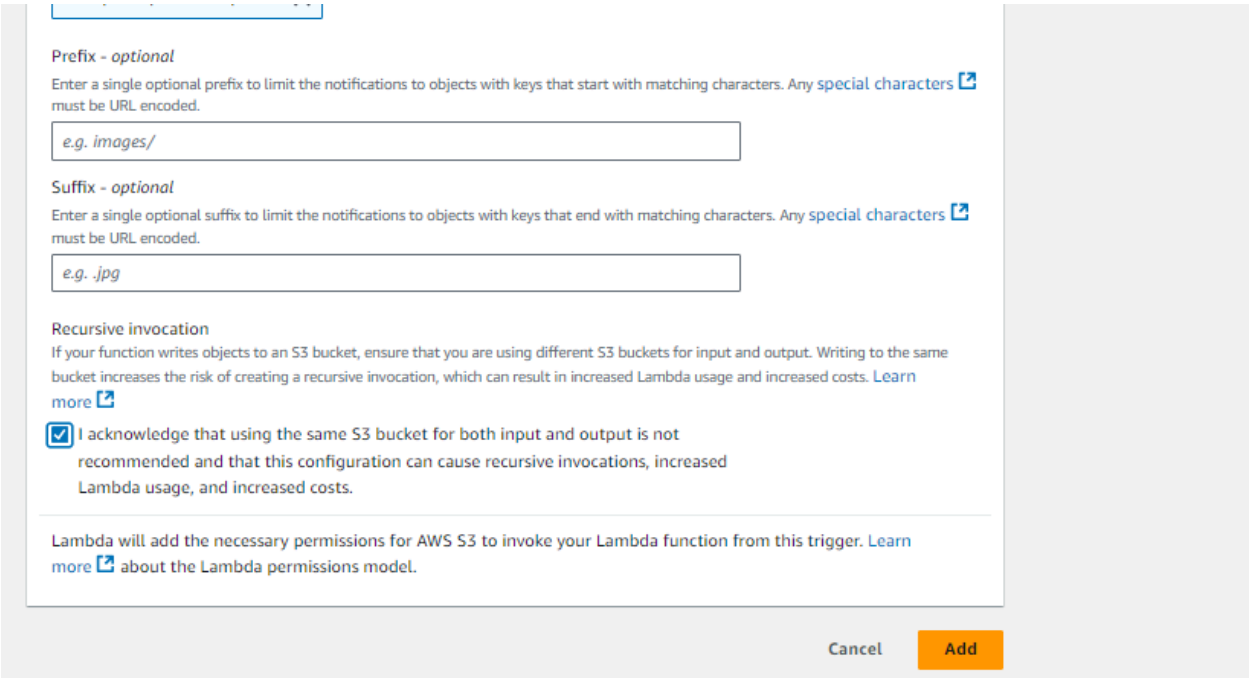
- Go back to your Lambda function.
- Click on "Add trigger" under the "Designer" section.
- Select "S3" as the trigger.
- Choose the bucket you created earlier.
- Select "All object create events".
- Click "Add".

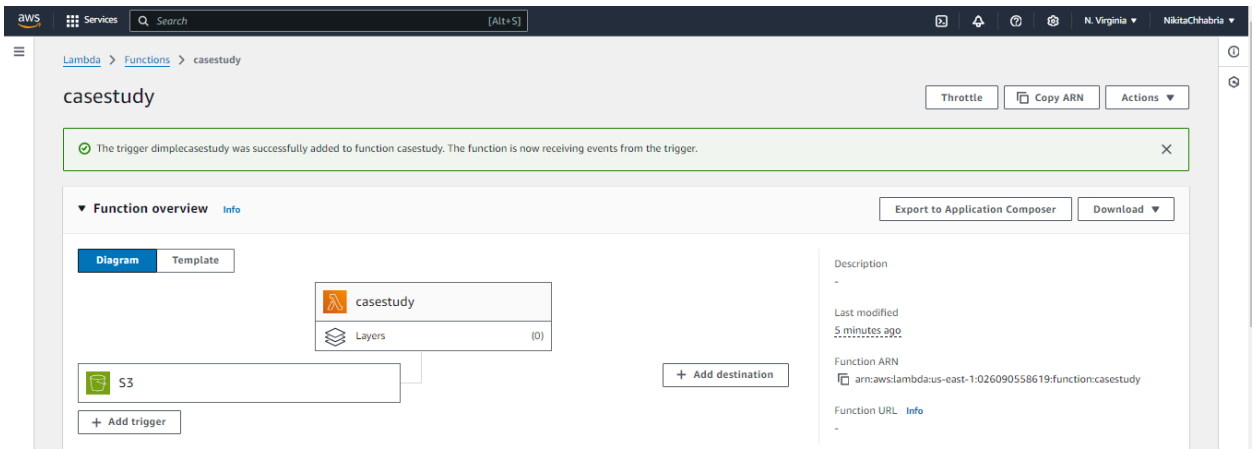




Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**





STEP IV :Upload a Sample Image to S3:

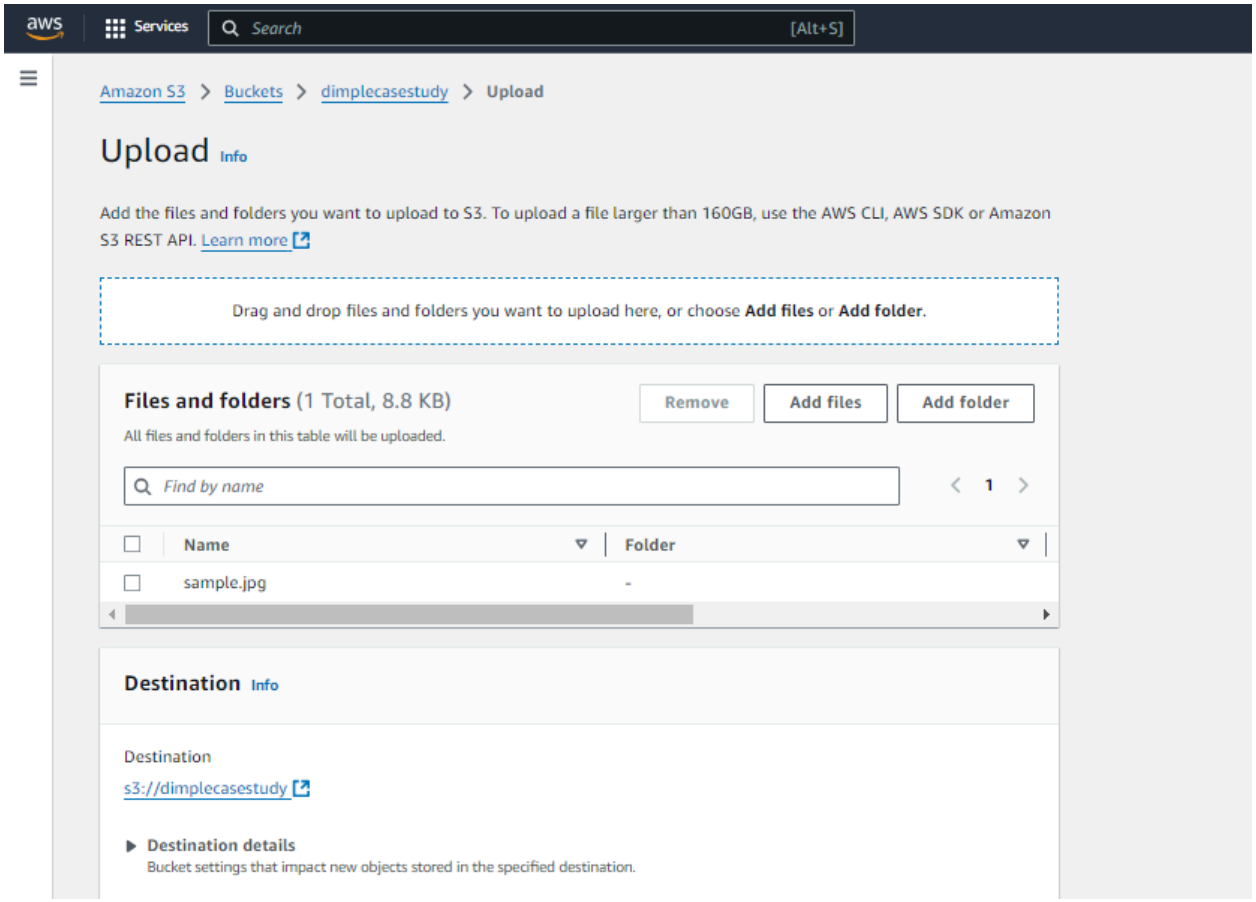
STEP 1:Go to the S3 bucket created earlier.

