MSIT3470 – Final Group Project

Real-Time Serverless Image Resizer Using AWS Lambda

Members: (Joshua Rupiya, Adnan Nazir Ahmed)

1.Executive Summary

This project delivers a complete real-time serverless image resizing system built on AWS. The app allows a user to upload an image, select a size, and receive a resized thumbnail through a secure, time-limited URL for 1 hour.

The solution demonstrates cloud-native design using managed services, Infrastructure as Code, and least-privilege security practices.

2. Core Functionality (Compute + Data + Reachability)

2.1 End-to-End Application Flow

The application provides a complete end-to-end flow using AWS-managed services, including the steps below:

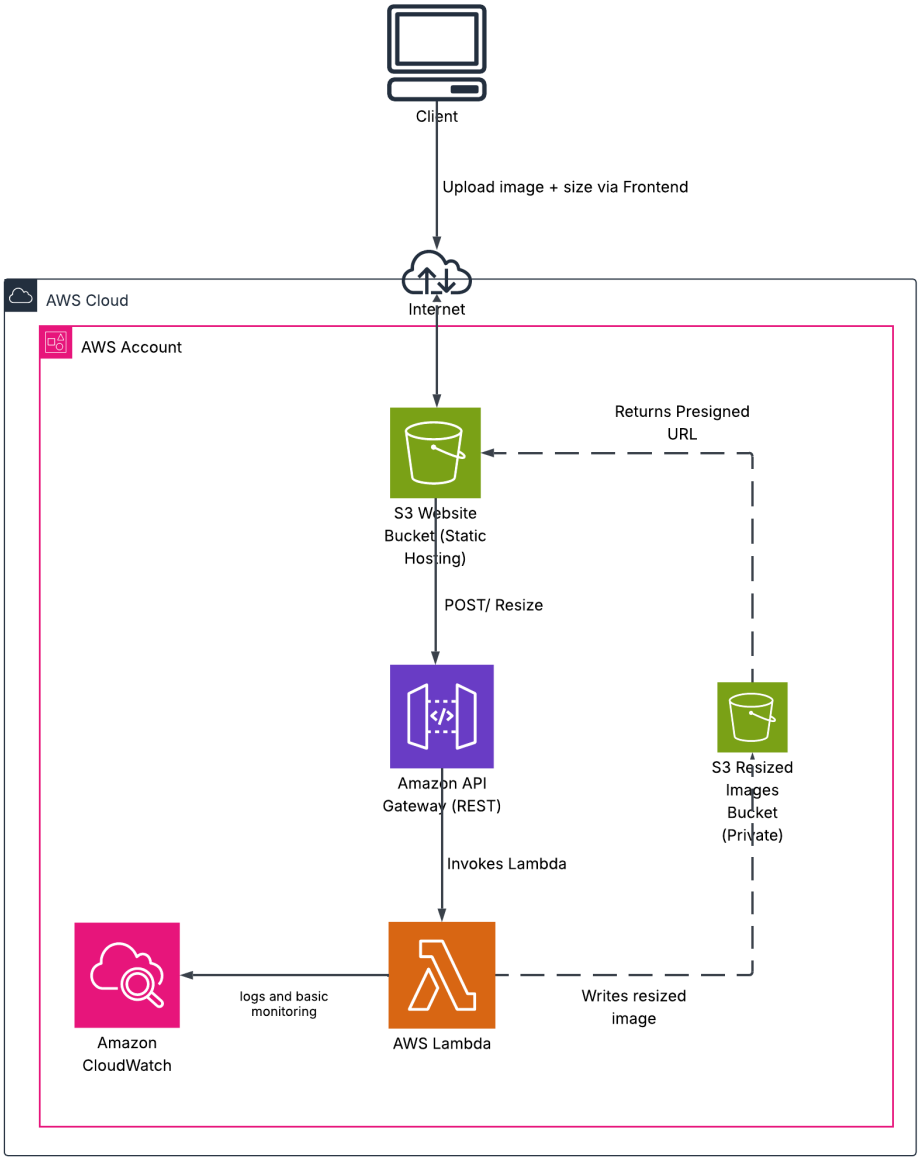
1. User accesses static frontend hosted in an S3 Website bucket.
2. User selects an image + resize option (small, medium, large).
3. Frontend sends a POST request to an API Gateway endpoint (/resize).
4. API Gateway triggers a Lambda function written in Python.
5. Lambda:
    - ❖ Decodes the uploaded image
    - ❖ Resizes it using Pillow
    - ❖ Stores it privately in a dedicated S3 bucket
    - ❖ Generates a presigned URL with controlled expiry of 1 hour.
    - ❖ Lambda returns the URL to the frontend.
6. Frontend displays:
    - ❖ Original preview
    - ❖ Resized thumbnail (actual size)
    - ❖ Secure clickable link

3. Architecture:

3.1 Architecture Diagram

The following diagram represents the actual deployed architecture and data flow of the application.

