



Cloud Computing Project

Presented by:

Geetha Krishna Venkatesh
Maraju

Satish kumar Buravelli

Sai Nandu Posina

Contents



Project Aim and Requirements



Architecture Overview



Implementation Steps



Monitoring and Testing



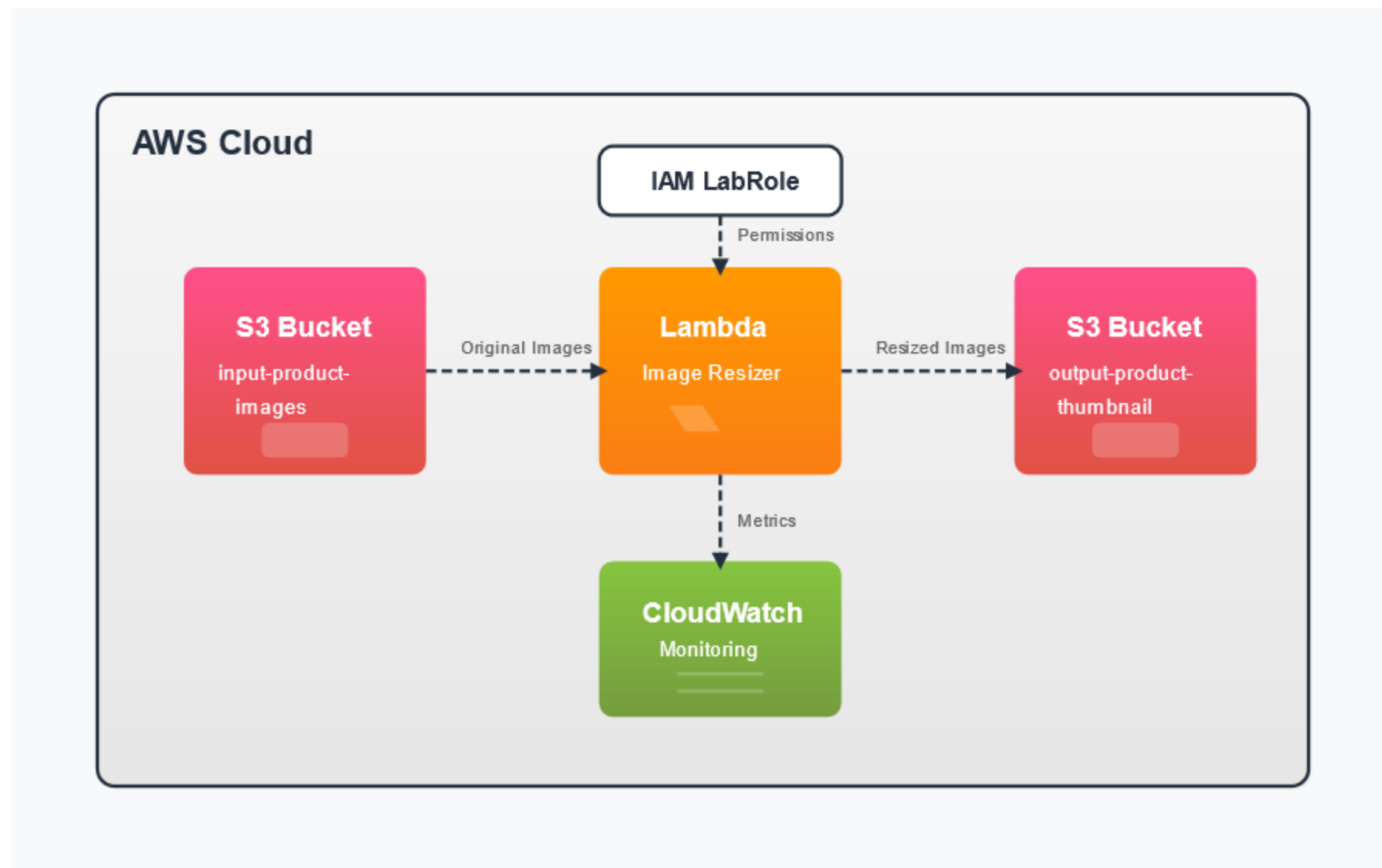
Conclusion

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

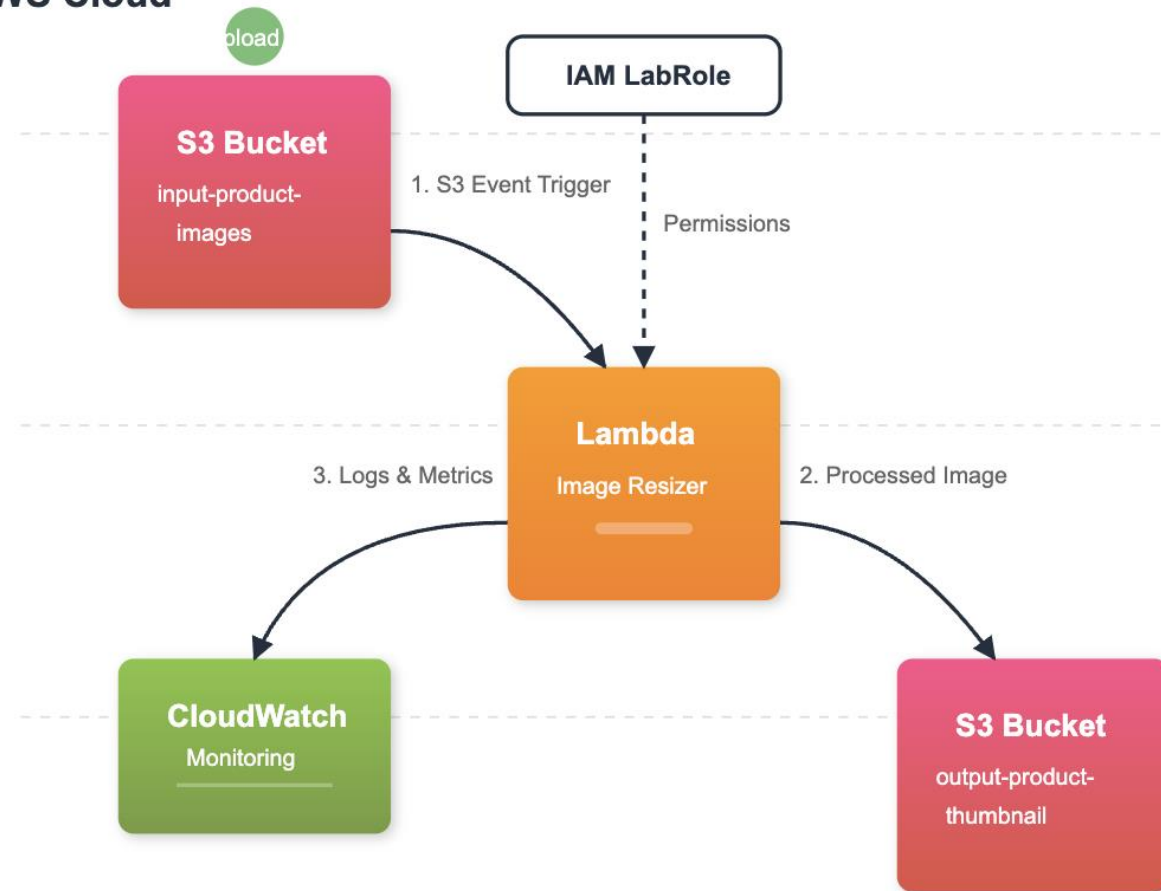
- ▶ The system architecture is designed to efficiently handle image resizing workflows using AWS services. It consists of a source S3 bucket named (input-product-images), which holds the original images, and a destination S3 bucket named (output-product-thumbnail) where the resized images will be stored. 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.