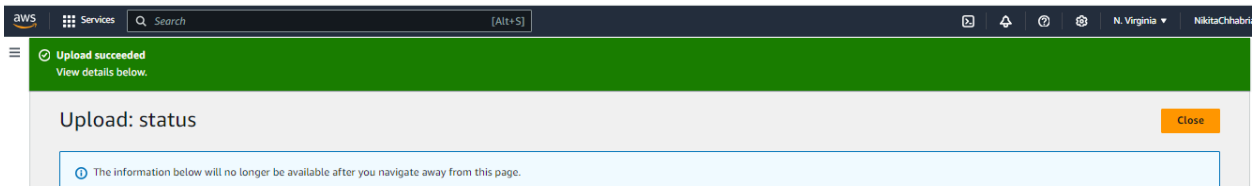
STEP 2:Click on "Upload" and select a sample image.

STEP 3: Click "Upload".

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**





Name:DIMPLE DALWANI **Year:2024-2025**

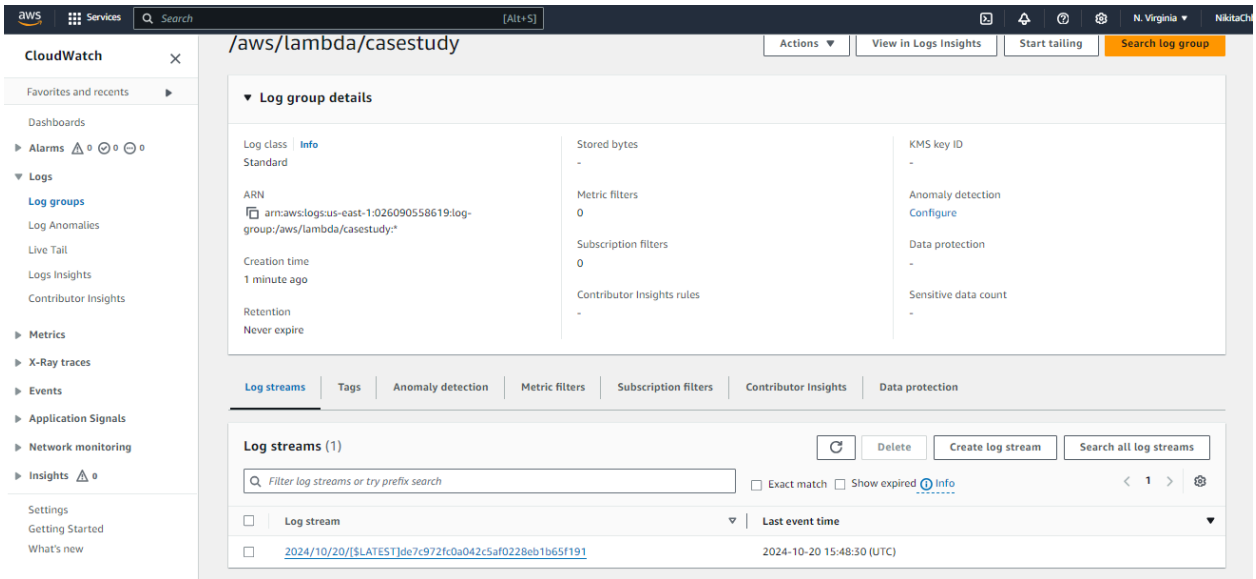
Division: D15C **Roll No: 8**

STEP V:Verify Lambda Execution:

STEP 1:Go to CloudWatch Logs by searching for "CloudWatch" in the search bar and selecting it.

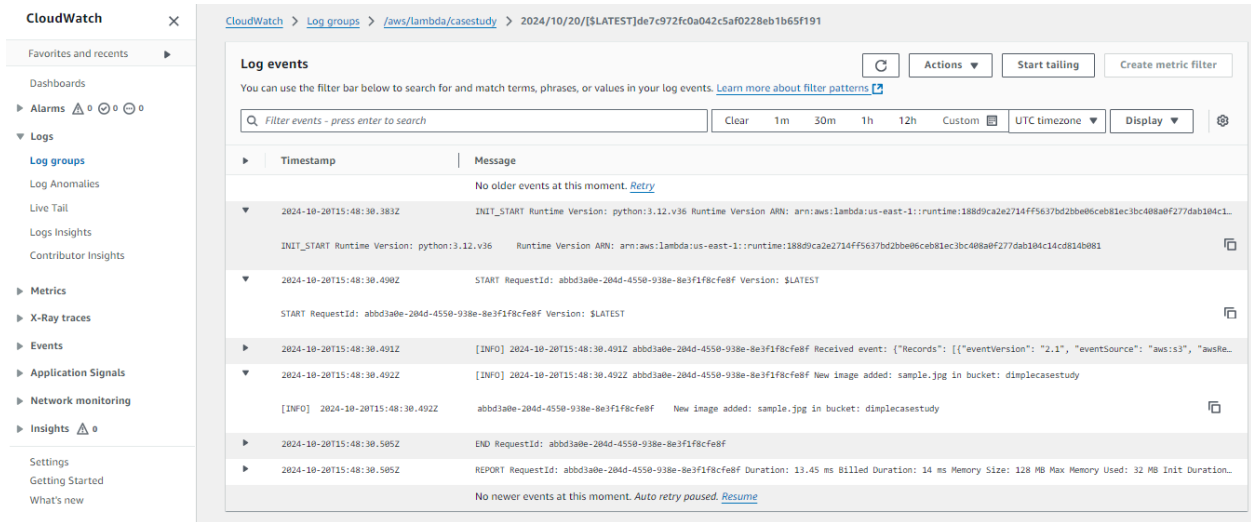
STEP 2:Click on "Logs" and find the log group for your Lambda function (e.g., `/aws/lambda/ImageProcessor`).

STEP 3:Verify that the logs show the details of the uploaded image.



Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



STEP VI:Create a Github Repository:

- Create Repositorywith name AWS-CodePipeline.
- Add the file buildspec.yml ,requirements.txt and lambda_function.py.

buildspec.yml file contains the code:

```
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.12
  build:
    commands:
      - echo "builld package"
      - zip -r lambda_function.zip .  # Packages the code
artifacts:
  files:
    - lambda_function.zip
```

requirements.txt file contains the code:

```
# Example dependency\nrequests
```

Name:DIMPLE DALWANI **Year:2024-2025**

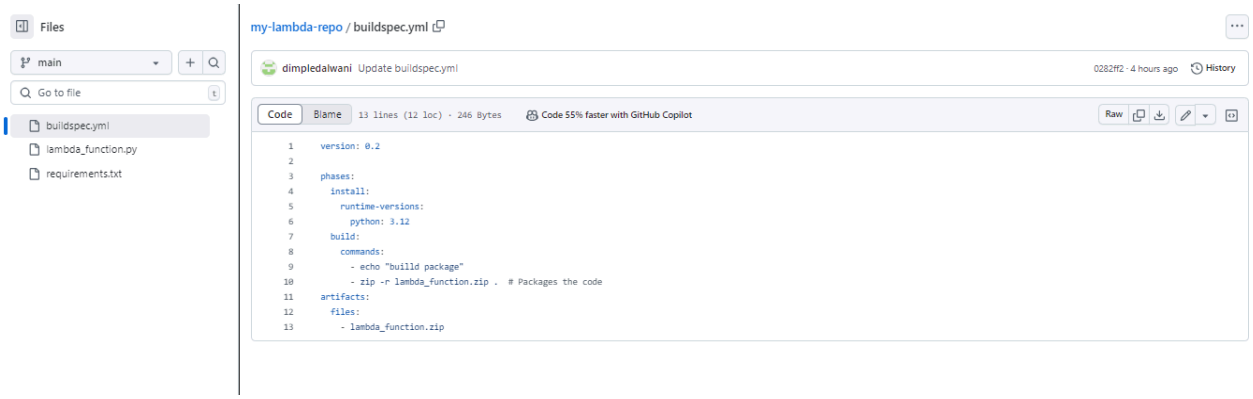
Division: D15C **Roll No: 8**

lambda_function.py file contains the code:

```
import json

import logging


logger = logging.getLogger()

logger.setLevel(logging.INFO)


def lambda_handler(event, context):

    # Log the event details

    logger.info(f"Received event: {json.dumps(event)}")

    # Extract bucket name and object key (file name)

    bucket_name = event['Records'][0]['s3']['bucket']['name']

    object_key = event['Records'][0]['s3']['object']['key']

    logger.info(f"New image added: {object_key} in bucket:

    {bucket_name}") return {

        'statusCode': 200,

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

    }
```

