

ML Model Deployment on AWS Lambda: Observing Scalability

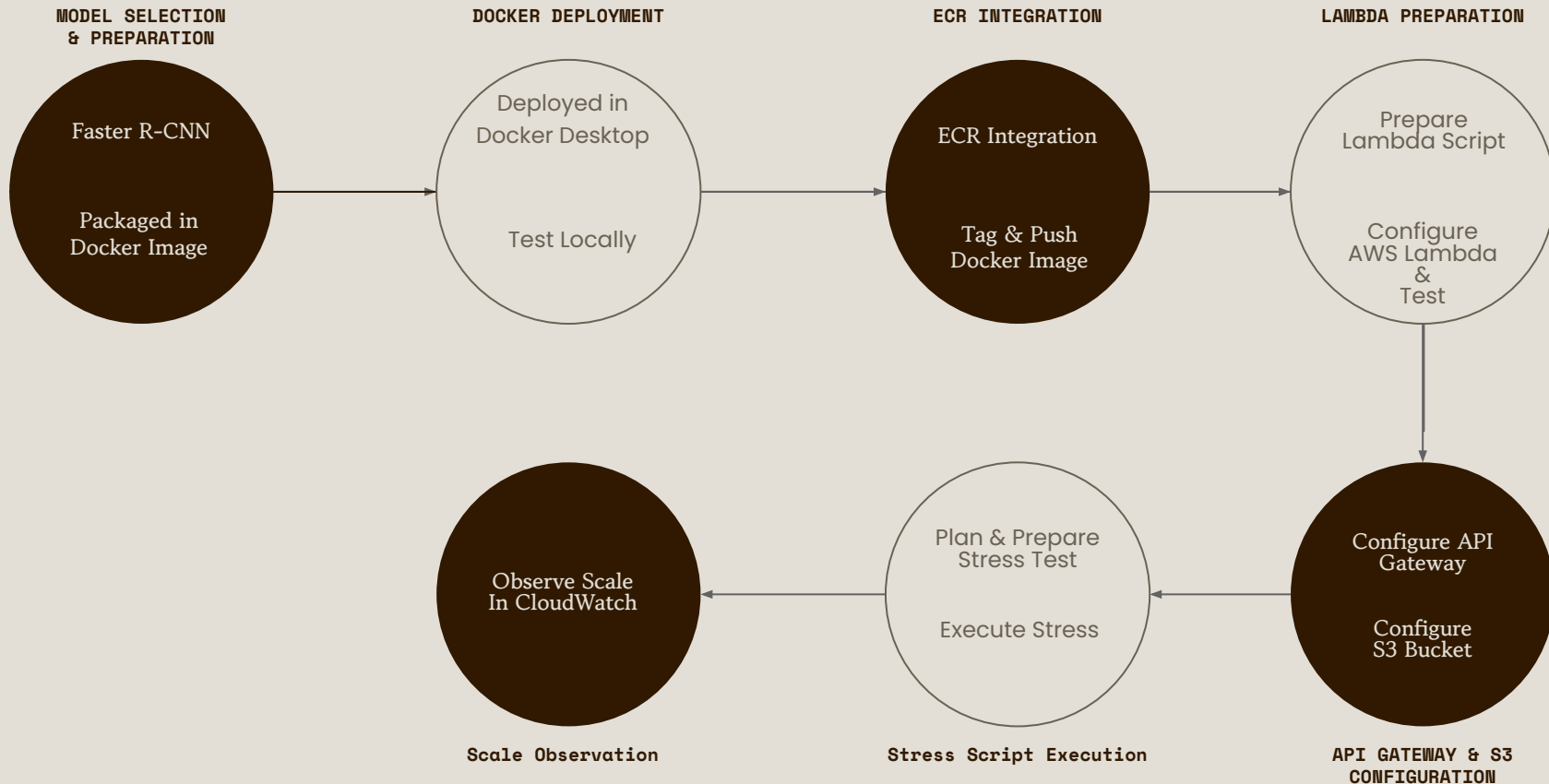
BOKHTIAR MEHEDY

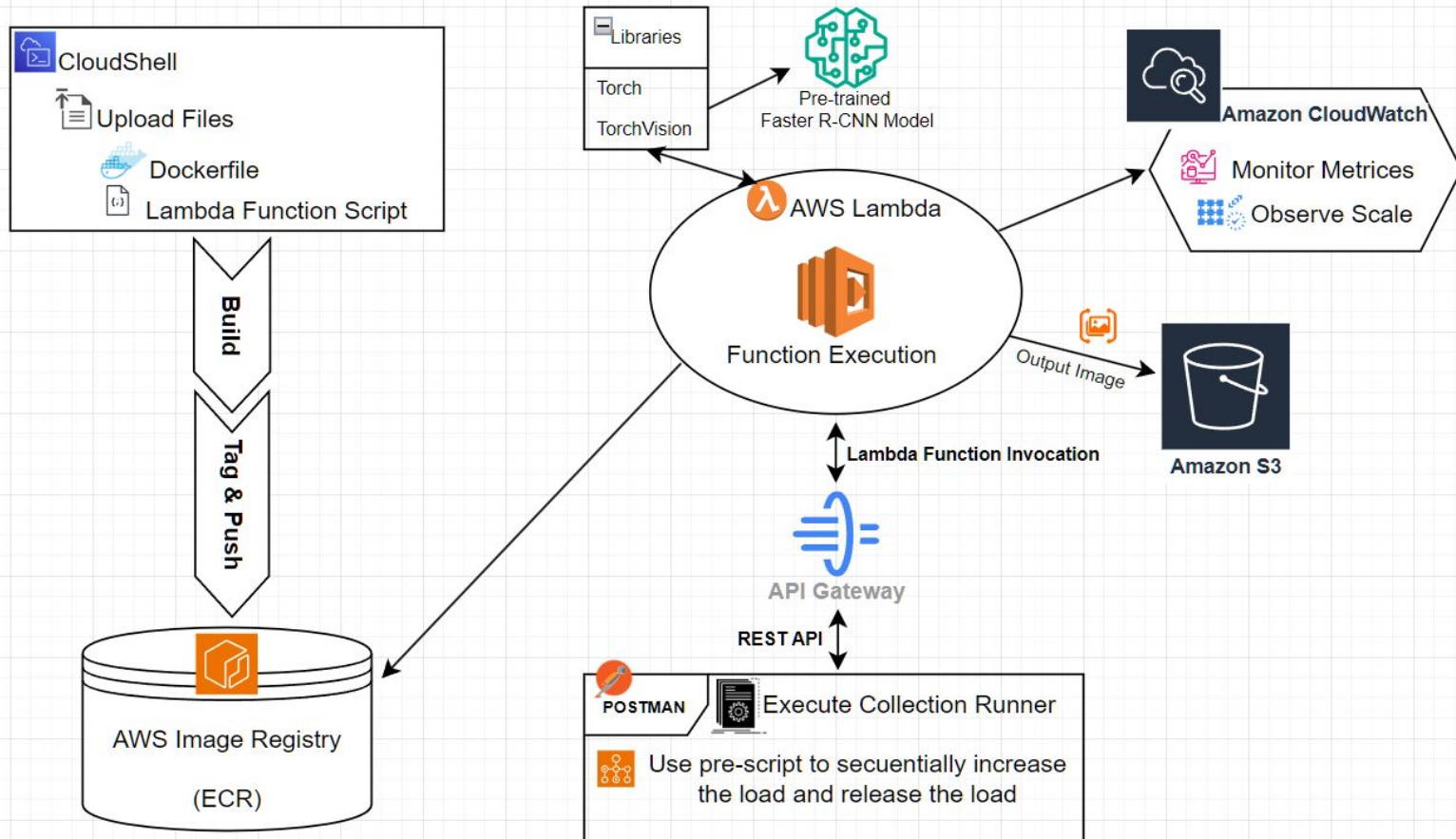
Objective

- ❑ Deploy a pre-trained Faster R-CNN model for accurate **object detection** on AWS Lambda
- ❑ Observe **deployment scale** by analyzing performance metrics in CloudWatch

Solution Overview

3





Model Selection

Model Evaluation & Selection

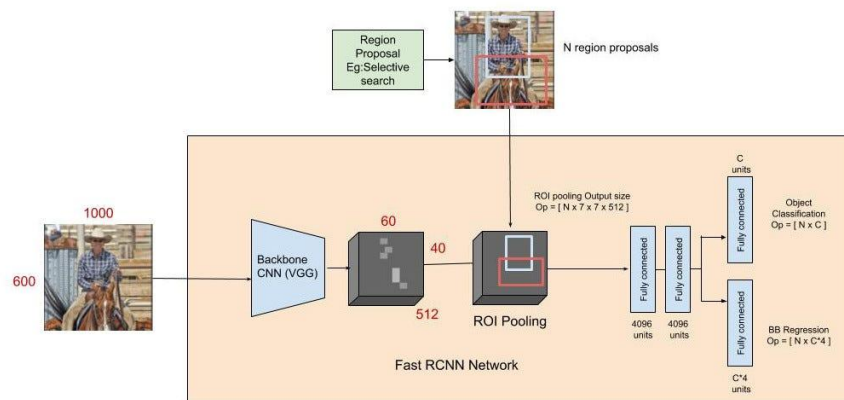
- Lightweight models like **SSDlite** were tried but observed poor detection performance.
- Selected **Faster R-CNN** for superior accuracy despite its heavier architecture.

Language and Library

- Decided to use **Python** as programming language
- PyTorch (Torch & TorchVision) as library
- Pre-trained model ***fasterrcnn_resnet50_fpn_v2*** and weights were used.

Challenge

- The size of the libraries are more than 700MB (170 MB CPU only)
- Size of the pre-trained model in 168MB
- Lambda has size constraints for **direct deployment** (250 MB limit for zipped packages) and **AWS Lambda Layers** (50 MB per layer)
- Necessitating the use of a **Docker image** for deployment (up to 10 GB)
- Using a Docker Image even considered standard for Lambda deployment



Preparing Docker Image

Dockerfile

- Configured **Dockerfile** with Lambda-compatible Python runtime to build the image.
- Packaged the Faster R-CNN model, dependencies (Torch and torchvision), and the `lambda_function.py` script that implements the object detection.

Validation

- Locally validated the image using Docker Desktop

```
1 # Start with the official AWS Lambda Python runtime image
2 FROM public.ecr.aws/lambda/python:3.8
3
4 # Install necessary dependencies
5 RUN pip install torch==2.4.1 torchvision==0.19.1 --index-url https://download.pytorch.org/whl/cpu
6
7 # Copy the Lambda function code
8 COPY lambda_function.py .
9
10 # Set the handler to function
11 CMD ["lambda_function.lambda_handler"]
12
```

Preparing Docker Image

Lambda Function Script

```
1 import os
2 import boto3
3 import torch
