Cloud Computing Project

Presented by:

Geetha Krishna Venkatesh Maroju

Satish kumar Buravelli

Sai Nandu Posina

Contents

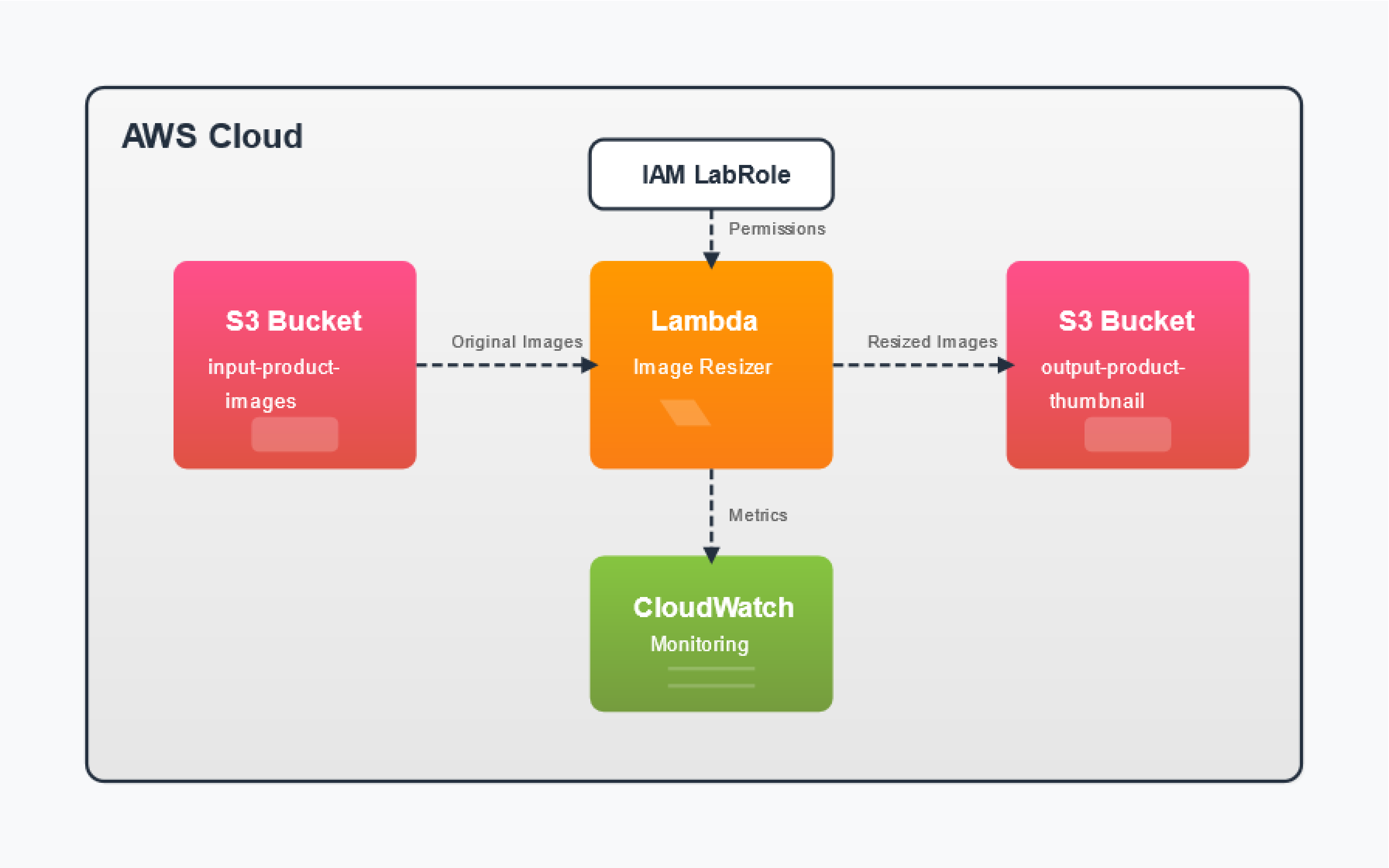
- Project Aim and Requirements
- Architecture Overview
- Implementation Steps
- Monitoring and Testing
- Conclusion

Project Aim and Requirements

- To develop and deploy a scalable image processing application using AWS Lambda
- Primary function: Automatic thumbnail generation from uploaded images
- Key requirements:
  - AWS Lambda service
  - Amazon S3 for image storage
  - CloudWatch for monitoring
  - Predefined IAM role (LabRole)
  - Appropriate trigger configurations