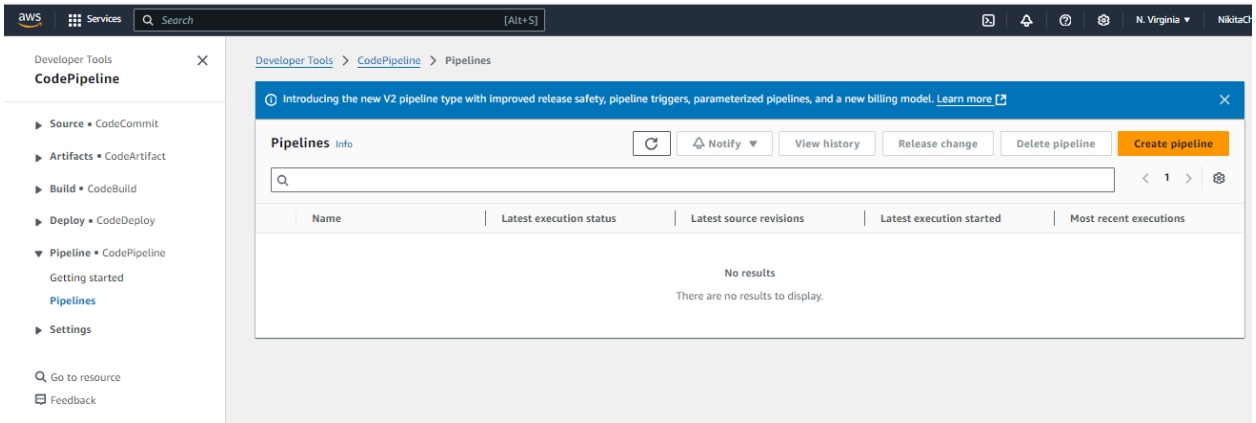
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



STEP VII:Set Up AWS CodePipeline:

STEP 1:Go to CodePipeline service and click on create pipeline.

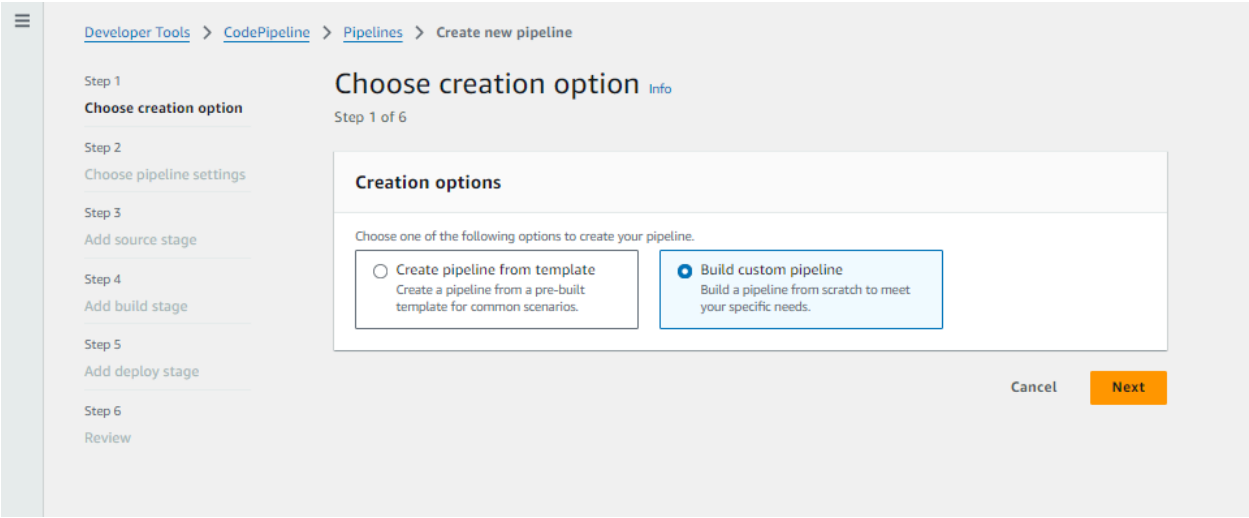


STEP 2:Choose creation option stage:

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

Choose “build custom pipeline”and then click on next.