4 from torchvision.models.detection import fasterrcnn_resnet50_fpn_v2, FasterRCNN_ResNet50_FPN_V2_Weights
5 from torchvision.transforms import functional as F
6 from PIL import Image, ImageDraw
7 import io
8 import base64
9
10 # Set model weights to download in /tmp
11 os.environ["TORCH_HOME"] = "/tmp"
12
13 weights = FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT
14 model = fasterrcnn_resnet50_fpn_v2(weights=weights, box_score_thresh=0.9)
15 model.eval()
16
17 # Initialize S3 client
18 s3_client = boto3.client('s3')
19
20 # Lambda function handler
21 def lambda_handler(event, context):
22     try:
23         # Directly decode the base64-encoded image from the body
24         image_data = base64.b64decode(event["body"])
25
26         # Decode image
27         image = Image.open(io.BytesIO(image_data))
28
29         # Preprocess the image
30         transform = weights.transforms()
31         image_tensor = transform(image).unsqueeze(0)
32
```

```
33
34     # Perform object detection
35     with torch.no_grad():
36         prediction = model(image_tensor)[0]
37
38     # Extract predictions
39     boxes = prediction["boxes"].cpu().numpy()
40     labels = prediction["labels"].cpu().numpy()
41
42     # Draw bounding boxes on the image
43     draw = ImageDraw.Draw(image)
44     for box in boxes:
45         draw.rectangle(box.tolist(), outline="red", width=3)
46
47     # Save the processed image to /tmp
48     processed_image_path = "/tmp/processed_image.jpg"
49     image.save(processed_image_path)
50
51     # Upload the processed image to an S3 bucket
52     bucket_name = "object-detected-images"
53     object_key = "processed_image.jpg"
54     s3_client.upload_file(processed_image_path, bucket_name, object_key)
55
56     s3_url = f"https://{bucket_name}.s3.amazonaws.com/{object_key}"
57
58     return {
59         "statusCode": 200,
60         "body": {
61             "boxes": boxes.tolist(),
62             "labels": labels.tolist(),
63             "s3_url": s3_url
64         }
65     }
```