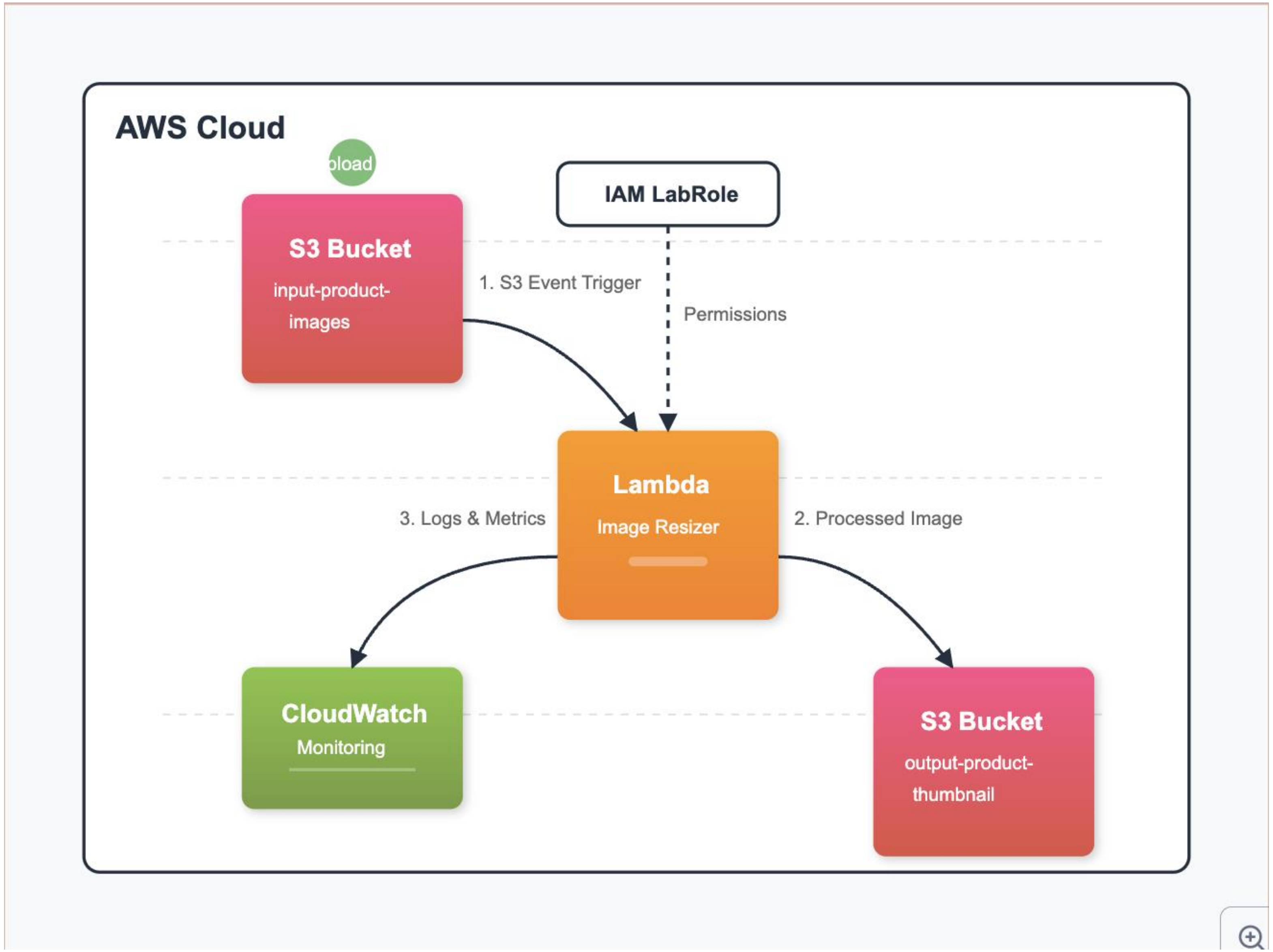
Architecture Overview

- The system architecture is designed to efficiently handle image resizing workflows using AWS services. It consists of a source S3 bucket named (input-product-images), which holds the original images, and a destination S3 bucket named(output-product-thumbnail) where the resized images will be stored. Image processing is handled by an AWS Lambda function that works with a set of permissions provided by the LabRole IAM role. This ensures secure and controlled access to the necessary resources. CloudWatch is integrated for monitoring and tracking performance metrics, hence allowing for efficient oversight of the system's operations.


AWS Cloud
IAM LabRole
Permissions
S3 Bucket
input-product-images
Original Images
Lambda
Image Resizer
Resized Images
S3 Bucket
output-product-thumbnail
Metrics
CloudWatch
Monitoring

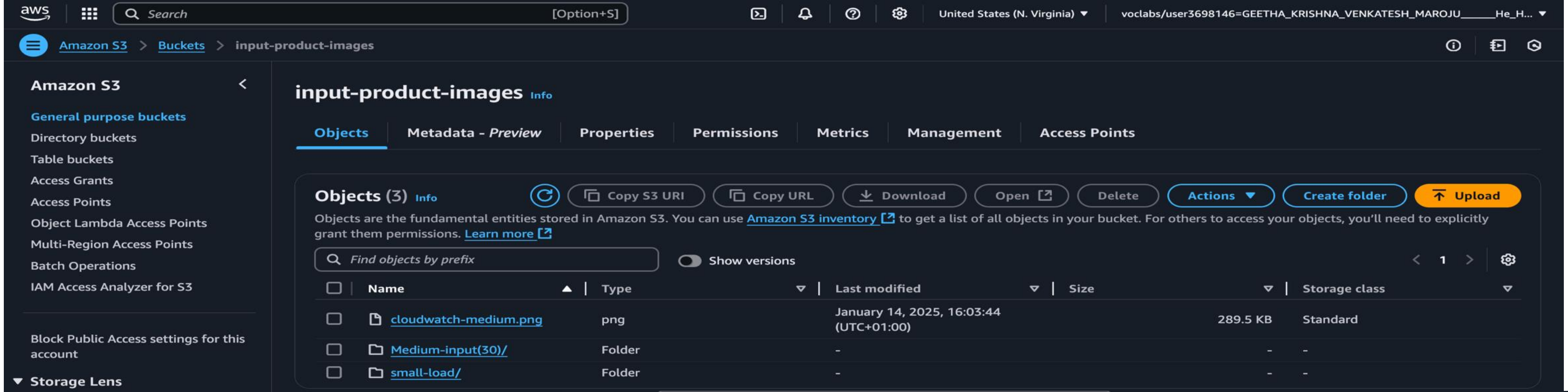
Architecture Overview

- This workflow triggers right from uploading an image into the source S3 bucket. The S3 event triggers and calls the Lambda function. The LabRole permissions enable the Lambda function to process the image into the required dimensions and place it into the destination S3 bucket. During this time, CloudWatch would have recorded the performance metrics and logs for system performance.


AWS Cloud
pload
IAM LabRole
S3 Bucket
input-product-images
1. S3 Event Trigger
Permissions
Lambda
Image Resizer
3. Logs & Metrics
2. Processed Image
CloudWatch
Monitoring
S3 Bucket
output-product-thumbnail

Implementation Steps
1. S3 Bucket Setup

- Create two S3 buckets-one source bucket is used for uploading the images to be resized and another one used as the target bucket, to store the output/ resized photos. Give it a name in accordance with naming conventions at AWS. In my case I named the source bucket as ”input-product-images” and the target bucket is named as “output-product-thumbnail”.
- Next, set up stream permissions on both buckets. Set adequate access policies to enable only the actions of the Lambda function to read in the source bucket and write to the destination bucket. Permission control should be highly specific through bucket policies and ACLs.