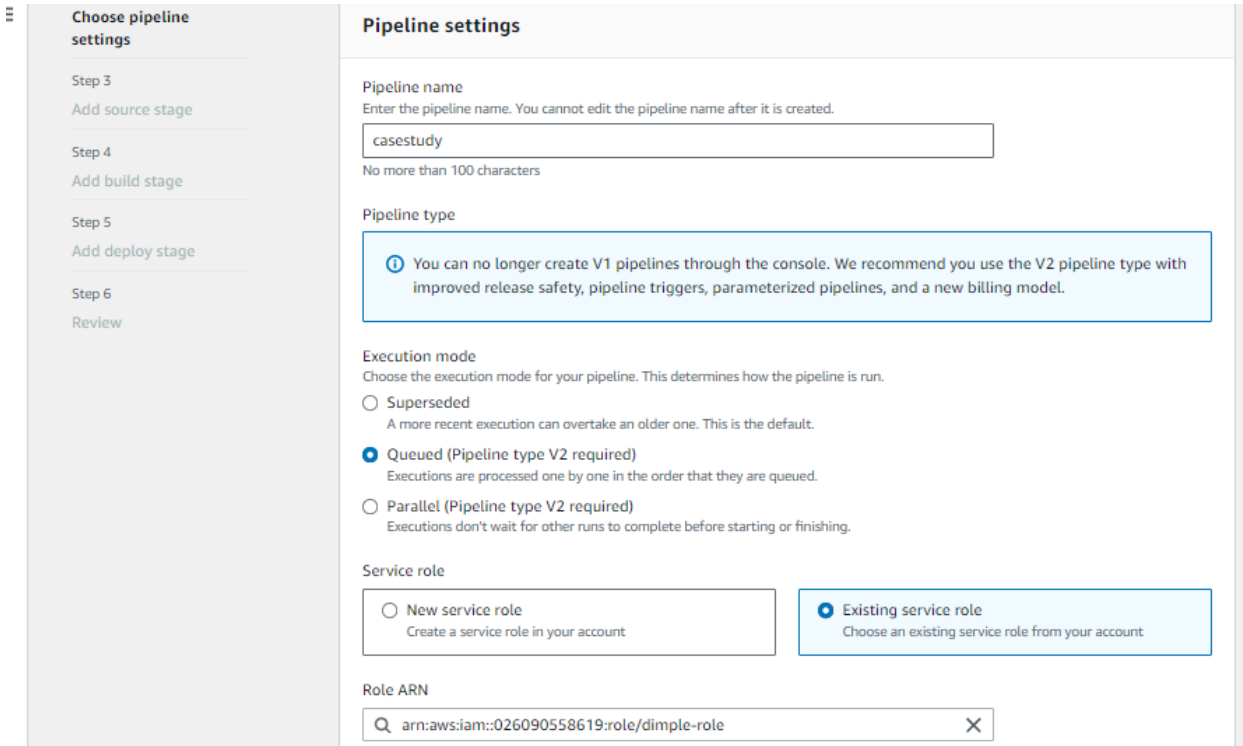


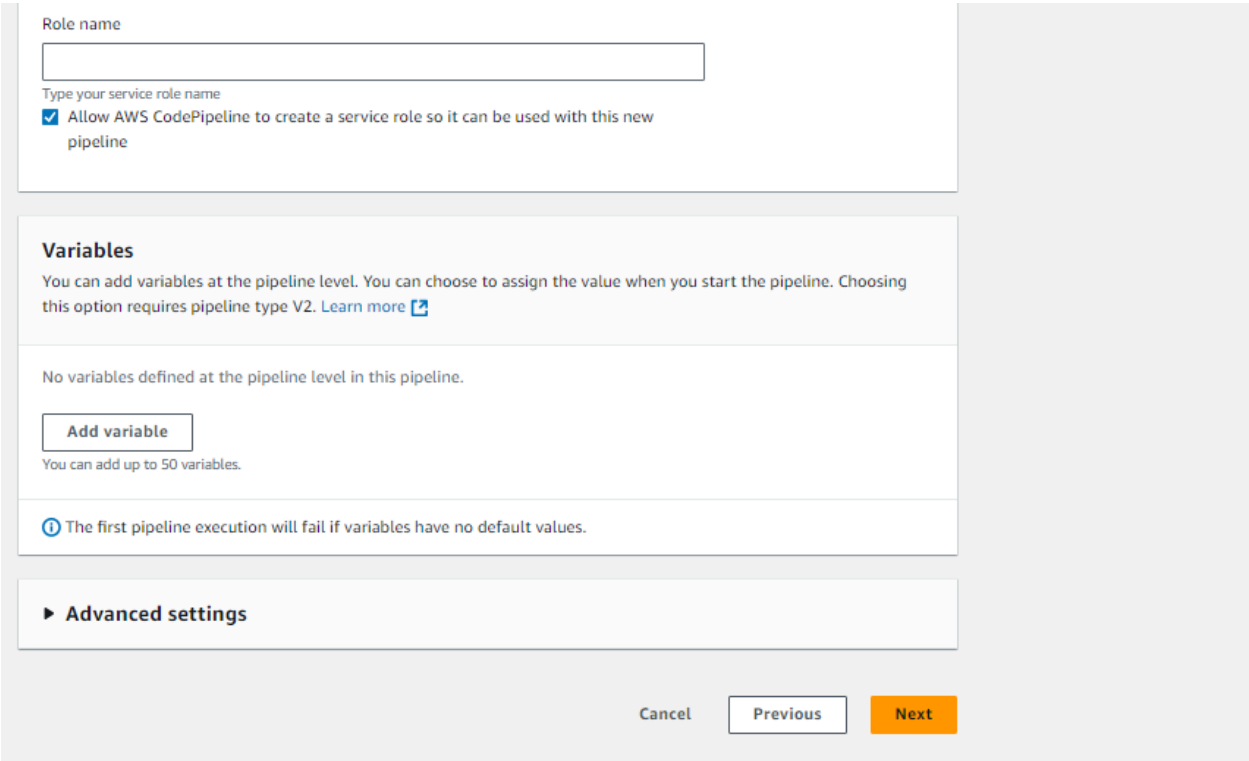
STEP 3:Choose pipeline setting stage:

Add the name of your pipeline click on existing role and then add your role arn and then click on next.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



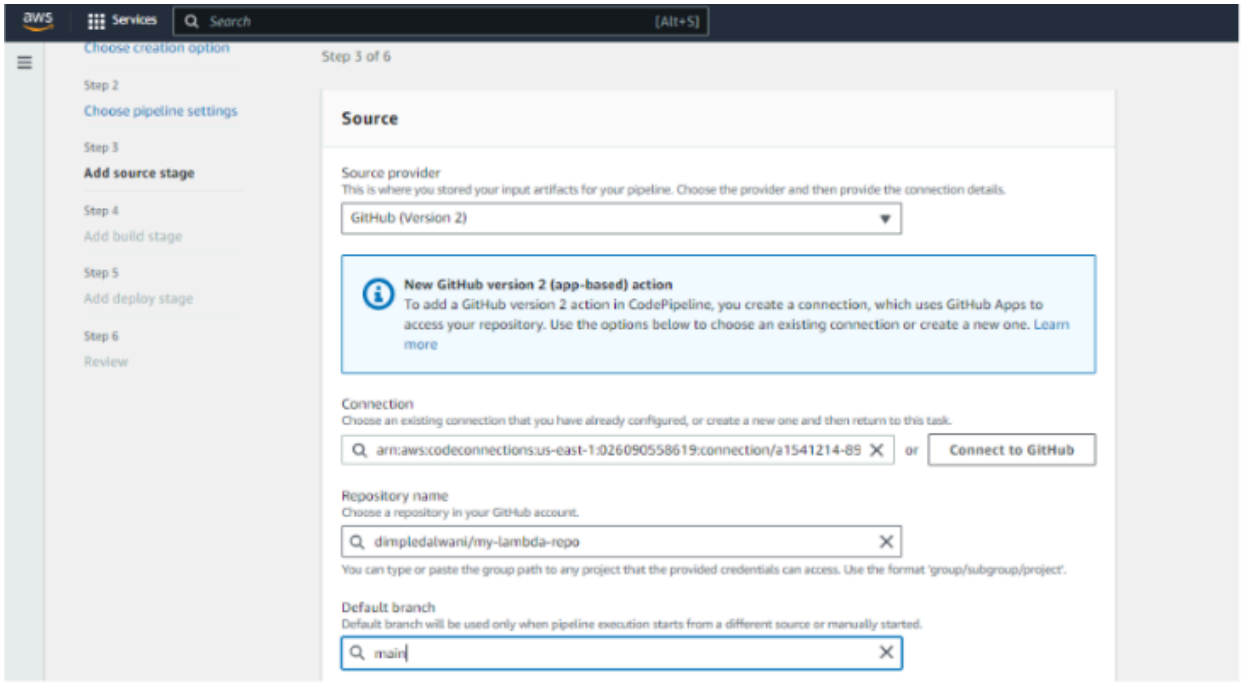


Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

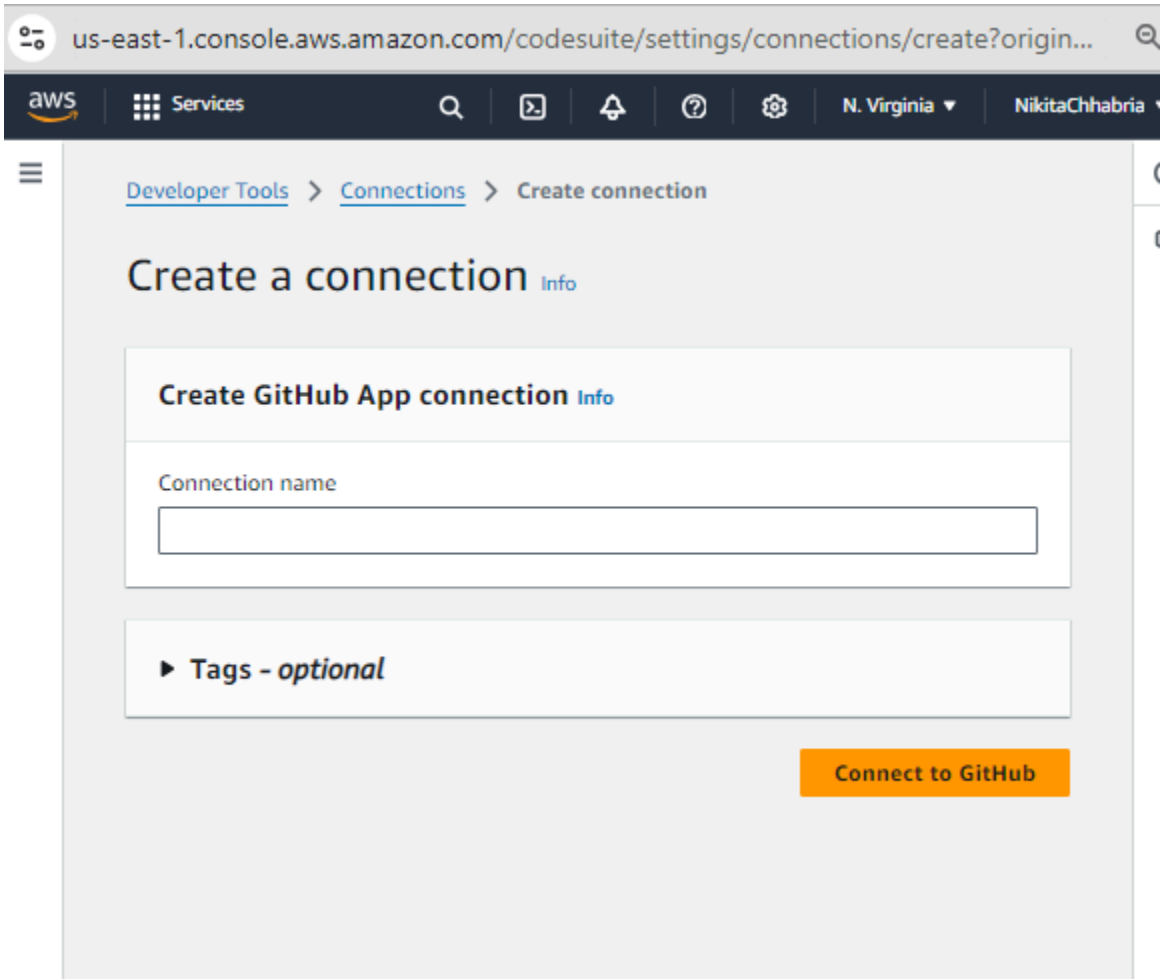
STEP 4:Add source stage:

Select “Github (Version2)” this will redirect you to another page and enter your connection name and it will connect you to your github account and then add your repository name and the default branch name and then click on next.



Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

Output artifact format
Choose the output artifact format.

- (●) **CodePipeline default**
  AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.
- ( ) **Full clone**
  AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions.

[x] Enable automatic retry on stage failure

Trigger

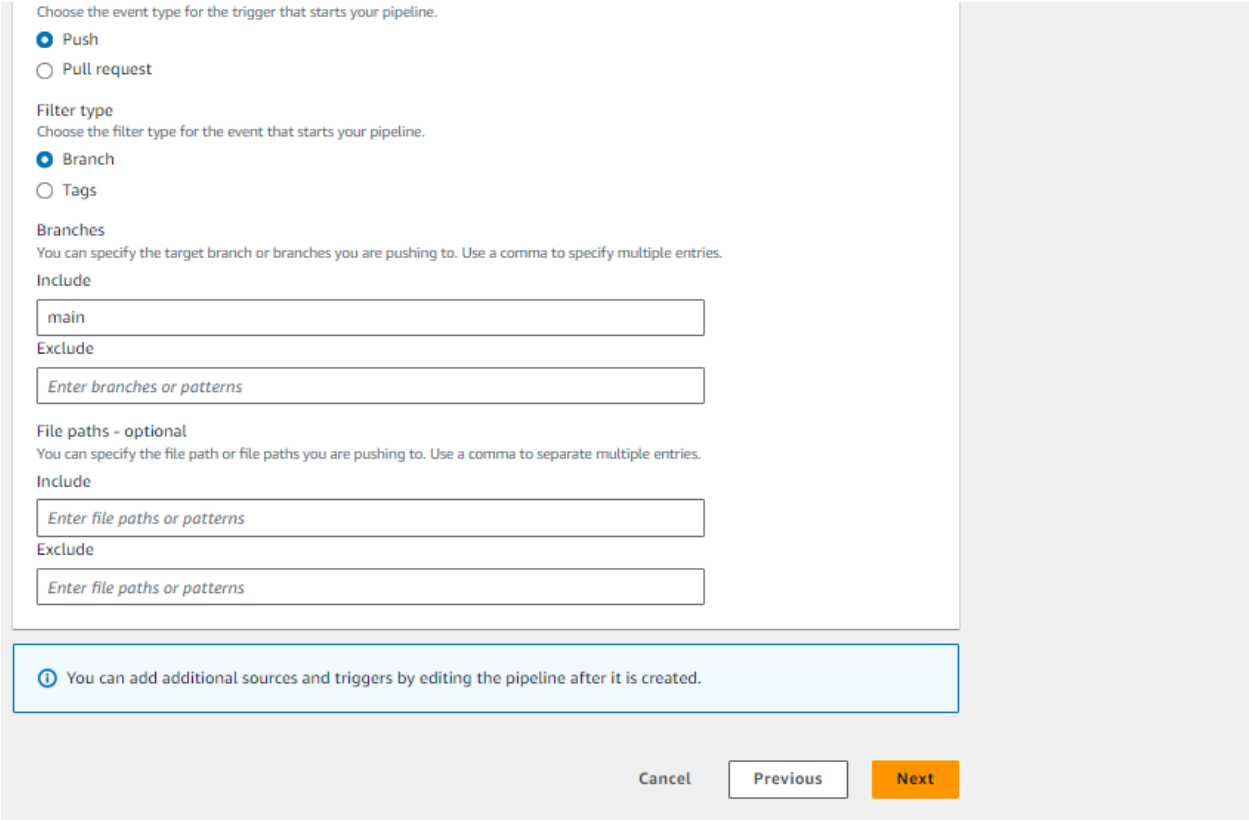
Trigger type
Choose the trigger type that starts your pipeline.

- ( ) No filter
  Starts your pipeline on any push and clones the HEAD.
- (●) Specify filter
  Starts your pipeline on a specific filter and clones the exact commit. Pipeline type V2 is required.
- ( ) Do not detect changes
  Don't automatically trigger the pipeline.

Event type
Choose the event type for the trigger that starts your pipeline.

- (●) Push
- ( ) Pull request

Filter type
Choose the filter type for the event that starts your pipeline

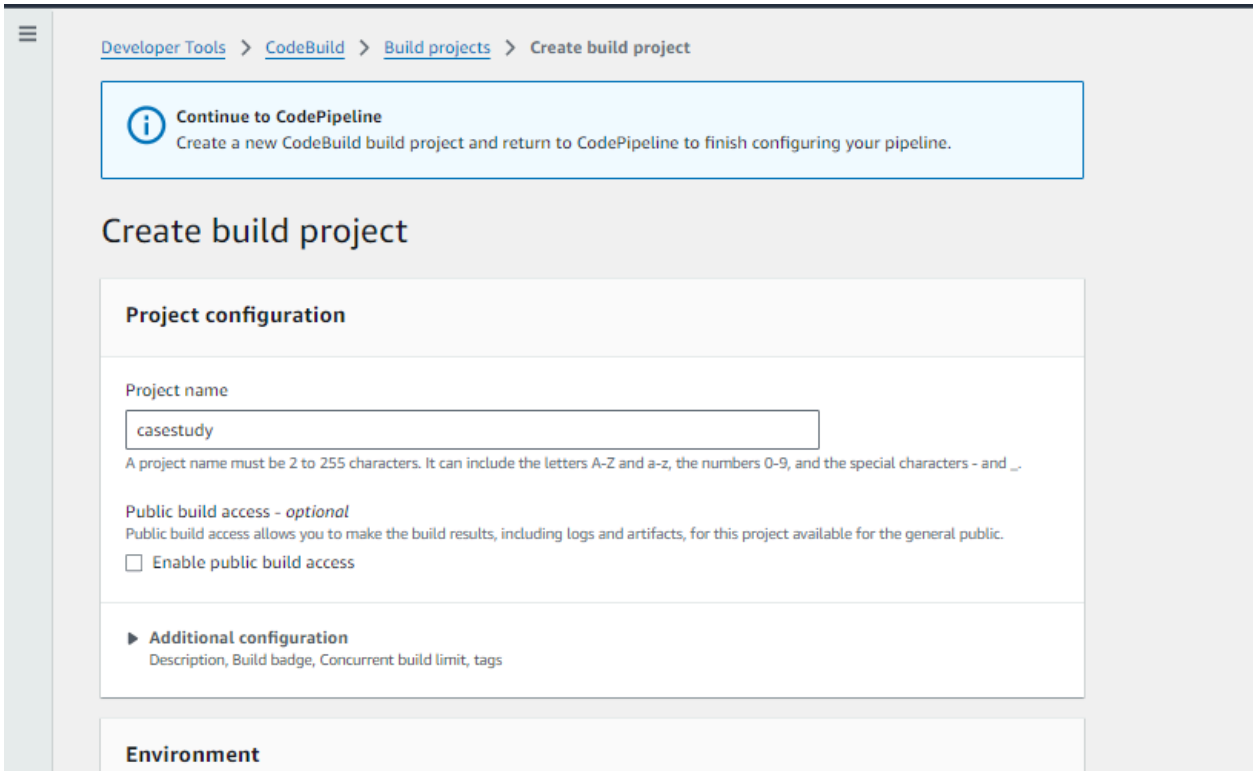


Step 5: Add Build Stage:

- Select Other build Provider.
- Select AWS CodeBuild.
    - Now Click on Create Project.
    - Project name: Image_Detector_Build
    - Enable public access.
    - Select Existing Role : AWS_Case_Study On 2 Places.
    - Buildspec: Use a buildespec file .
    - Buildspec name: buildspec.yml .
    - Keep Rest all to Default.
- Select Created Project.
- Keep Rest all to default.