Configuring ECR

Configure the ECR

- To push our Docker Image we configured AWS Elastic Cloud Repository (ECR)
- Later we will create the Lambda Function from our image on the ECR

The screenshot displays the AWS Management Console interface for creating a new private repository in Amazon ECR. The top navigation bar shows the AWS logo, a search bar, and the user's profile. The main content area is titled 'Create private repository' and is divided into two main sections: 'General settings' and 'Encryption'.

General settings

Repository name
Provide a concise name. Repository names support namespaces, which is recommended for grouping similar repositories.
998273845759.dkr.ecr.us-east-1.amazonaws.com/
20 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, and special characters _ and -.

Image tag mutability [Info](#)
Specify the tag mutability setting to use. When tag immutability is turned on for a repository, tags are prevented from being overwritten.

☒ **Mutable**
Image tags can be overwritten.

☐ **Immutable**
Image tags are prevented from being overwritten.

Encryption

Encryption configuration [Info](#)
By default, repositories use the industry standard Advanced Encryption Standard (AES) encryption. You can optionally choose to use a key stored in the AWS Key Management Service (KMS) to encrypt the images in your repository.

☒ **AES-256**
Industry standard Advanced Encryption Standard (AES) encryption

☐ **AWS KMS**
AWS Key Management Service (KMS)

[Image scanning settings - deprecated](#)

At the bottom right, there are two buttons: 'Cancel' and 'Create'.

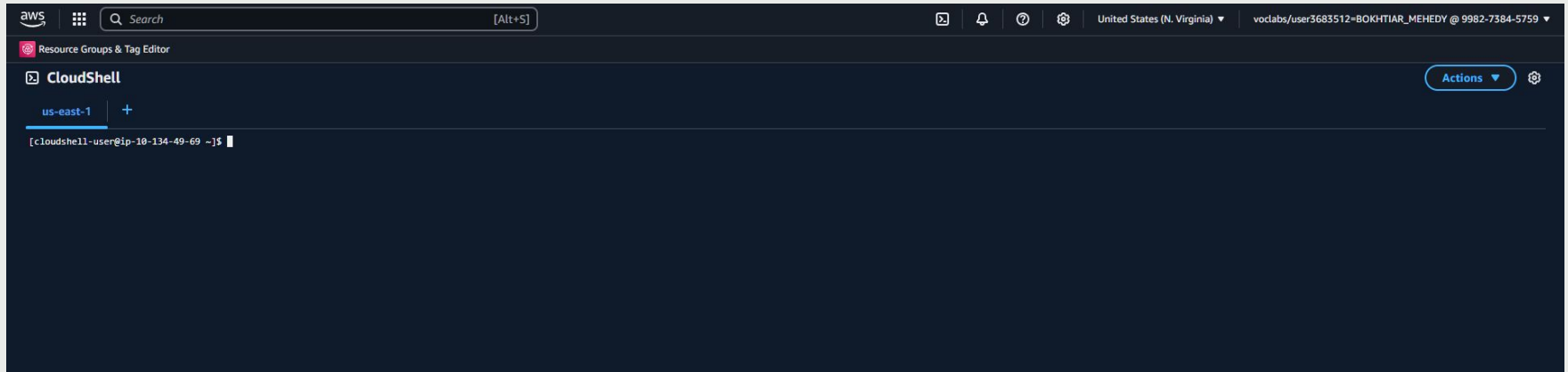
Using AWS CloudShell

AWS CLI

Unable to configure AWS CLI in local PC due to limited access (couldn't create a new IAM).

AWS CloudShell

Decided to use CloudShell to deploy the image in ECR



Build, Tag, & Push Image

Build, Tag, & Push the image to Repository

- We uploaded the **Dockerfile** and the **lambda_function.py** in CloudShell
- Then using CloudShell command window we ran commands to build the image and the Tag and Push it to the repository.

```
CloudShell
us-east-1 +
[cloudshell-user@ip-10-134-49-69 ~]$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 998273845759.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/cloudshell-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[cloudshell-user@ip-10-134-49-69 ~]$ ls
Dockerfile  lambda_function.py
[cloudshell-user@ip-10-134-49-69 ~]$
```

```
CloudShell
us-east-1 +
[cloudshell-user@ip-10-134-49-69 ~]$ docker tag lambda-image:latest 998273845759.dkr.ecr.us-east-1.amazonaws.com/image-detection-repo:latest
[cloudshell-user@ip-10-134-49-69 ~]$ docker push 998273845759.dkr.ecr.us-east-1.amazonaws.com/image-detection-repo:latest
The push refers to repository [998273845759.dkr.ecr.us-east-1.amazonaws.com/image-detection-repo]
f2674f2659d2: Pushed
e83856074ab5: Pushed
6d51ed7ff43e: Pushed
2f3c3a733224: Pushed
0091b0f618e4: Pushed
ec2c8b689377: Pushed
d7a84c55183c: Pushed
71de022b9e13: Pushed
latest: digest: sha256:7d81866676e629f4d57c9b762b806357cdf2de89cc5a26946f57c130bca1b70a size: 2002
[cloudshell-user@ip-10-134-49-69 ~]$
```