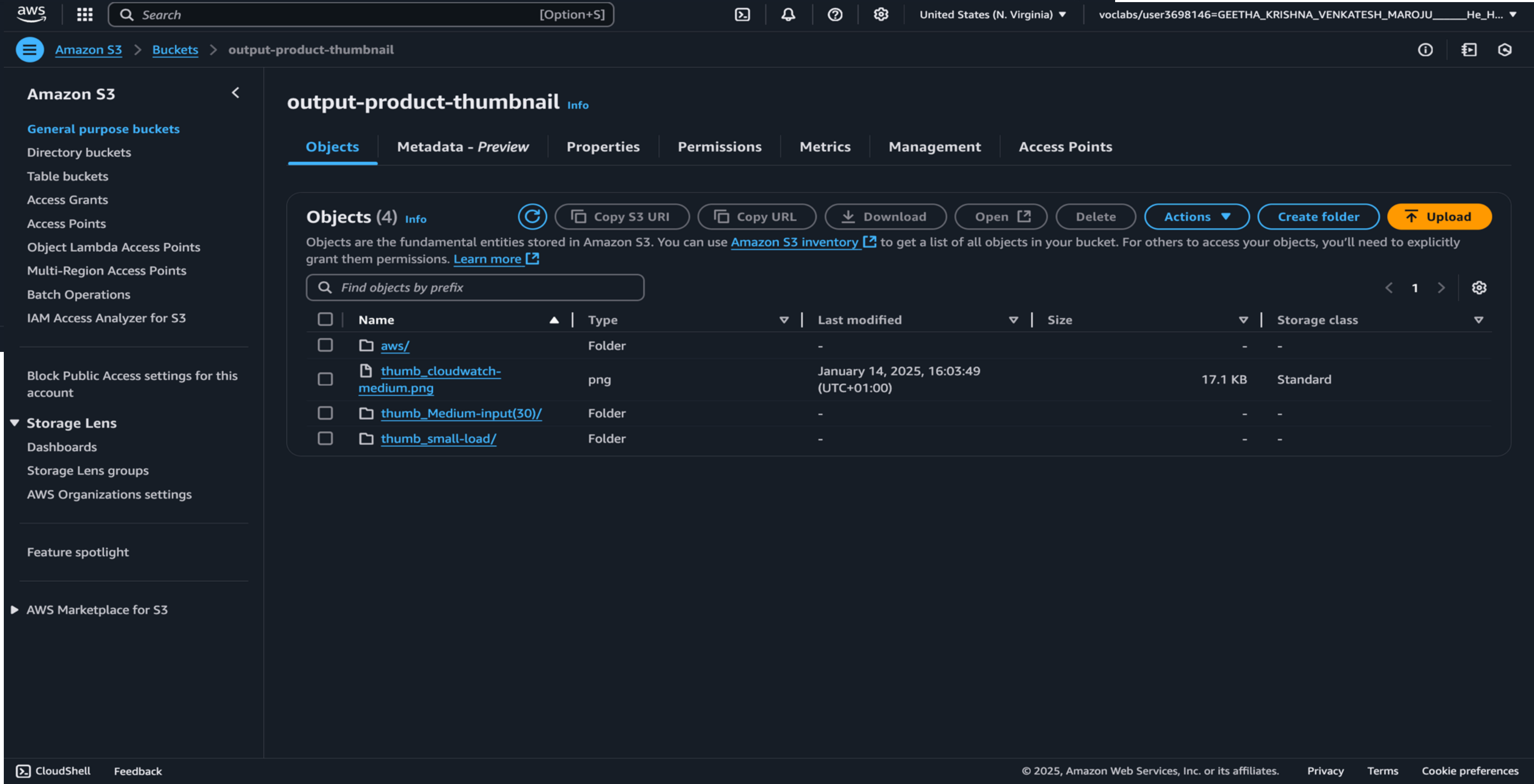


input-product-images Info

Objects (3) 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
□	cloudwatch-medium.png	png	January 14, 2025, 16:03:44 (UTC+01:00)	289.5 KB	Standard
□	Medium-input(30)/	Folder	-	-	-
□	small-load/	Folder	-	-	-



output-product-thumbnail Info

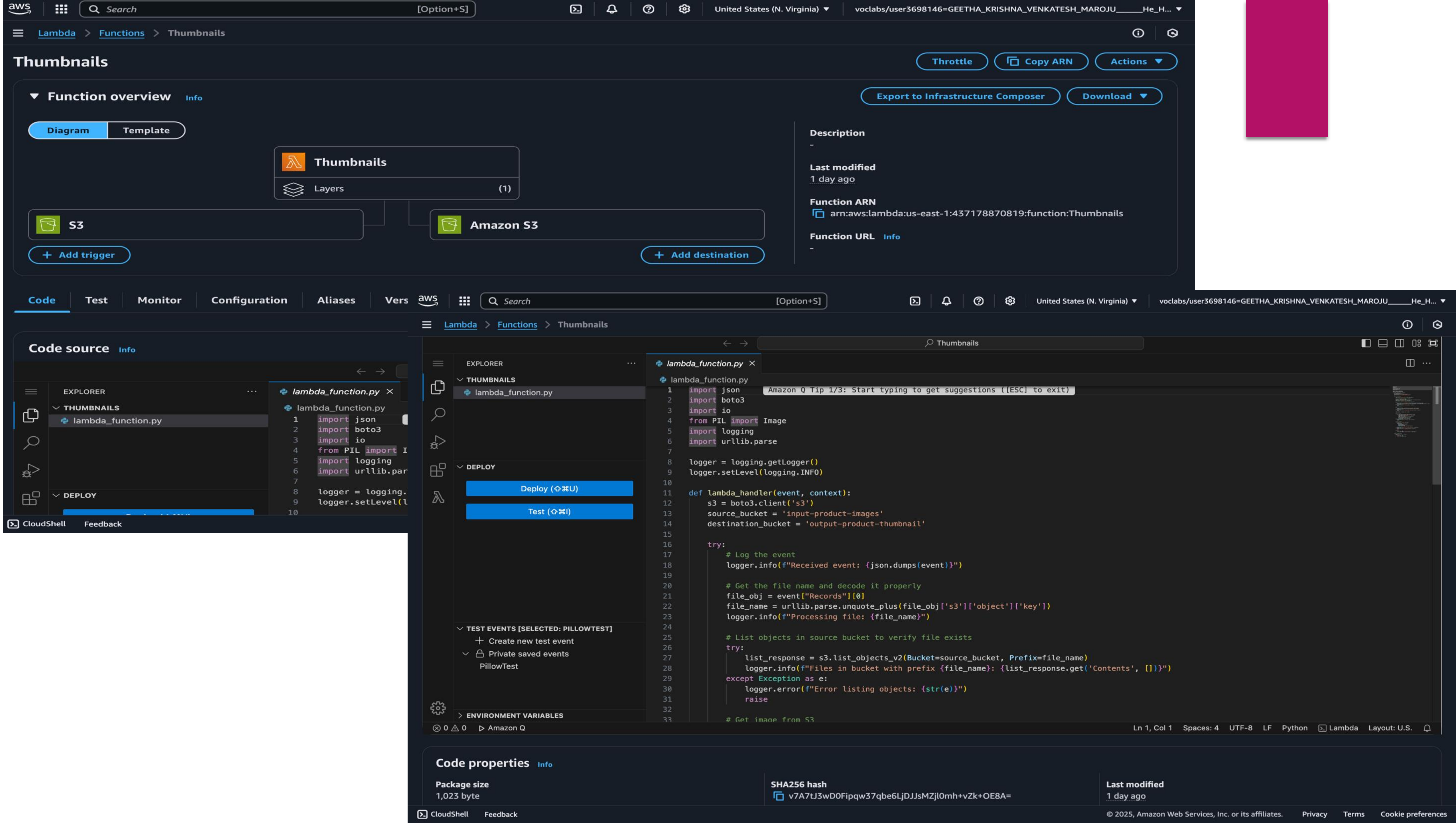
Objects (4) Info

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
□	aws/	Folder	-	-	-
□	thumb_cloudwatch-medium.png	png	January 14, 2025, 16:03:49 (UTC+01:00)	17.1 KB	Standard
□	thumb_Medium-input(30)/	Folder	-	-	-
□	thumb_small-load/	Folder	-	-	-