Screenshot 1. Architecture diagram of the solution:

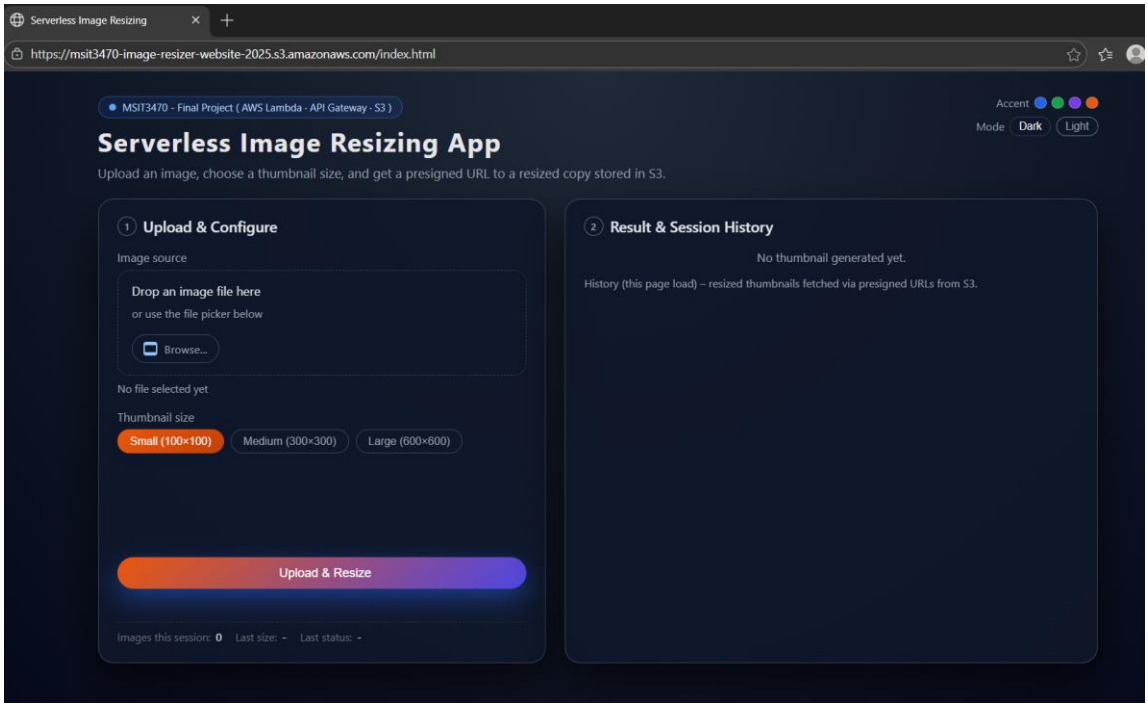


The frontend is hosted on Amazon S3 static website hosting. User requests are sent to Amazon API Gateway, which invokes an AWS Lambda function. Lambda resizes the image and stores it in a private S3 bucket, returning a presigned URL to the client.