Build, Tag, & Push Image

```
CloudShell Actions ⓘ

us-east-1 +

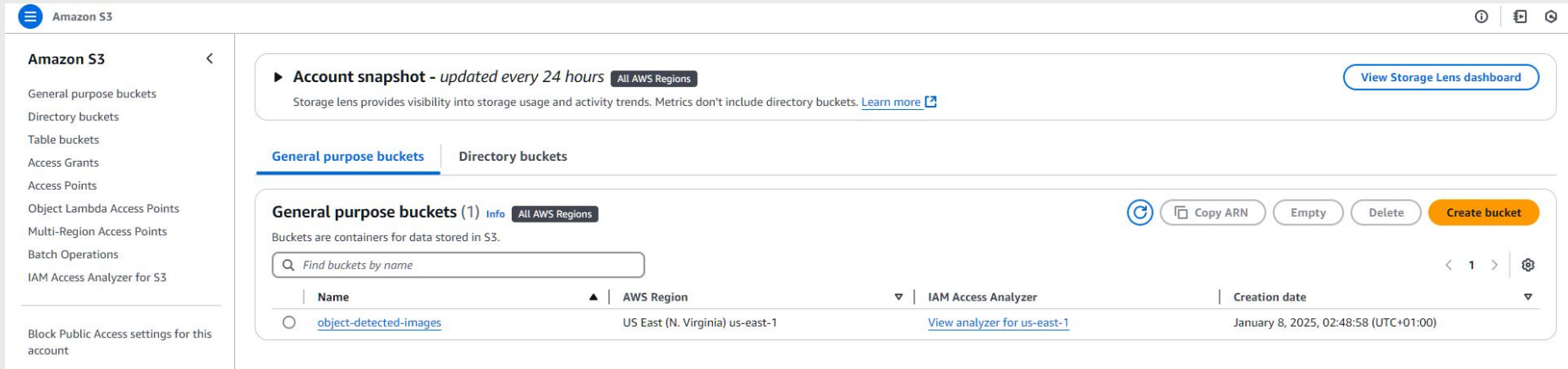
[cloudshell-user@ip-10-134-49-69 ~]$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password stdin 998273845759.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/cloudshell-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[cloudshell-user@ip-10-134-49-69 ~]$ ls
Dockerfile  lambda_function.py
[cloudshell-user@ip-10-134-49-69 ~]$ docker --version
Docker version 25.0.5, build 5dc9bcc
[cloudshell-user@ip-10-134-49-69 ~]$ docker build -t lambda-image .
[*] Building 189.7s (8/8) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 550B
-> [internal] load metadata for public.ecr.aws/lambda/python:3.8
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [1/3] FROM public.ecr.aws/lambda/python:3.8sha256:8f0e264805ec1d4a3e8e89e897f6085f93a980ea8b868066d5ba1b10b4b7392
-> resolve public.ecr.aws/lambda/python:3.8sha256:8f0e264805ec1d4a3e8e89e897f6085f93a980ea8b868066d5ba1b10b4b7392
-> sha256:2e1f849595e34411550bccc0901191931eeb3ada3c1226f8d8399814ff4900 / 1.50kB
-> sha256:c04c180a9d05f43722c4c1dae77a36d0851430979306aea37c55381560a6c2f / 67.99kB / 87.99kB
-> sha256:50fa92adb1b6d9e272221b3ad560fffa7a91b06af875d8d91e9c404a3d03678 / 417B / 417B
-> sha256:8f0e264805ec1d4a3e8e89e897f6085f93a980ea8b868066d5ba1b10b4b7392 / 772B / 772B
-> sha256:335ec1770ff5fec4c7e2d0009cf84024cf6f2e0b1c2208723ccef97959d / 4.30kB / 4.30kB
-> sha256:551ae20c11330302bc3a1d0ab0bc2cfeacde34da0433da09f60316d28c68 / 109.07kB / 109.07kB
-> sha256:35e8cd243a506790d0fe4ac7e61c71ab19aa4b834415590e52d380b930e5fe78 / 2.60kB / 2.60kB
-> sha256:e1318558c667cfa5c742bbbcf199869dfca96580e715221d0bc3be5c6d25c / 56.34kB / 56.34kB
-> sha256:7b27d0e323bc7a902add6347a485506de16a7365b6c021ac74101972d573559a / 21.07kB / 21.07kB
-> extracting sha256:551ae20c11330302bc3a1d0ab0bc2cfeacde34da0433da09f60316d28c68 / 9.7s
-> extracting sha256:c04c180a9d05f43722c4c1dae77a36d0851430979306aea37c55381560a6c2f / 0.0s
-> extracting sha256:50fa92adb1b6d9e272221b3ad560fffa7a91b06af875d8d91e9c404a3d03678 / 0.0s
-> extracting sha256:8f0e264805ec1d4a3e8e89e897f6085f93a980ea8b868066d5ba1b10b4b7392 / 0.0s
-> extracting sha256:e1318558c667cfa5c742bbbcf199869dfca96580e715221d0bc3be5c6d25c / 0.1s
-> extracting sha256:7b27d0e323bc7a902add6347a485506de16a7365b6c021ac74101972d573559a / 5.5s
-> [internal] load build context
-> => transferring context: 2.23kB
[2/3] RUN pip install torch==2.4.1 torchvision==0.19.1 --index-url https://download.pytorch.org/whl/cpu
-> [3/3] COPY lambda_function.py .
-> exporting to image
-> exporting layers
-> writing image sha256:189ca2d8318119bfb17aa68f4f4bec0ae26ccbd676a6474ee398f239bc20810
-> naming to docker.io/library/lambda-image
[cloudshell-user@ip-10-134-49-69 ~]$
```

Create & Configure S3 Bucket

Create and Configure S3 Bucket

Along with JSON response (object coordinators: box) our function should store the object detected output image in a S3 bucket.