Implementation Steps
2. Lambda Function Creation

- The next step is to create a new AWS Lambda function, for this we need to search for lambda in AWS console then after we created the new lambda function named as the “Thumbnails” and it uses the runtime as “Python 3.9” and assigning it the “LabRole” IAM role during setup.This will grant the required permission to work with S3 and resize images using this function. The memory setting and timeout are set to have better performance. As for the image resizing, it can be done in a programming language of choice, Python in this case, which will use its libraries to deal with images. Environment variables are defined for the source and destination bucket names to simplify the configuration.



Thumbnails

Description
-

Last modified
1 day ago

Function ARN
arn:aws:lambda:us-east-1:437178870819:function:Thumbnails

Function URL Info
-

lambda_function.py

```
import json          Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
import boto3
import io
from PIL import Image
import logging
import urllib.parse

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

def lambda_handler(event, context):
    s3 = boto3.client('s3')
    source_bucket = 'input-product-images'
    destination_bucket = 'output-product-thumbnail'

    try:
        # Log the event
        logger.info(f"Received event: {json.dumps(event)}")

        # Get the file name and decode it properly
        file_obj = event["Records"][0]
        file_name = urllib.parse.unquote_plus(file_obj['s3']['object']['key'])
        logger.info(f"Processing file: {file_name}")

        # List objects in source bucket to verify file exists
        try:
            list_response = s3.list_objects_v2(Bucket=source_bucket, Prefix=file_name)
            logger.info(f"Files in bucket with prefix {file_name}: {list_response.get('Contents', [])}")
        except Exception as e:
            logger.error(f"Error listing objects: {str(e)}")
            raise

        # Get image from S3
```

Code properties Info

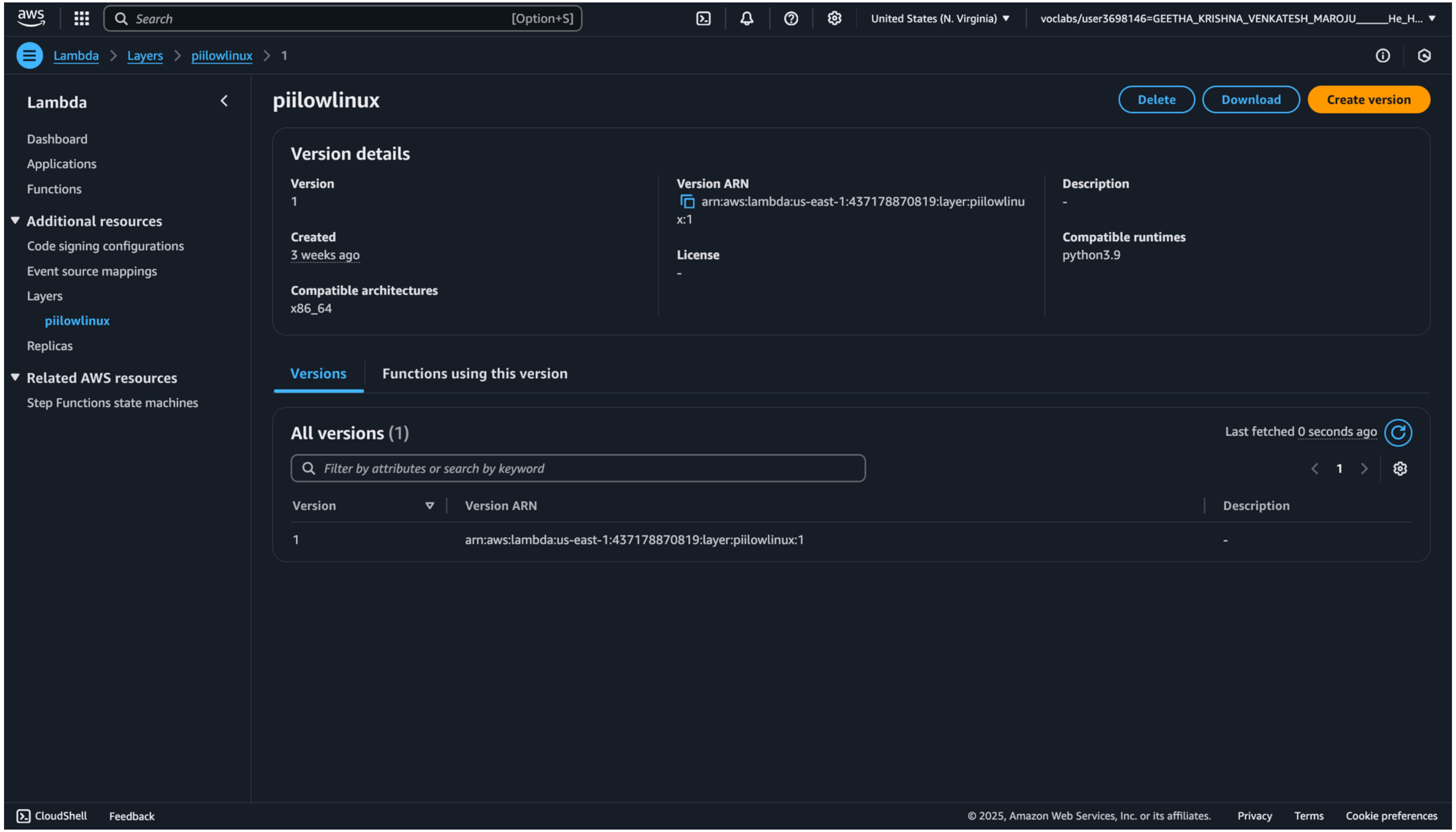
Package size
1,023 byte

SHA256 hash
v7A7tJ3wD0Fipqw37qbe6LjDJJsMZjl0mh+vZk+OE8A=

Last modified
1 day ago

2a. Lambda Layer Creation

- In order to enable Python to process images, a Lambda layer was created with the Pillow library-a library not natively available within the default Lambda runtime. First, it was necessary to create a new Lambda layer and upload a ZIP file containing the Pillow package. In that respect, the layer was set to Python 3.9 to ensure smooth integration. The layer was attached to the Lambda function once it had been created with the dependencies required for the resizing and processing of images. This ensured that this function was loaded for the process in full capability.