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



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.

Amazon S3

General purpose buckets

- Directory buckets
- Table buckets
- Access Grants
- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- IAM Access Analyzer for S3

Block Public Access settings for this account

▼ Storage Lens

- Dashboards
- Storage Lens groups
- AWS Organizations settings

Feature spotlight

► AWS Marketplace for S3

Objects

Metadata - Preview

Properties

Permissions

Metrics

Management

Access Points

Objects (3)

Info

↺

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

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](#)

Show versions

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	cloudwatch-medium.png	png	January 14, 2025, 16:03:44 (UTC+01:00)	289.5 KB	Standard
<input type="checkbox"/>	Medium-input(30)/	Folder	-	-	-
<input type="checkbox"/>	small-load/	Folder	-	-	-

aws

Search

[Option+S]

📄

🔔

❓

⚙️

United States (N. Virginia)

Amazon S3 > Buckets > output-product-thumbnail

Amazon S3

- General purpose buckets
- Directory buckets
- Table buckets
- Access Grants
- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- IAM Access Analyzer for S3

- Block Public Access settings for this account
- ▼ **Storage Lens**
 - Dashboards
 - Storage Lens groups
 - AWS Organizations settings
- Feature spotlight
- **AWS Marketplace for S3**

output-product-thumbnail

Info

Objects

Metadata - Preview

Properties

Permissions

Metrics

Management

Access Points

Objects (4)

Info

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

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](#)

Q

Find objects by prefix

<

1

>

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	aws/	Folder	-	-	-
<input type="checkbox"/>	thumb_cloudwatch-medium.png	png	January 14, 2025, 16:03:49 (UTC+01:00)	17.1 KB	Standard
<input type="checkbox"/>	thumb_Medium-input(30)/	Folder	-	-	-
<input type="checkbox"/>	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

Throttle

Copy ARN

Actions

▼ Function overview [Info](#)

Diagram

Template

Thumbnail icon

Thumbnails

Layers icon

Layers (1)

S3 icon

S3

Amazon S3 icon

Amazon S3

+ Add trigger

+ Add destination

Export to Infrastructure Composer

Download ▼

Description

-

Last modified

1 day ago

Function ARN

[arn:aws:lambda:us-east-1:437178870819:function:Thumbnails](#)

Function URL [Info](#)

-

[Lambda](#) > [Functions](#) > [Thumbnails](#)

The screenshot displays the AWS CloudShell environment. On the left, the 'EXPLORER' pane shows a file tree with a 'THUMBNAILS' folder containing 'lambda_function.py' and a 'DEPLOY' folder. The main editor window shows the code for 'lambda_function.py'. The code includes imports for 'json', 'boto3', 'io', 'PIL Image', 'logging', and 'urllib.parse'. The 'lambda_handler' function is defined, which takes 'event' and 'context' as arguments. It uses 'boto3' to interact with S3 buckets, specifically 'input-product-images' and 'output-product-thumbnail'. The interface also features a 'Deploy' button (labeled 'Deploy (⇧⌘U)') and a 'Test' button (labeled 'Test (⇧⌘I)').

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

TEST EVENTS [SELECTED: PILLOWTEST]
+ Create new test event
Private saved events
PillowTest

ENVIRONMENT VARIABLES

```
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
```

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python Lambda Layout: U.S.