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

Account snapshot - updated every 24 hours All AWS Regions [View Storage Lens dashboard](#)

Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. [Learn more](#)

General purpose buckets | Directory buckets

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

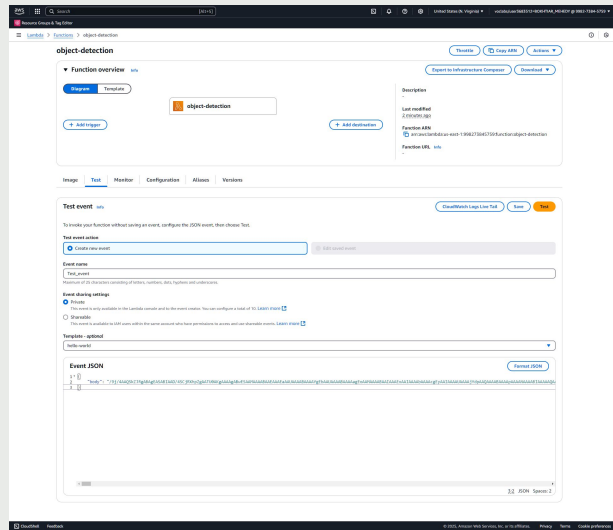
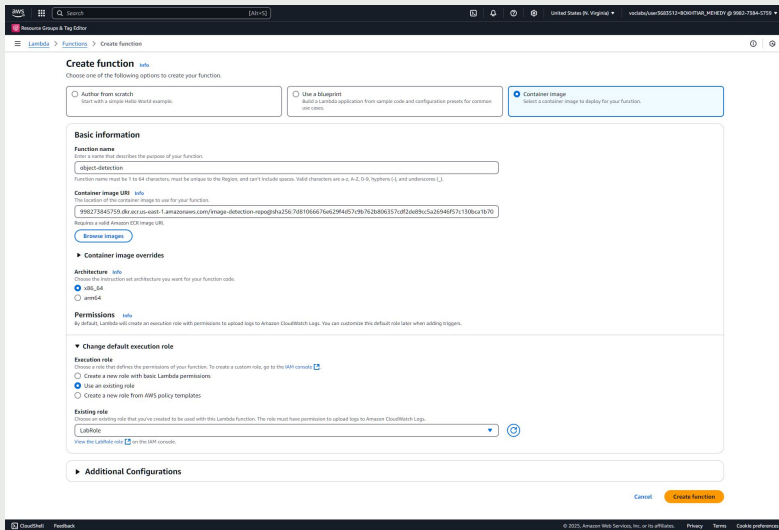
Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
<input type="radio"/> object-detected-images	US East (N. Virginia) us-east-1	View analyzer for us-east-1	January 8, 2025, 02:48:58 (UTC+01:00)

Create & Configure Lambda Function

Create and Configure AWS Lambda Function

- We created (selecting From Image) and configured a Lambda function to deploy our ML model that will detect objects in an image.
- We used an existing role to configure execution permission
- We tested the function using the Test option



Create & Configure Lambda Function

Encountered an error and changed the configurations to fix

The test was failed because of the max execution timeout, so we configured and increased the limits

Executing function: failed [logs](#)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "errorMessage": "2025-01-07T18:41:58.518Z 7f3a40eb-6663-418e-8630-bfd7c5069836 Task timed out after 3.04 seconds"
}
```

Summary

Code SHA-256
70b1066676e629f4d57c9b762b806357cdf2de89cc5a26946f57c130bca1b70a

Request ID
7f3a40eb-6663-418e-8630-bfd7c5069836

Duration
3041.29 ms

Resources configured
128 MB

Log output

Execution time
34.00 seconds ago

Function version
SLATEST

Billed duration
3000 ms

Max memory used
25 MB

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
INIT_REPORT Init Duration: 1000.13 ms Phase: Init Status: timeout
INIT_REPORT Init Duration: 3003.38 ms Phase: Invoke Status: timeout
START RequestId: 7f3a40eb-6663-418e-8630-bfd7c5069836 Version: SLATEST
2025-01-07T18:41:58.518Z 7f3a40eb-6663-418e-8630-bfd7c5069836 Task timed out after: 3.04 seconds

END RequestId: 7f3a40eb-6663-418e-8630-bfd7c5069836
REPORT RequestId: 7f3a40eb-6663-418e-8630-bfd7c5069836 Duration: 3041.29 ms Billed Duration: 3000 ms Memory Size: 128 MB Max Memory Used: 25 MB
```

aws

Search

101/112

Resource Groups & Tag Editor

Lambda > Functions > object-detector > Edit basic settings

Edit basic settings

Basic settings

Description - optional

Memory info
Your function is allocated CPU proportional to the memory configured.
10240 MB
Set memory to between 128 MB and 10240 MB

Ephemeral storage info
You can configure up to 10 GB of ephemeral storage (tmp) for your function. [View pricing](#)
10240 MB
Set ephemeral storage (tmp) to between 512 MB and 10240 MB.

SnapStart info
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is eligible to snapshot operations, review the [SnapStart compatibility considerations](#). For Python and .NET runtimes, see [pricing](#).
Selected runtimes: .NET 8 (.NET6/.NET5/.NET Core 3.1), Java 11, Java 21, Python 3.12, Python 3.10

Timeout
10 min 0 sec

Execution role
☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role

Lambda

View the Lambda role on the IAM console.

Cancel Save

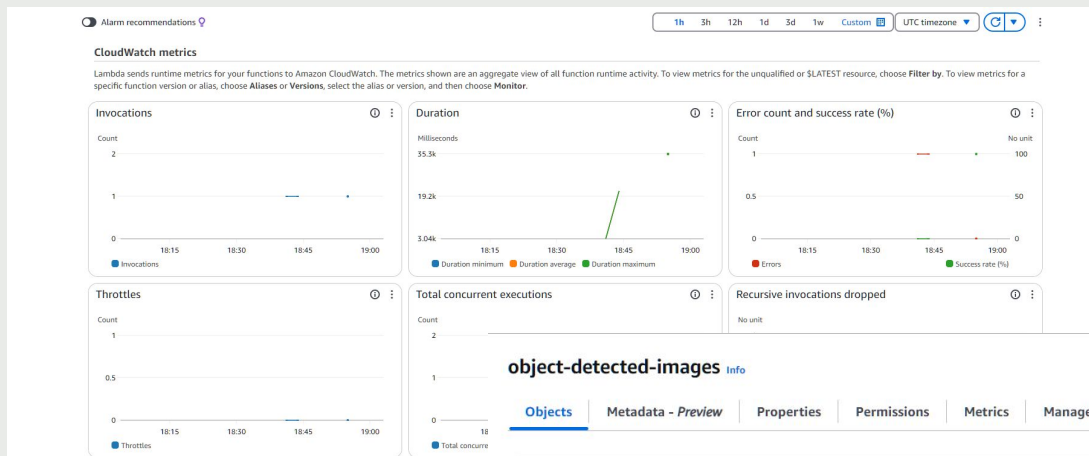
CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Contact preferences