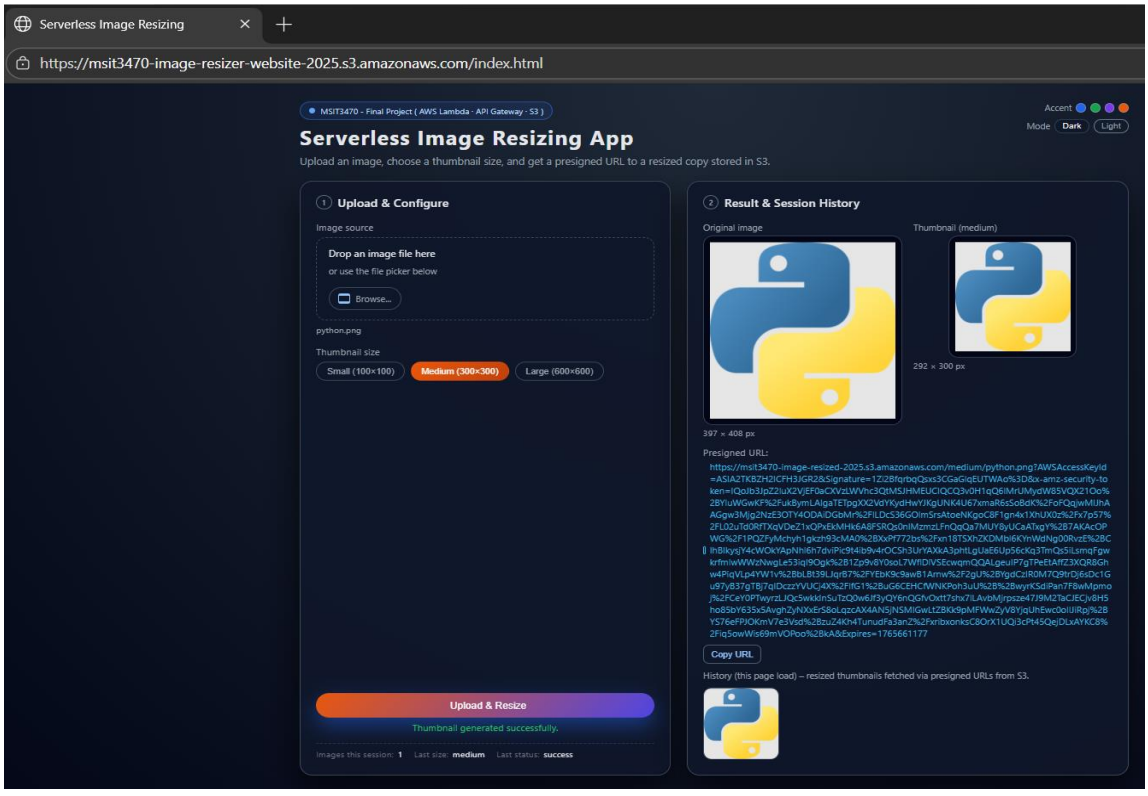
3.2 Frontend User Interface:

The diagrams below show how the application frontend interface look like as demonstrated below:

Screenshot 2. Frontend initial screen:



Screenshot 3. Successful resize result with the preview and the presigned URL:



4. Cloud infrastructure and IaC:

4.1 Cloud Services Used

- **AWS Lambda:** responsible for computing for resizing function
- **Amazon S3**: for static website + secure storage for thumbnails
- **Amazon API Gateway:** for public API endpoint
- **CloudWatch Logs:** for monitoring and debugging

All infrastructure is provisioned using Terraform, ensuring repeatability and version control as shown with terraform output command:

Screenshot 4. Showing Terraform apply output:

```
josh@DESKTOP-QJ0NFO2:~/MSIT3470-final-project-image-resizer/infra$ terraform output
api_invoke_url = "https://8xob6xirxj.execute-api.us-east-1.amazonaws.com/prod"
lambda_name = "msit3470-final-project-lambda"
resized_bucket = "msit3470-image-resized-2025"
website_bucket = "msit3470-image-resizer-website-2025"
website_url = "http://msit3470-image-resizer-website-2025.s3-website-us-east-1.amazonaws.com"
josh@DESKTOP-QJ0NFO2:~/MSIT3470-final-project-image-resizer/infra$
```