	1980	1985	1990	1995	2000	2005	2010	2015	2020
Population	76.5	80.5	84.5	88.5	92.5	96.5	100.5	104.5	108.5
GDP per capita	1,200	1,500	1,800	2,100	2,400	2,700	3,000	3,300	3,600
Life expectancy at birth	65	68	71	74	77	80	83	86	89
Urban population (%)	35	40	45	50	55	60	65	70	75
Employment rate (%)	55	58	61	64	67	70	73	76	79
Unemployment rate (%)	15	12	10	8	7	6	5	4	3
Government expenditure as % of GDP	18	20	22	24	26	28	30	32	34
Private consumption as % of GDP	55	58	61	64	67	70	73	76	79
Investment as % of GDP	12	14	16	18	20	22	24	26	28
Exports as % of GDP	10	12	14	16	18	20	22	24	26
Imports as % of GDP	12	14	16	18	20	22	24	26	28
Current account balance as % of GDP	-2	-2	-2	-2	-2	-2	-2	-2	-2
Foreign direct investment as % of GDP	1	2	3	4	5	6	7	8	9
Official development assistance as % of GDP	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3
Net international reserves as % of GDP	5	6	7	8	9	10	11	12	13
Public debt as % of GDP	40	45	50	55	60	65	70	75	80
Private debt as % of GDP	10	12	14	16	18	20	22	24	26
Total debt as % of GDP	50	57	64	71	78	85	92	99	106
Household savings as % of GDP	8	9	10	11	12	13	14	15	16
Corporate savings as % of GDP	4	5	6	7	8	9	10	11	12
Government savings as % of GDP	2	3	4	5	6	7	8	9	10
Total savings as % of GDP	14	17	20	23	26	29	32	35	38
Capital formation as % of GDP	12	14	16	18	20	22	24	26	28
Fixed capital formation as % of GDP	10	12	14	16	18	20	22	24	26
Consumption of fixed capital as % of GDP	2	2	2	2	2	2	2	2	2
Net capital formation as % of GDP	10	12	14	16	18	20	22	24	26
Research and development as % of GDP	1	1.5	2	2.5	3	3.5	4	4.5	5
High-tech exports as % of total exports	5	6	7	8	9	10	11	12	13
High-tech imports as % of total imports	10	11	12	13	14	15	16	17	18
High-tech trade balance as % of total trade	-5	-5	-5	-5	-5	-5	-5	-5	-5
High-tech FDI as % of total FDI	10	12	14	16	18	20	22	24	26
High-tech ODA as % of total ODA	5	6	7	8	9	10	11	12	13
High-tech NIRA as % of total NIRA	5	6	7	8	9	10	11	12	13
High-tech PDI as % of total PDI	5	6	7	8	9	10	11	12	13
High-tech CFI as % of total CFI	5	6	7	8	9	10	11	12	13
High-tech HFI as % of total HFI	5	6	7	8	9	10	11	12	13
High-tech GFI as % of total GFI	5	6	7	8	9	10	11	12	13
High-tech IPI as % of total IPI	5	6	7	8	9	10	11	12	13
High-tech EPI as % of total EPI	5	6	7	8	9	10	11	12	13
High-tech SBI as % of total SBI	5	6	7	8	9	10	11	12	13
High-tech TPI as % of total TPI	5	6	7	8	9	10	11	12	13
High-tech CPI as % of total CPI	5	6	7	8	9	10	11	12	13
High-tech DFI as % of total DFI	5	6	7	8	9	10	11	12	13
High-tech OFI as % of total OFI	5	6	7	8	9	10	11	12	13
High-tech RFI as % of total RFI	5	6	7	8	9	10	11	12	13
High-tech BFI as % of total BFI	5	6	7	8	9	10	11	12	13
High-tech LFI as % of total LFI	5	6	7	8	9	10	11	12	13
High-tech MFI as % of total MFI	5	6	7	8	9	10	11	12	13
High-tech NFI as % of total NFI	5	6	7	8	9	10	11	12	13
High-tech OFI as % of total OFI	5	6	7	8	9	10	11	12	13
High-tech RFI as % of total RFI									

Code properties Info		
Package size	SHA256 hash	Last modified
1,023 byte	 v7A7tJ3wD0Fipqw37qbe6LjDJJsMZjl0mh+vZk+OE8A=	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

piilowlinux

Delete

Download

Create version

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

-

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

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.