Test Lambda Function

Test succeeded

- The configuration changes fixed the issue.
- We got a response with status code 200 and expected payload.
- We can observe the Invocation, Error, and other metrics in CloudWatch. Also the object detected image is stored in the S3 bucket.



object-detected-images [Info](#)

[Objects](#) [Metadata - Preview](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

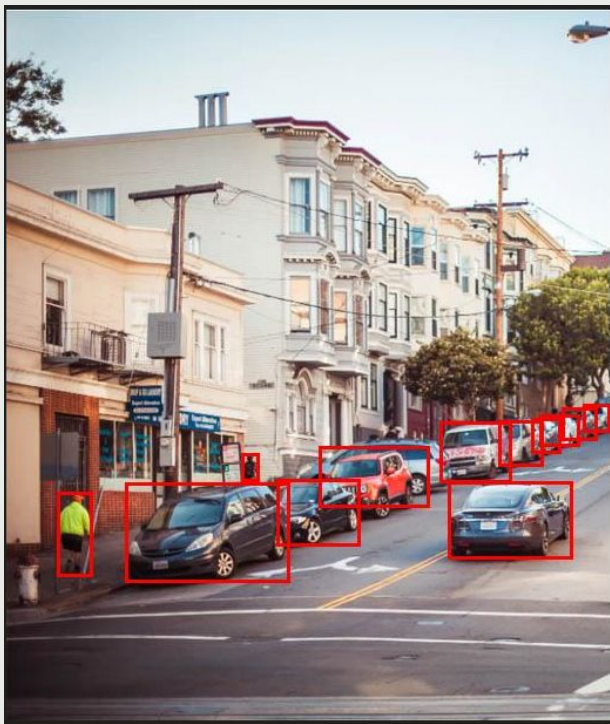
Objects (1) [Info](#)

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](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	processed_image.jpg	jpg	January 14, 2025, 13:18:44 (UTC+01:00)	55.4 KB	Standard

Test Lambda Function



object-detection

Throttle Copy ARN Actions

Export to Infrastructure Composer Download

Function overview

Diagram Template

object-detection

+ Add trigger + Add destination

Image Test Monitor Configuration Aliases Versions

Executing function: succeeded [Copy](#)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": {
    "objects": [
      {
        "bbox": [390, 530, 570, 600],
        "label": "car",
        "score": 0.95
      },
      {
        "bbox": [210, 620, 340, 680],
        "label": "car",
        "score": 0.92
      },
      {
        "bbox": [110, 660, 200, 750],
        "label": "car",
        "score": 0.88
      },
      {
        "bbox": [70, 660, 95, 750],
        "label": "person",
        "score": 0.98
      }
    ]
  }
}
```

Summary

Code SHA-256
9c447c02084961853860e75ec8c0b1ffc92d99273a0da0ff9b7933b305f21

Request ID
req03baa-5274-4222-af35-2e77e457dd9

Duration
333.1025 ms

Resources configured
10240 MB

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
2.4%
52.4%
52.4%
52.7%
52.7%
52.8%
52.8%
52.8%
52.8%
52.8%
```

Execution time
1 minute ago

Function version
\$LATEST

Billed duration
333.1 ms

Max memory used
1024 MB

Test event

CloudWatch Logs Live Tail Save Test

Create and Configure API Gateway

Planning Stress

- While planning to stress the Lambda function with multiple concurrent invocation we planned to expose the Lambda Function with REST API that we can call from a API agent like Postman.
- We'll call the REST API endpoint concurrently that will invoke the Lambda function.

Create and Configure API Gateway

We created and configured API Gateway to expose and POST method REST API endpoint.

The screenshot displays the AWS API Gateway console. On the left, the 'APIs' list shows three APIs. The main area shows the 'Create REST API' wizard. The 'API details' section has 'New API' selected. The 'API name' is 'detect-object'. The 'Description - optional' field is empty. The 'API endpoint type' is set to 'Regional'. The 'Create API' button is highlighted in orange.

API Gateway > APIs

APIs
Custom domain names [Updated](#)
Domain name access associations [New](#)
VPC links

APIs (3/3)

Name	Description	ID	Protocol

Create REST API

API details

☒ **New API**
Create a new REST API.

☐ Clone existing API
Create a copy of an API in this AWS account.

☐ Import API
Import an API from an OpenAPI definition.

☐ Example API
Learn about API Gateway with an example API.

API name
detect-object

Description - optional

API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.
Regional

[Cancel](#) [Create API](#)

Create and Configure API Gateway

Resources

Create resource

Resource details

Path: /

Resource ID: ded6ca77a5

Update documentation

Enable CORS

Methods (0)

Delete

Create method

No methods

No methods defined.

Create resource

Resource details

Path: /

Resource name: /

Create resource

Create method

Method details

Method type: POST

Integration type

Lambda function

HTTP

Mock

Lambda proxy integration

Send the request to your Lambda function as a structured event.

Lambda function

Integration timeout

Method request settings

URL query string parameters

HTTP request headers

Request body

Create method

Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

Stage: "New stage"

Stage name: lab-prod

A new stage will be created with the default settings. Edit your stage settings on the Stage page.

Deployment description

Deploy

Test API

Test API Endpoint from POSTMAN

We sent request from **Postman** to verify the API endpoint exposed in the API Gateway with the **Invoke URL** and configured resource

Successfully created deployment for detect-object. This deployment is active for lab-prod.