aws
Search [Option+S]
United States (N. Virginia) ▼
voclabs/user3698146=GEETHA_KRISHNA_VENKATESH_MAROJU_____He_H... ▼
Lambda > Layers > piilowlinux > 1
Lambda
Dashboard
Applications
Functions
▼ Additional resources
Code signing configurations
Event source mappings
Layers
piilowlinux
Replicas
▼ Related AWS resources
Step Functions state machines
Delete
Download
Create version
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

piilowlinux

Version details

Version
1

Created
3 weeks ago

Compatible architectures
x86_64

Version ARN
arn:aws:lambda:us-east-1:437178870819:layer:piilowlinux:1

License
-

Description
-

Compatible runtimes
python3.9

Versions | Functions using this version

All versions (1)

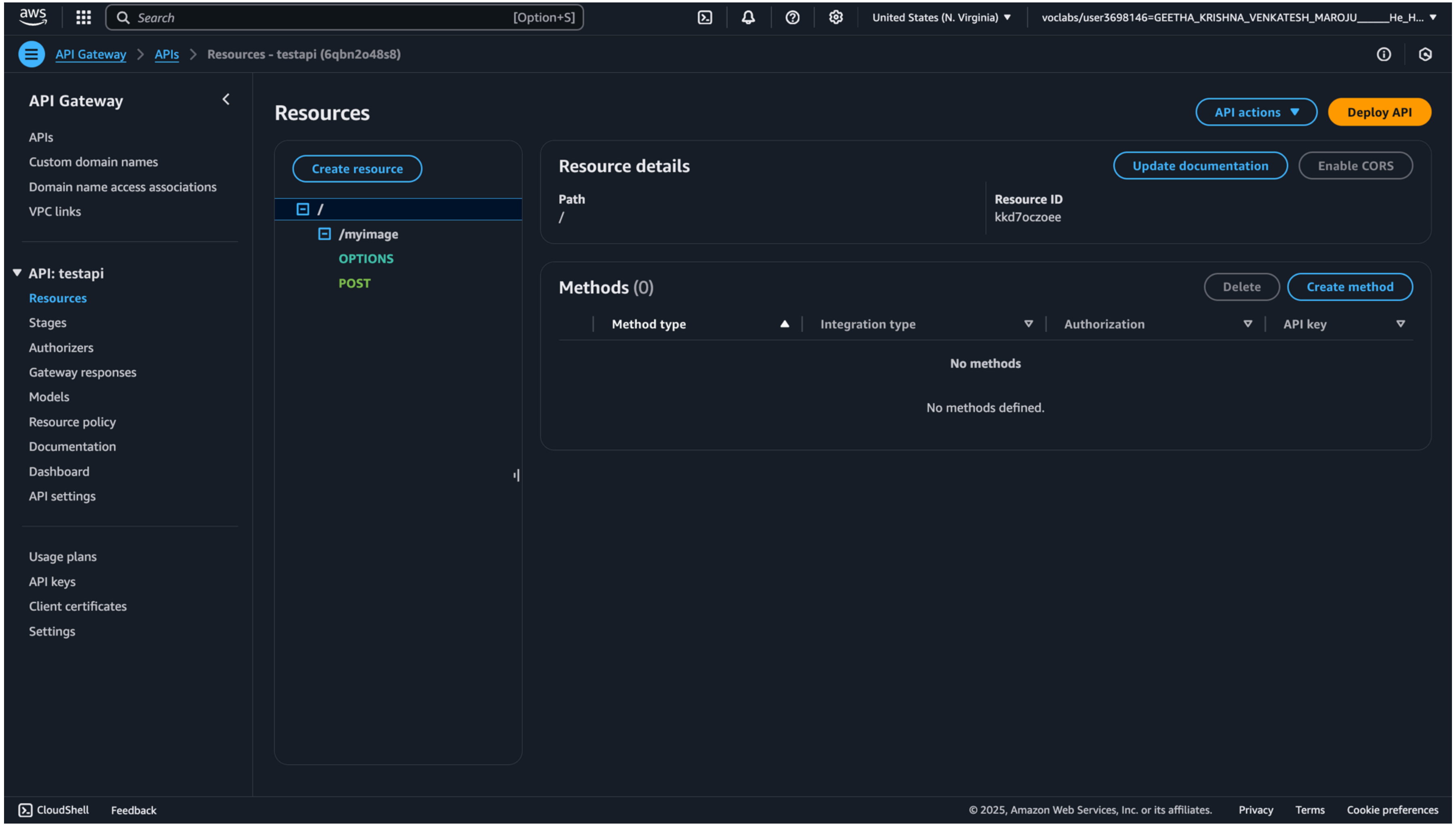
Last fetched 0 seconds ago

Filter by attributes or search by keyword

Version	Version ARN	Description
1	arn:aws:lambda:us-east-1:437178870819:layer:piilowlinux:1	-

2b. API Gateway Setup

- API Gateway was implemented to expose the Lambda function via HTTP so that users could start an image processing job via a secured endpoint. A new REST API was created and methods like GET and POST were configured to handle the requests of image processing. API Gateway was then integrated with the Lambda function, and appropriate request and response mappings were set up to ensure smooth communication between components. API deployment was finally performed to a targeted stage called 'dev1', through which the flow was triggered. With this setup, interaction with the image processing system via HTTP is solid and secure.