API Gateway

APIs

Custom domain names

Domain name access associations

VPC links

▼ API: testapi

Resources

Stages

Authorizers

Gateway responses

Models

Resource policy

Documentation

Dashboard

API settings

Usage plans

API keys

Client certificates

Settings

Resources

Create resource

/

/myimage

OPTIONS

POST

Resource details

Path

/

Resource ID

kkd7oczoe

API actions

Deploy API

Update documentation

Enable CORS

Methods (0)

Delete

Create method

Method type	Integration type	Authorization	API key
No methods			
No methods defined.			

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

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, Werkzeug, 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 flask-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
(imagevenv) ubuntu@ip-172-31-27-155:~$ nano test.py
(imagevenv) 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      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      0      0      0    | 0.00    0.00

Type      Name                                     # reqs   # fails | Avg    Min    Max    Med    | req/s  failures/s
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
POST      /dev1/myimage                           9         0 | 1301   201   2286   1800   | 0.00    0.00
Aggregated                                     9         0 | 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 | 685    184   2286   230    | 1.00    0.00
Aggregated                                     22         0 | 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 | 517    184   2286   220    | 2.83    0.00
Aggregated                                     34         0 | 517    184   2286   220    | 2.83    0.00

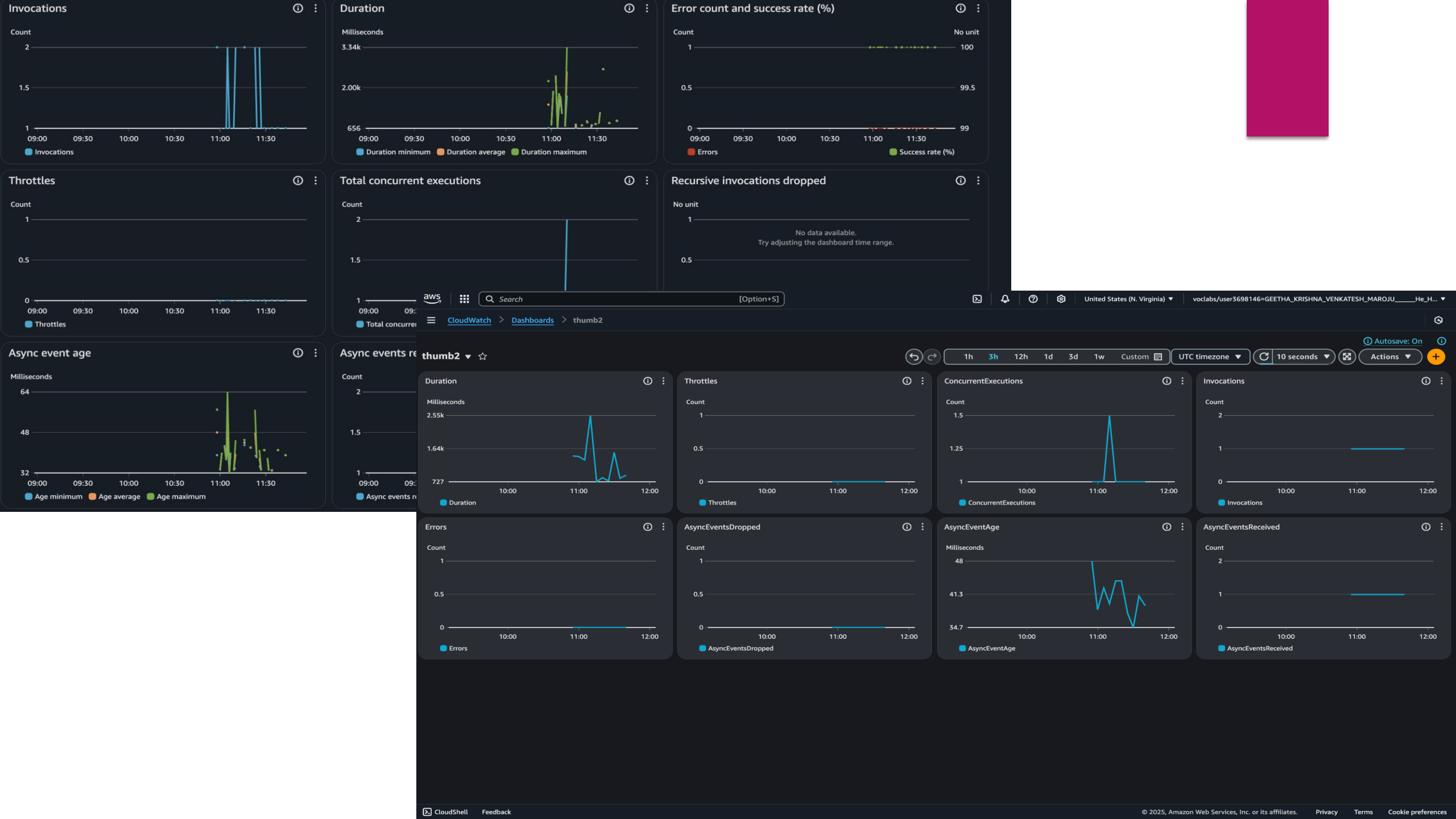
[2025-01-15 16:42:10,675] ip-172-31-27-155/INFO/locust.main: --run-time limit reached, shutting down
[2025-01-15 16:42:10,696] ip-172-31-27-155/INFO/locust.main: Shutting down (exit code 0)
Type      Name                                     # reqs   # fails | Avg    Min    Max    Med    | req/s  failures/s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