Stages

- lab-prod
 - /detect-object POST

Stage details

Stage name: lab-prod

Rate: 10000

Cache cluster: Inactive

Burst: 5000

Default method-level caching: Inactive

Web ACL: -

Client certificate: -

Invoke URL: <https://g2owdcdm2.execute-api.us-east-1.amazonaws.com/lab-prod>

Active deployment: 1he5ef on January 08, 2025, 01:31 (UTC+01:00)

Logs and tracing

CloudWatch logs: Inactive

Detailed metrics: Inactive

X-Ray tracing: Inactive

Data tracing: Inactive

Lambda Load - Bosco / Detect Object

POST <https://x99sryjca9.execute-api.us-east-1.amazonaws.com/lab-prod/detect-object>

Params Authorization Headers (10) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON

200 OK • 9.62 s • 1.33 KB

```
{
  "statusCode": 200,
  "body": {
    "boxes": [
      [
        101.804443359375,
        403.3091735839844,
        241.9481964111328,
        488.74859619140625
      ],
      [
        376.2754211425781,
        402.61669921875,

```

Stress Lambda Function

Stress Using Collection Runner in POSTMAN

- We decided to use Postman Collection Runner to send concurrent API request that will subsequently invoke the lambda function.
- We'll monitor mainly the (Concurrent Execution) metrics along with the Invocation, Duration and Error Count.
- Our goal was to increase the stress gradually (increasing the number of concurrent API call) and then release the stress (reducing concurrent API call number).
- We planned to the concurrent number of API: 2, 4, 8, 16 then 8, 4, 2.

Unfortunate Event

- Unfortunately we lost our AWS account twice (once not knowing the consequence of large concurrent API call, then trying to gradual increase and decrease.
- We also lost one of our group member unfortunately in the last minute because of schedule mismatch.
- **So we kept the number very conservative during the stress: 1, 2, 4, 2, 1 concurrent requests 30 seconds apart**

Stress Lambda Function

Postman Collection Runner

- We ran the postman Collection Runner to execute the pre-script

Run order

Deselect All Select All Reset

☒ POST Detect Object

Functional Performance

Choose how to run your collection

☒ Run manually
Run this collection in the Collection Runner.

☐ Schedule runs
Periodically run collection at a specified time on the Postman Cloud.

☐ Automate runs via CLI
Configure CLI command to run on your build pipeline.

Run configuration

Iterations ⓘ

Delay ⓘ
 ms

Data file ⓘ

Select File

☐ Persist responses for a session ⓘ

☐ Turn off logs during run ⓘ

> Advanced settings

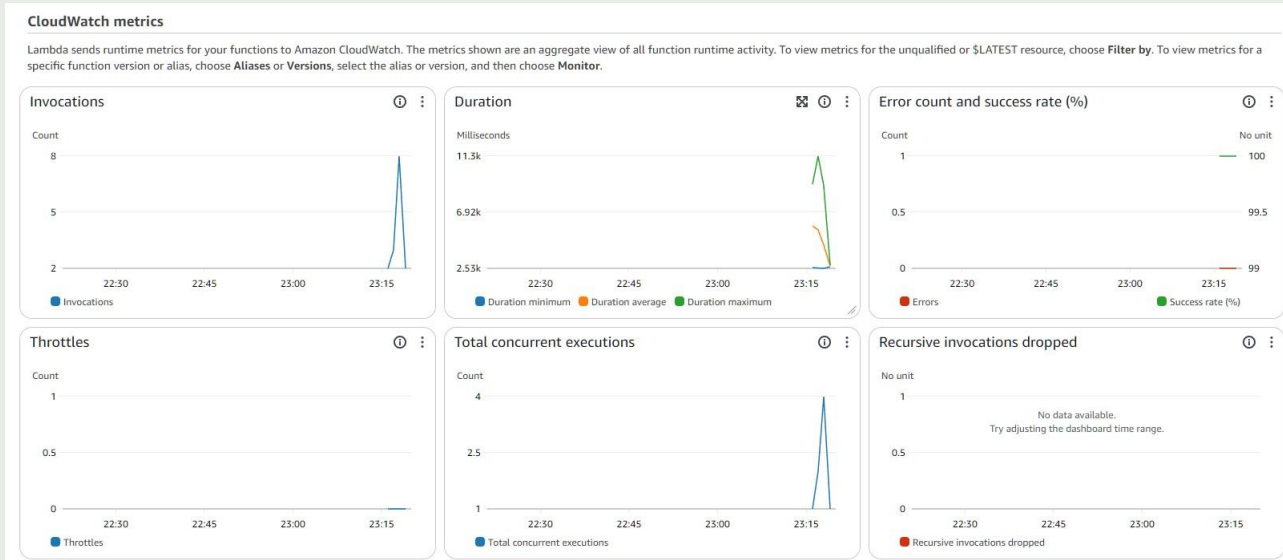
Run Lambda Load - Bosco

```
8 // Get the current iteration and calculate the request count
9 const currentIteration = pm.info.iteration;
10 const totalIterations = pm.info.iterationCount; // Automatically retrieves the total iterations
11
12 // Calculate the number of requests
13 let requestCount;
14 if (currentIteration <= Math.floor(totalIterations / 2)) {
15   // Increasing in the first half
16   requestCount = Math.pow(2, currentIteration);
17 } else {
18   // Decreasing in the second half
19   requestCount = Math.pow(2, totalIterations - currentIteration - 1);
20 }
21
22 console.log(`Iteration ${currentIteration + 1} of ${totalIterations}: Sending ${requestCount}
23 concurrent requests.`);
24
25 // Send multiple requests
26 for (let i = 0; i < requestCount; i++) {
27   pm.sendRequest({
28     url: apiUrl,
29     method: "POST",
30     header: headers,
31     body: {
32       mode: "raw",
33       raw: requestBody,
34     },
35     (err, response) => {
36       if (err) {
37         console.error(`Request ${i + 1} failed:`, err);
38       } else {
39         console.log(`Response from request ${i + 1}:`, response);
40       }
41     }
42   });
43 }
```