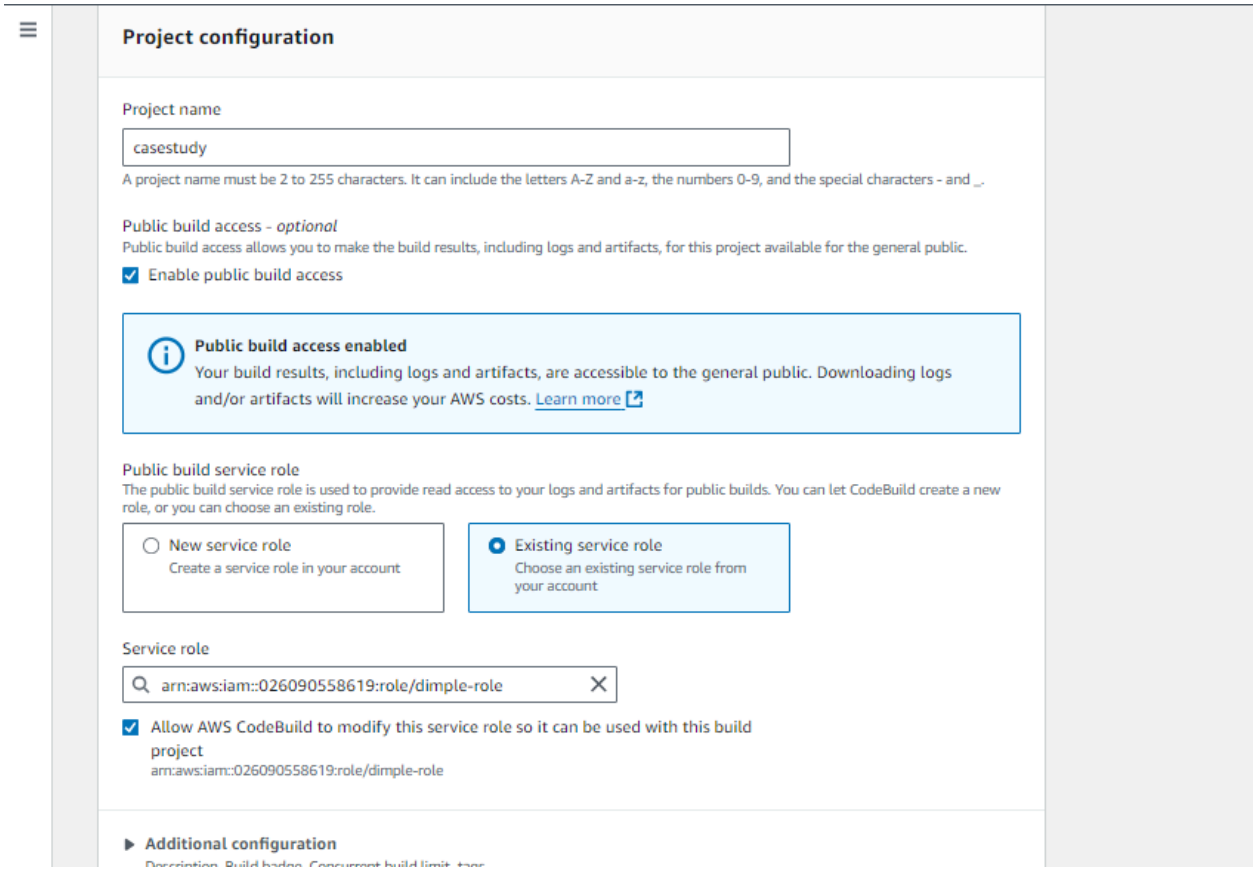
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



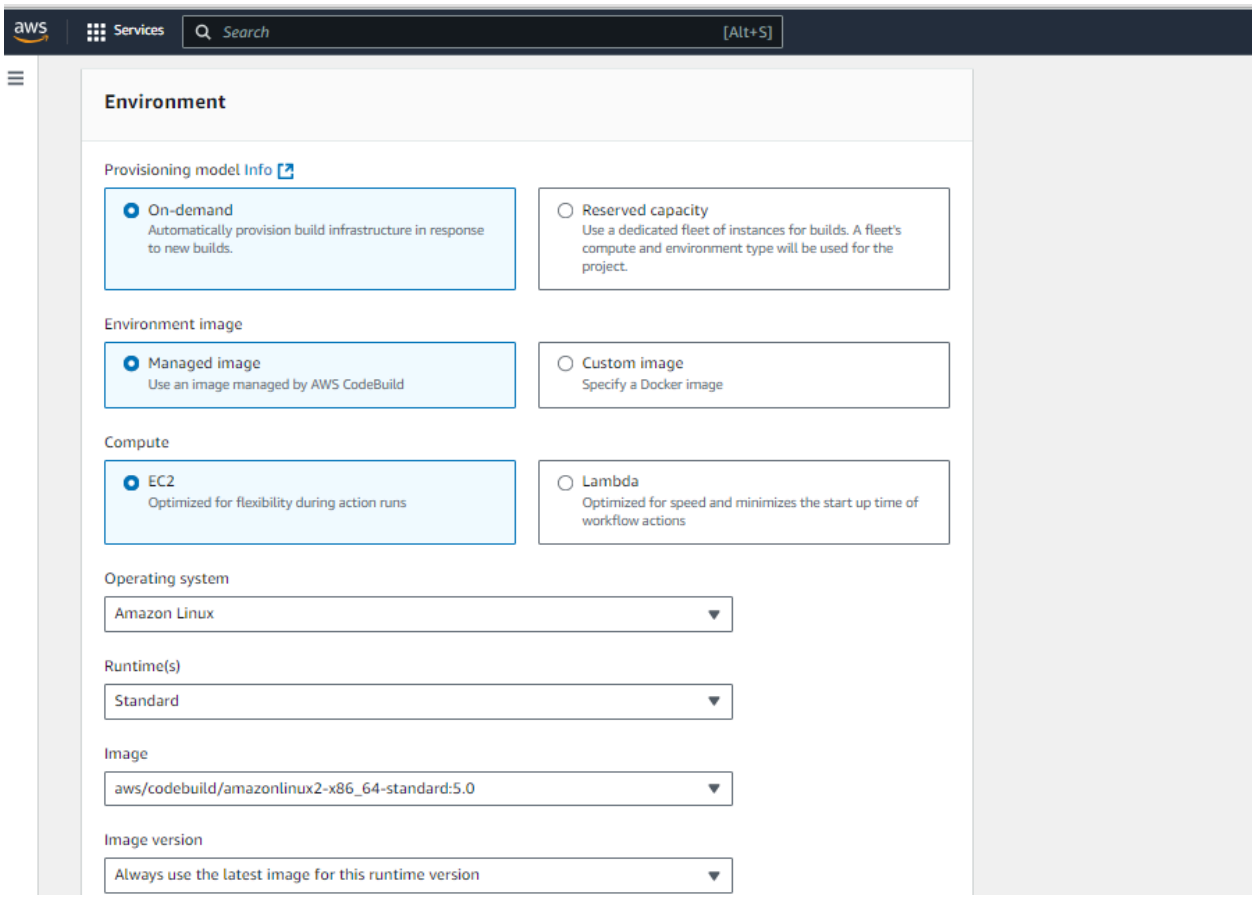
Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

aws/codebuild/amazonlinux2-x86_64-standard:5.0

Image version

Always use the latest image for this runtime version

☐ Use GPU-enhanced compute

Service role

○ New service role
Create a service role in your account

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

Role ARN

arn:aws:iam::026090558619:role/dimple-role

☑ Allow AWS CodeBuild to modify this service role so it can be used with this build project

▶ **Additional configuration**
Timeout, privileged, certificate, VPC, compute type, environment variables, file systems

aws Services Search [Alt+S]