Path
/

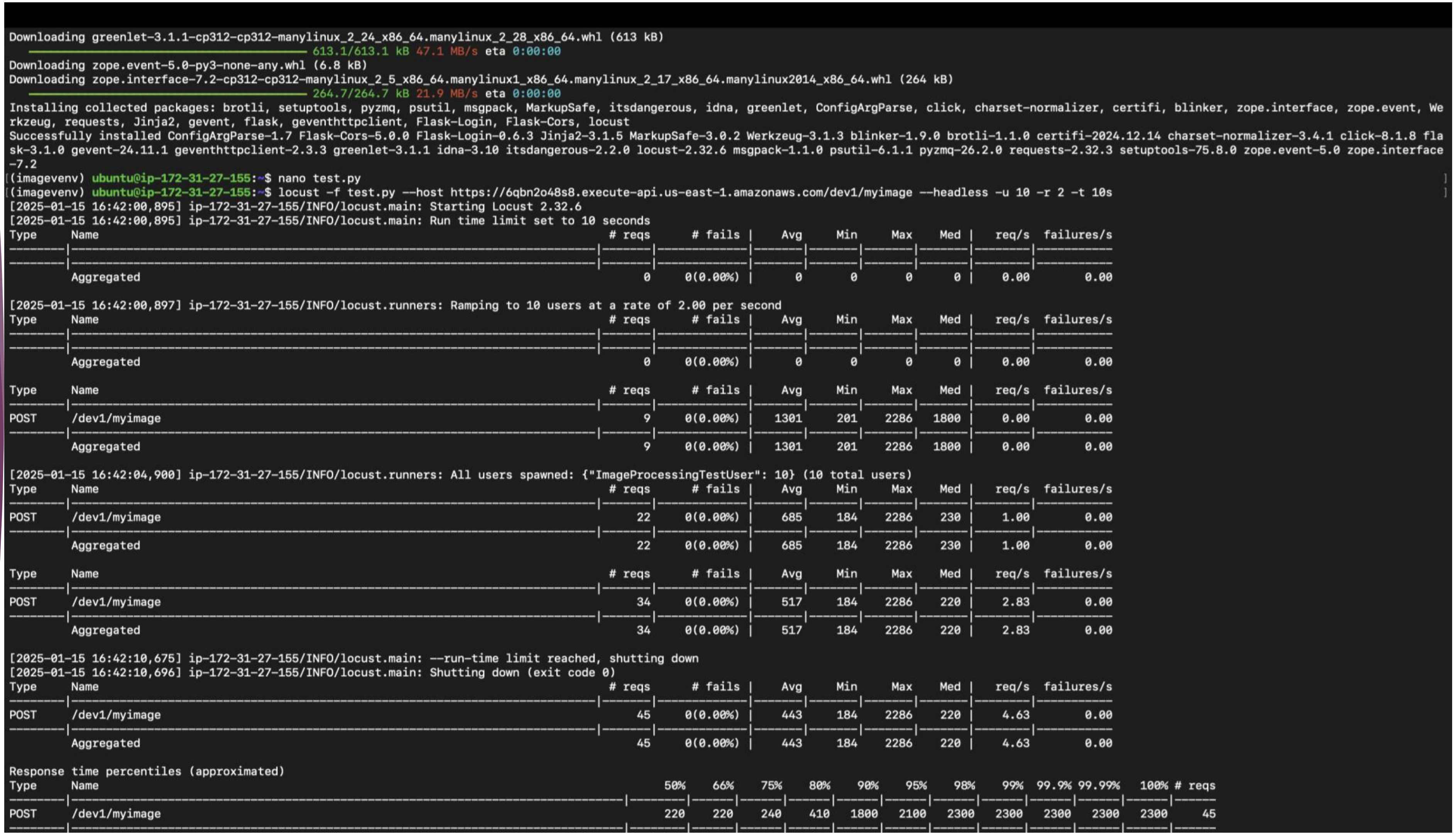
Resource ID
kkd7oczoee

Implementation Steps
3. Trigger Configuration

- Set up an S3 event trigger on the source bucket that will call the Lambda function. An event to listen to is "Object Created" for the function to be triggered every time a new image is uploaded. Besides that, file type filters can be further added, say ".jpg" and ".png", to have the trigger narrow down to those relevant image formats. This guarantees that only these files of interest are processed to smoothen the workflow.