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.

CloudWatch > Dashboards > thumbnail

thumbnail

Autosave: Off

1h3h12h1d3d1wCustomUTC timezone10 secondsActionsSave

Duration

Milliseconds

976664353

16:0016:1516:3016:45

Duration

Throttles

Count

10.50

16:0016:1516:3016:45

Throttles

ConcurrentExecutions

Count

4.412.71

16:0016:1516:3016:45

ConcurrentExecutions

Invocations

Count

210

16:0016:1516:3016:45

Invocations

Errors

Count

10.50

16:0016:1516:3016:45

Errors

AsyncEventsReceived

Count

21

16:0016:15

AsyncEventsReceived

AsyncEventsDropped

Count

0

16:0016:15

AsyncEventsDropped

AsyncEventAge

Milliseconds

0

16:0016:15

AsyncEventAge

aws

Search

[Option+S]

United States (N. Virginia)

voclabs/user3698146=GEETHA_KRISHNA_VENKATESH_MAROJU_____He_H...

Dashboard

EC2 Global View

Events

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Capacity Reservations

Images

AMIs

AMI Catalog

Elastic Block Store

Volumes

Snapshots

Lifecycle Manager

Network & Security

Security Groups

Elastic IPs

Placement Groups

Key Pairs

Network Interfaces

Load Balancing

Load Balancers

Instances (1/1) Info

Last updated less than a minute ago

Connect

Instance state

Actions

Launch instances

Find Instance by attribute or tag (case-sensitive)

All states

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

i-0b66b15faffc29a18 (testec2)

DetailsStatus and alarmsMonitoringSecurityNetworkingStorageTags

Instance summary Info

Instance ID

i-0b66b15faffc29a18

IPv6 address

-

Hostname type

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

Answer private resource DNS name

IPv4 (A)

Auto-assigned IP address

3.80.113.173 [Public IP]

IAM Role

-

Public IPv4 address

3.80.113.173 | open address

Instance state

Running

Private IP DNS name (IPv4 only)

ip-172-31-27-155.ec2.internal

Instance type

t2.micro

VPC ID

vpc-0fe50c6af67065b21 (default)

Subnet ID

subnet-0e62d8e4a90109f6c

Private IPv4 addresses

172.31.27.155

Public IPv4 DNS

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

Elastic IP addresses

-

AWS Compute Optimizer finding

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

Auto Scaling Group name

-

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

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.