Buildspec

Build specifications

○ 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

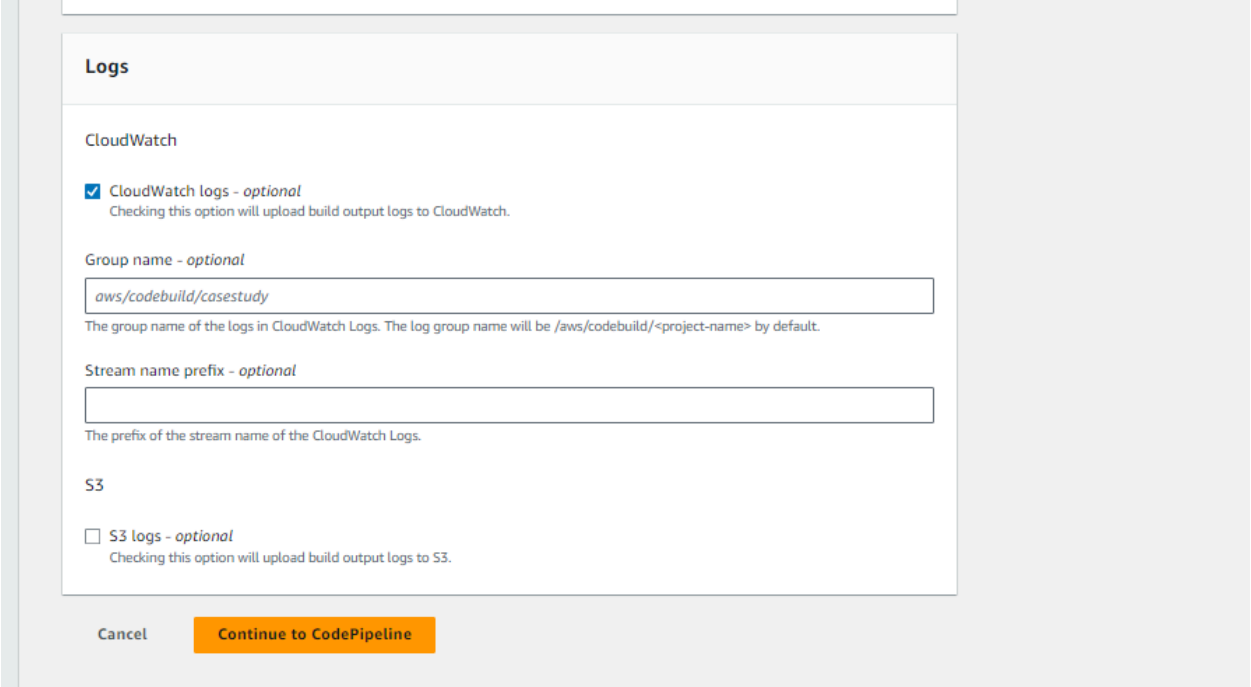
Batch configuration

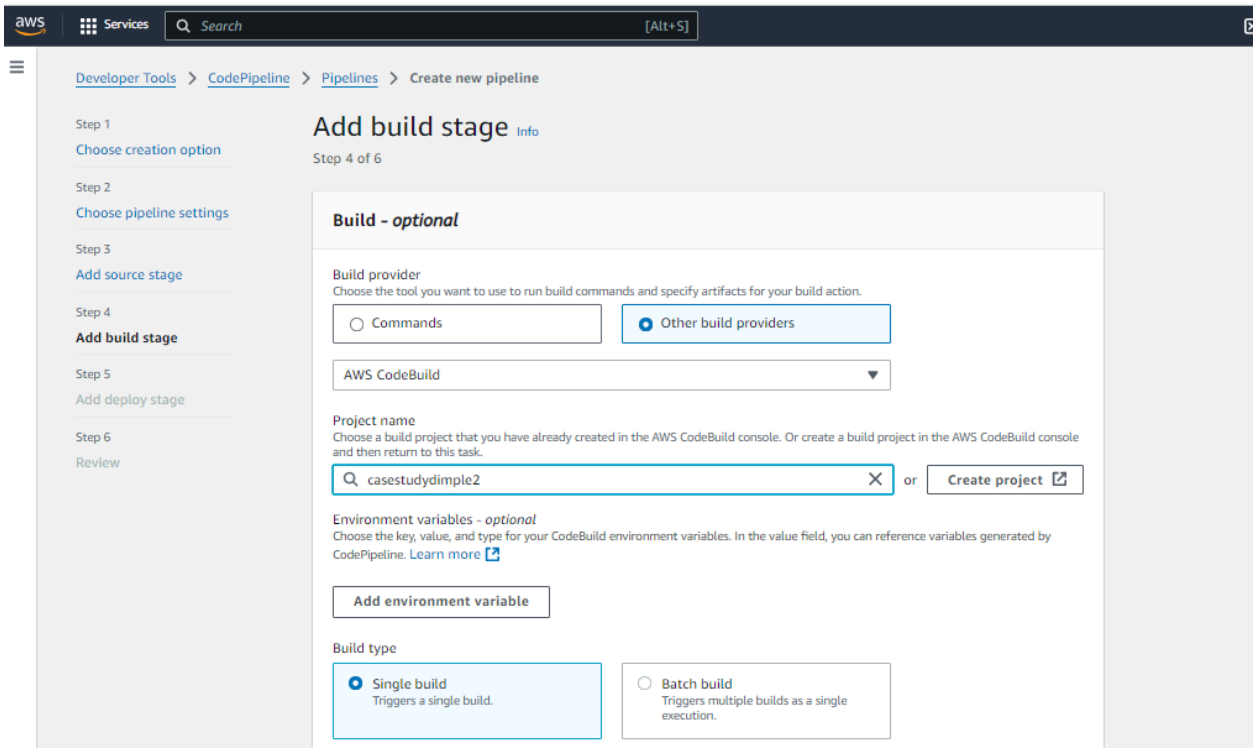
You can run a group of builds as a single execution. Batch configuration is also available in advanced option when starting build.

☐ Define batch configuration - *optional*
You can also define or override batch configuration when starting a build batch.