Implementation Steps
4. Testing Configuration

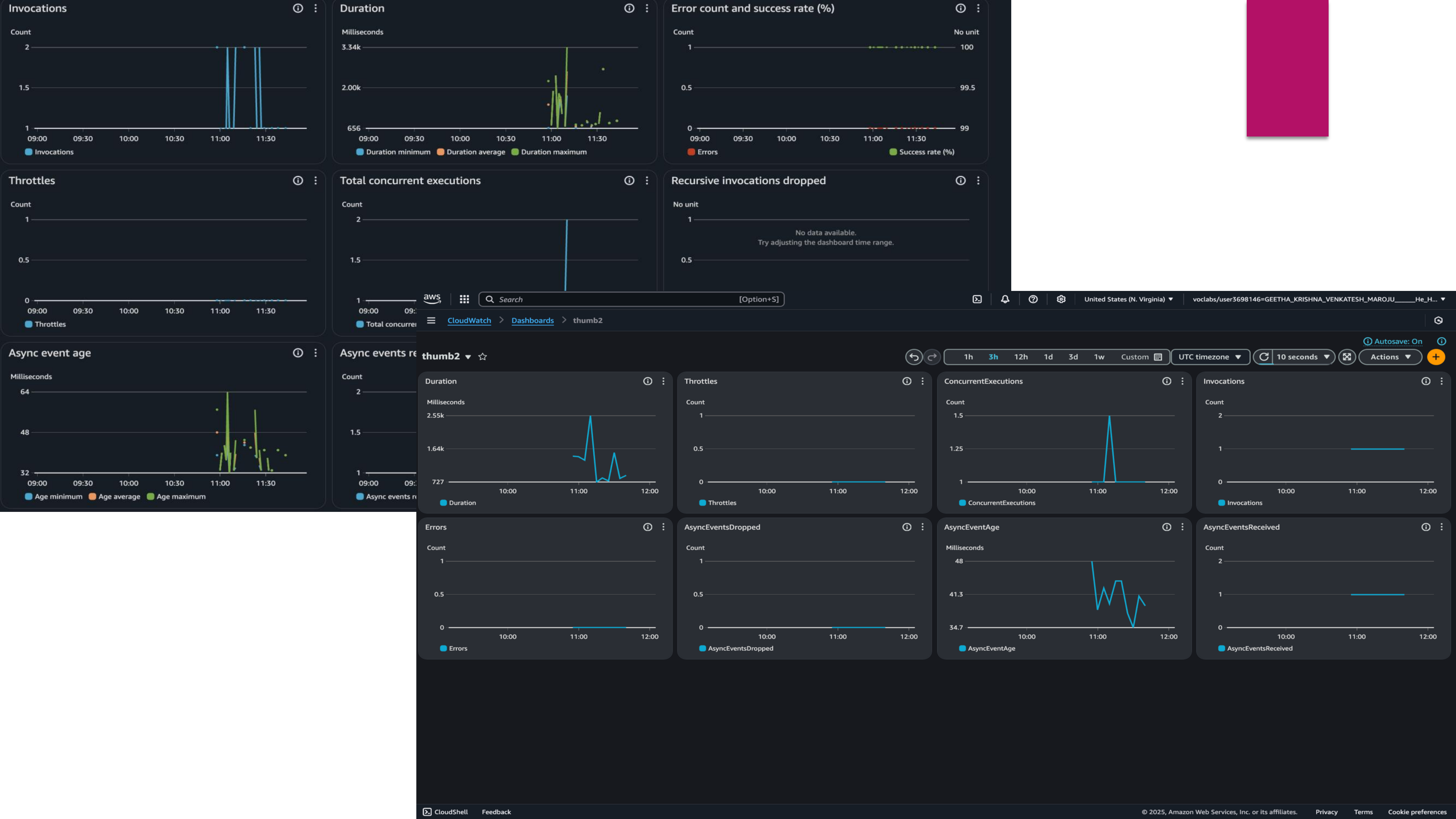
- Sample image upload to the source bucket will finally test the system. The functionality is tested by verifying that the resized images are populated in the destination bucket. AWS CloudWatch will be observed for logs and metrics to confirm that the Lambda function executes without errors and performance as expected. To this regard, necessary adjustments in the configuration would be done based on the observations so as to have the system up and running properly.



```
Downloading greenlet-3.1.1-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (613 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 613.1/613.1 kB 47.1 MB/s eta 0:00:00
Downloading zope.event-5.0-py3-none-any.whl (6.8 kB)
Downloading zope.interface-7.2-cp312-cp312-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (264 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 264.7/264.7 kB 21.9 MB/s eta 0:00:00
Installing collected packages: brotli, setuptools, pyzmq, psutil, msgpack, MarkupSafe, itsdangerous, idna, greenlet, ConfigArgParse, click, charset-normalizer, certifi, blinker, zope.interface, zope.event, We
rkzeug, requests, Jinja2, gevent, flask, geventhttpclient, Flask-Login, Flask-Cors, locust
Successfully installed ConfigArgParse-1.7 Flask-Cors-5.0.0 Flask-Login-0.6.3 Jinja2-3.1.5 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 brotli-1.1.0 certifi-2024.12.14 charset-normalizer-3.4.1 click-8.1.8 fla
sk-3.1.0 gevent-24.11.1 geventhttpclient-2.3.3 greenlet-3.1.1 idna-3.10 itsdangerous-2.2.0 locust-2.32.6 msgpack-1.1.0 psutil-6.1.1 pyzmq-26.2.0 requests-2.32.3 setuptools-75.8.0 zope.event-5.0 zope.interface
-7.2
[(imageenv) ubuntu@ip-172-31-27-155:~$ nano test.py
[(imageenv) ubuntu@ip-172-31-27-155:~$ locust -f test.py --host https://6qbn2o48s8.execute-api.us-east-1.amazonaws.com/dev1/myimage --headless -u 10 -r 2 -t 10s
[2025-01-15 16:42:00,895] ip-172-31-27-155/INFO/locust.main: Starting Locust 2.32.6
[2025-01-15 16:42:00,895] ip-172-31-27-155/INFO/locust.main: Run time limit set to 10 seconds
Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                         0     0(0.00%) |       0       0       0       0 |    0.00        0.00

[2025-01-15 16:42:00,897] ip-172-31-27-155/INFO/locust.runners: Ramping to 10 users at a rate of 2.00 per second
Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                         0     0(0.00%) |       0       0       0       0 |    0.00        0.00

Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
POST     /dev1/myimage                                                                      9     0(0.00%) |    1301     201    2286    1800 |    0.00        0.00
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                         9     0(0.00%) |    1301     201    2286    1800 |    0.00        0.00

[2025-01-15 16:42:04,900] ip-172-31-27-155/INFO/locust.runners: All users spawned: {"ImageProcessingTestUser": 10} (10 total users)
Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
POST     /dev1/myimage                                                                     22     0(0.00%) |     685     184    2286     230 |    1.00        0.00
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                        22     0(0.00%) |     685     184    2286     230 |    1.00        0.00

Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
POST     /dev1/myimage                                                                     34     0(0.00%) |     517     184    2286     220 |    2.83        0.00
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                        34     0(0.00%) |     517     184    2286     220 |    2.83        0.00

[2025-01-15 16:42:10,675] ip-172-31-27-155/INFO/locust.main: --run-time limit reached, shutting down
[2025-01-15 16:42:10,696] ip-172-31-27-155/INFO/locust.main: Shutting down (exit code 0)
Type     Name                                                                          # reqs      # fails |     Avg     Min     Max     Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
POST     /dev1/myimage                                                                     45     0(0.00%) |     443     184    2286     220 |    4.63        0.00
--------|-----------------------------------------------------------------------------|------|-------------|--------|-------|-------|--------|--------|-----------
         Aggregated                                                                        45     0(0.00%) |     443     184    2286     220 |    4.63        0.00

Response time percentiles (approximated)
Type     Name                                                                                  50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|--------------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
POST     /dev1/myimage                                                                         220    220    240    410   1800   2100   2300   2300   2300   2300   2300     45
--------|--------------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
```

Monitoring and Testing
1. Performance Monitoring

- Performance monitoring of the system is very crucial for reliable operation. AWS CloudWatch provides a set of metrics on the activity of the Lambda function that includes invocation count, execution duration, error count, and memory usage. These will, in turn, enable observation of real-time behavior of the system and pinpoint optimization opportunities. These insights can be used to tune the system to maintain its efficiency and potential bottlenecks. Graphs and screenshots of these metrics are attached herein, to give a clear view of the system performance during testing.