CloudWatch Report

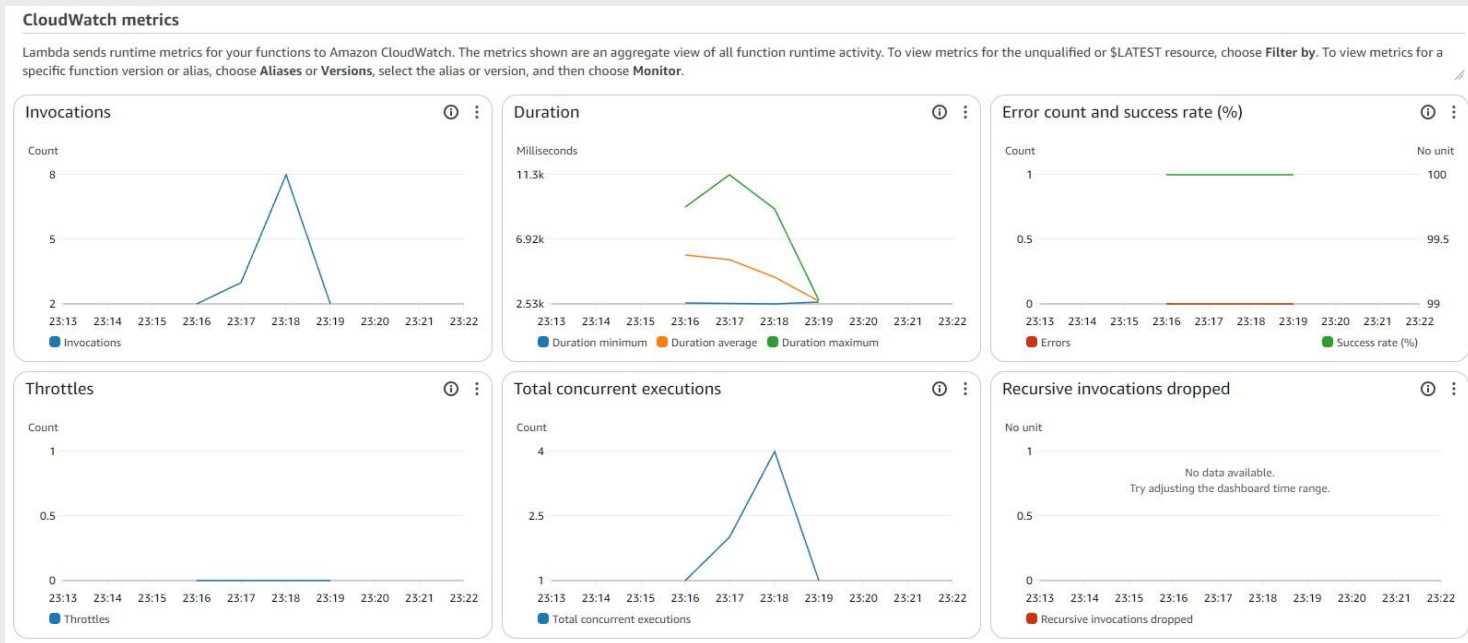
CloudWatch Metrics to Observe the Deployment Scale

- We observed that the number of **Total Concurrent Execution** increases with the higher number of Lambda Function invocation.
- Also when the invocation is decreased the **Total Concurrent Execution** number is decreased.
- Iteration delay in-between was **30 seconds**



CloudWatch Report

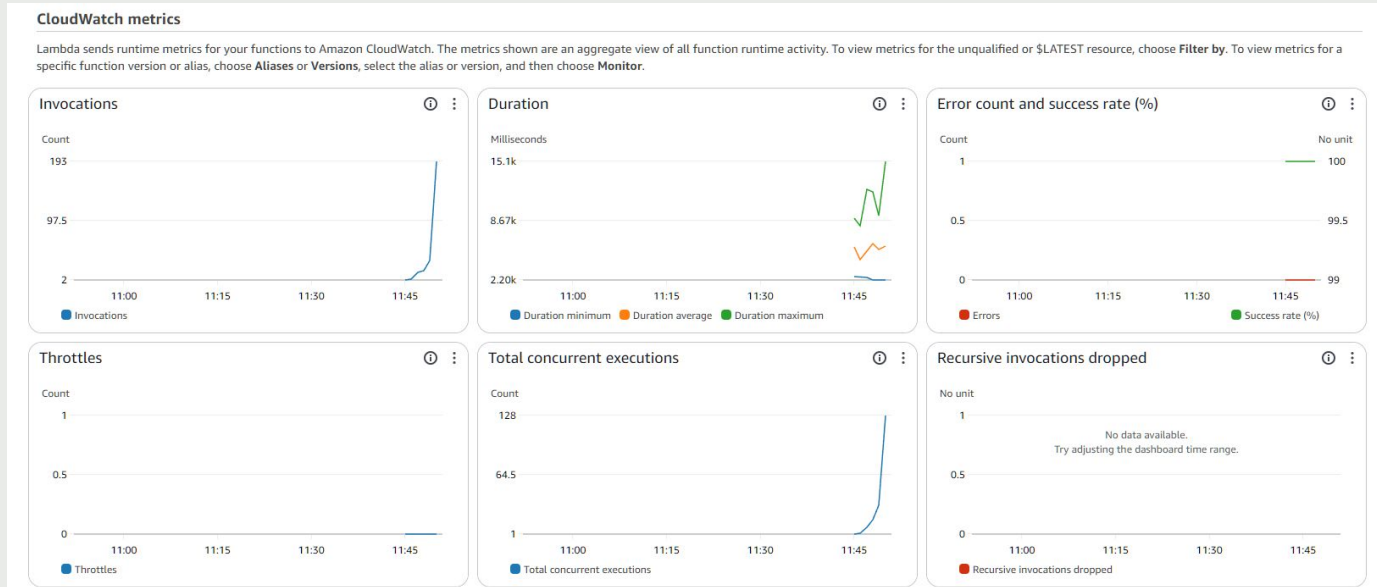
Closer Look of Deployment Scale Out and In



CloudWatch Report

CloudWatch Metrics to Observe the Deployment Scale

- Account crashed before decrement could be observed. But we can compare with the previous run.



Thank You!