Name:DIMPLE DALWANI **Year:2024-2025**

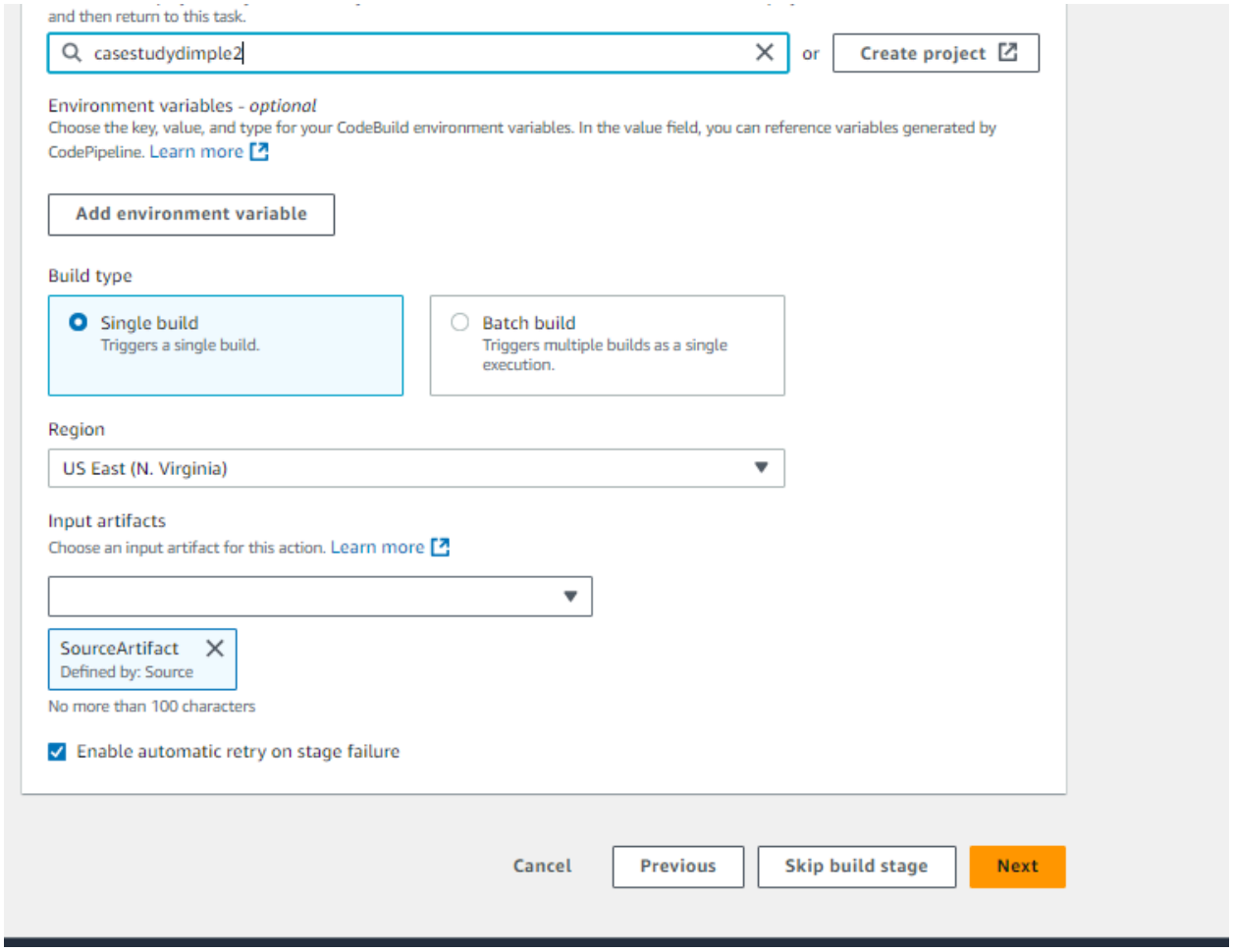
Division: D15C **Roll No: 8**





Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

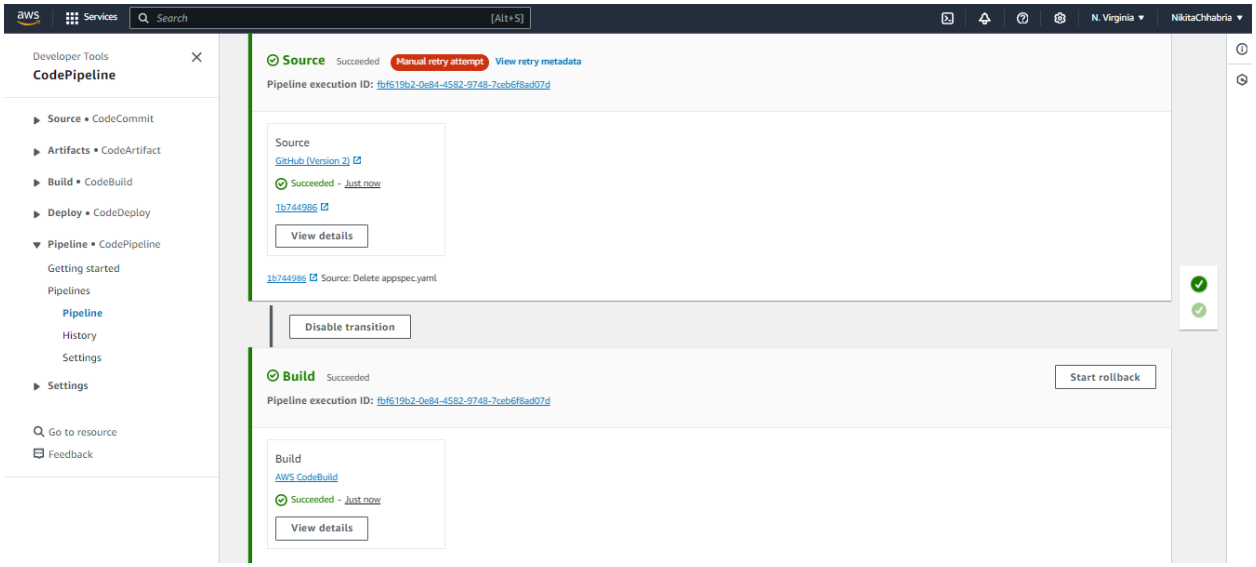


STEP 6:Add deploy stage:
Skip the deploy stage .

STEP 7:The CodePipeline is created successfully.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**



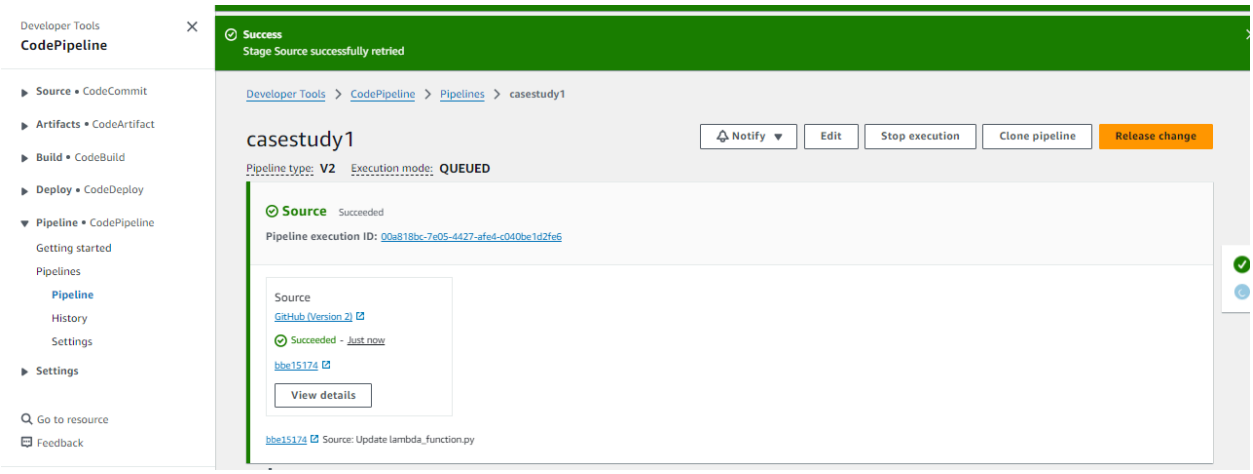
STEP VIII:Update Lambda Code and Test CodePipeline:

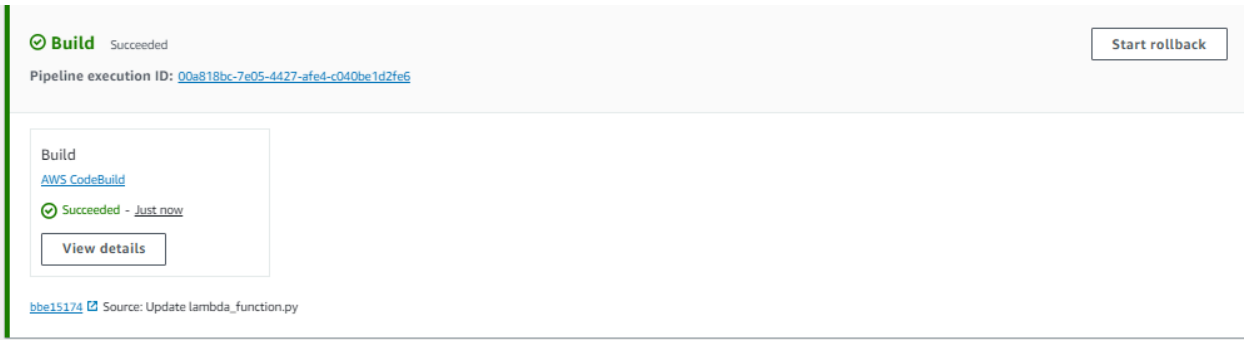
- Make a change to your Lambda function code in your repository.
- Commit and push the changes to the repository.
- Check CodePipeline to see the pipeline execution and verify that the Lambda function is updated.

```python
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Log the event details
    logger.info(f"Received event: {json.dumps(event)}")

    # Extract bucket name and object key (file name)
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    object_key = event['Records'][0]['s3']['object']['key']

    logger.info(f"New image added: {object_key} in bucket: {bucket_name}")

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

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**





Conclusion: Implementing a serverless image processing workflow with AWS Lambda, S3, and CodePipeline provides an efficient, scalable, and cost-effective solution for real-time image processing tasks. By leveraging the event-driven architecture of AWS Lambda, images uploaded to S3 buckets can be processed automatically without the need for manual intervention or server management. The integration with AWS CodePipeline ensures that any updates to the Lambda function are seamlessly deployed, enabling continuous integration and delivery. This setup is highly versatile and can be adapted for various applications such as image resizing, metadata extraction, content moderation, and thumbnail generation, making it an ideal choice for modern, cloud-based image processing needs.

GUIDELINES: Using AWS Personal for setting up your serverless image processing workflow involves creating and configuring IAM roles to ensure seamless integration and operation of various AWS services like Lambda, S3, CodeBuild, and CodePipeline. The IAM role is essential as it provides the necessary permissions for these services to interact with each other. Without the correct permissions, the workflow components will fail to trigger or deploy, causing the entire system to malfunction.

Name:DIMPLE DALWANI **Year:2024-2025**

Division: D15C **Roll No: 8**

To ensure proper access, attach specific policies to the IAM role. These policies should include AWSLambda_FullAccess, AmazonS3FullAccess, AmazonCodeBuildAdminAccess, AmazonCodePipeline_FullAccess, and CloudWatchLogsFullAccess. These permissions allow Lambda to execute code in response to events, S3 to store and retrieve images, CodeBuild to build and package the Lambda function, and CodePipeline to automate the deployment process. Additionally, the CloudWatchLogsFullAccess policy enables logging and monitoring of the functions and processes involved.

It is also crucial to configure the trust relationship for the IAM role to include the services codepipeline.amazonaws.com and codebuild.amazonaws.com. This configuration allows CodePipeline and CodeBuild to assume the role and perform the necessary actions within your AWS environment. By setting up these permissions and trust relationships correctly, you ensure that your serverless workflow operates smoothly and efficiently, with all components having the access they need to function properly.