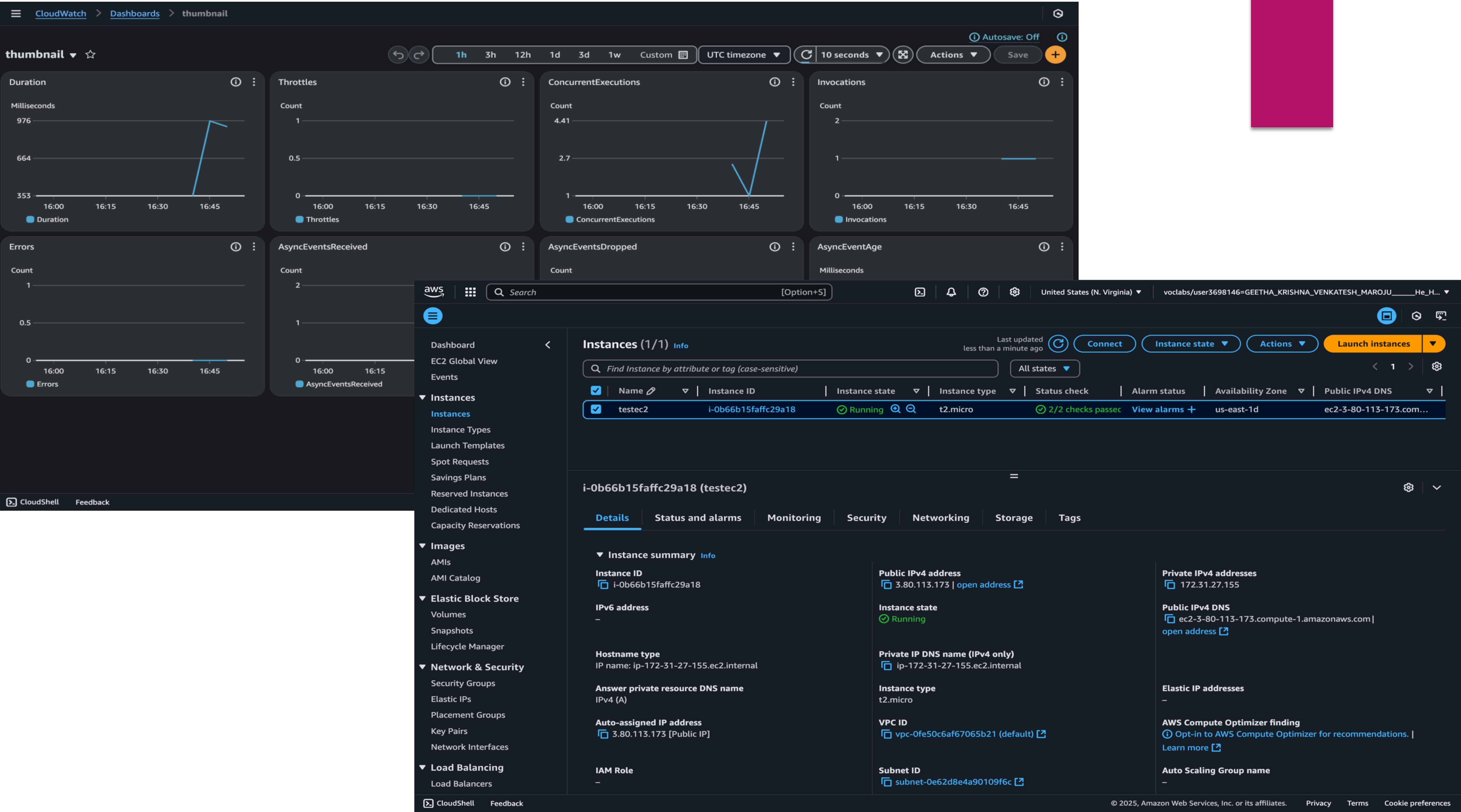

Invocations
Count
2
1.5
1
09:00
09:30
10:00
10:30
11:00
11:30
Invocations
Duration
Milliseconds
3.34k
2.00k
656
Duration minimum
Duration average
Duration maximum
Error count and success rate (%)
Count
No unit
100
99.5
99
0.5
0
Errors
Success rate (%)
Throttles
Total concurrent executions
Recursive invocations dropped
No data available.
Try adjusting the dashboard time range.
Async event age
64
48
32
Age minimum
Age average
Age maximum
Async events re
Async events r
Total concurrer
aws
Search
[Option+S]
United States (N. Virginia)
voclabs/user3698146=GEETHA_KRISHNA_VENKATESH_MAROJU_____He_H...
CloudWatch
Dashboards
thumb2
Autosave: On
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
10 seconds
Actions
Duration
2.55k
1.64k
727
10:00
11:00
12:00
Throttles
ConcurrentExecutions
1.25
Invocations
Errors
AsyncEventsDropped
AsyncEventAge
41.3
34.7
AsyncEventsReceived
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Monitoring and Testing
2. Scaling Testing Methods

- To test the scalability of the system, several images were manually uploaded to the source bucket to simulate parallel executions. This procedure tested how the Lambda function would handle multiple requests at the same time. Metrics watched in CloudWatch include processing times and success/failure rates to see how the system responds under load. Observations from these tests demonstrate the system's capability to scale efficiently while maintaining performance.

Monitoring and Testing
3. Testing Scenarios

- It involves a number of different test scenarios aimed at proving that the system is operational under various conditions. These included tests for uploading a single image to check the basic functionality, multiple concurrent uploads to check on load handling to evaluate consistent performance of the system. The system was also tested with different formats of images, such as.jpg and.png.
- For stress testing, Locust was used to simulate high loads on the system. The following command was executed to conduct the stress test:
  " locust -f test.py --host https://6qbn2o48s8.execute-api.us-east-1.amazonaws.com/dev1/myimage --headless -u 10 -r 2 -t 10s".
- This command initiated 10 simulated users with a spawn rate of 2 users per second for a duration of 10 seconds. The results are updated below screenshots.



Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
testec2	i-0b66b15faffc29a18	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1d	ec2-3-80-113-173.com...

i-0b66b15faffc29a18 (testec2)

Instance summary Info

Instance ID
i-0b66b15faffc29a18

Public IPv4 address
3.80.113.173 | open address

Private IPv4 addresses
172.31.27.155

IPv6 address
–

Instance state
Running

Public IPv4 DNS
ec2-3-80-113-173.compute-1.amazonaws.com | open address

Hostname type
IP name: ip-172-31-27-155.ec2.internal

Private IP DNS name (IPv4 only)
ip-172-31-27-155.ec2.internal

Answer private resource DNS name
IPv4 (A)

Instance type
t2.micro

Elastic IP addresses
–

Auto-assigned IP address
3.80.113.173 [Public IP]

VPC ID
vpc-0fe50c6af67065b21 (default)

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

IAM Role
–

Subnet ID
subnet-0e62d8e4a90109f6c

Auto Scaling Group name
–

Conclusion

- The project designed a serverless architecture for image processing using AWS, which helped in implementing an efficient and scalable solution. With AWS Lambda, the system scales automatically to whatever workload is required of it. Further, this is cost-effective and robust since implementation was done leveraging institutional AWS resources.

- It generated thumbnails from uploaded images reliably, proving to be consistent in performance for file types and sizes. This project also provided great experience in the development of serverless applications and further developed skills in cloud architecture, automation, and performance monitoring. These learnings set a very strong foundation for using serverless technologies in future projects.