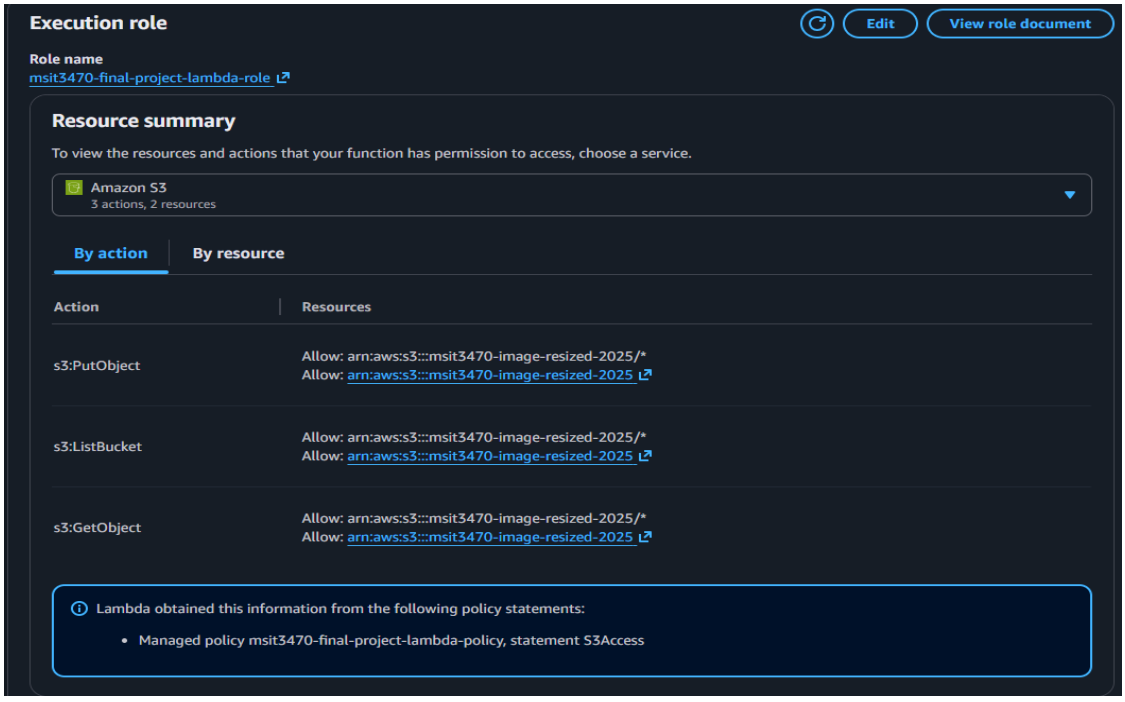
5. IAM, Networking, and Security

The application uses least-privilege IAM roles, private S3 buckets, and presigned URLs.

5.1 IAM — Least Privilege

Implemented the least privilege as shown below:

Screenshot 5. Showing lambda permission roles by action:



Lambda only has access to (s3: PutObject, s3: GetObject, s3: ListBucket) on the specific resized image bucket.

API Gateway can invoke only this Lambda.

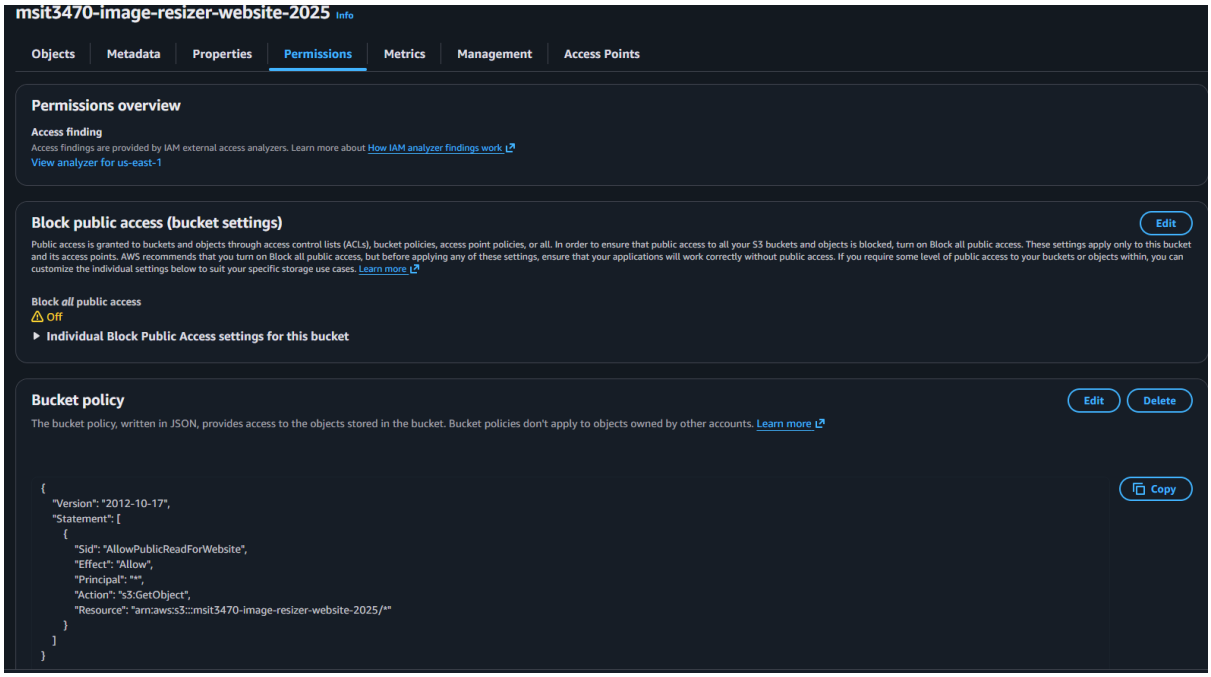
No wildcard * permissions exist in the role.

5.2 Security

Resized image bucket: private

Frontend bucket: public website hosting with restricted website policy as shown:

Screenshot 6. Showing the website policy attached:



Images are also accessed only through presigned URLs, not public URLs.

5.3 Secrets

No credentials are hardcoded, and access is strictly scoped to required services only.

Temporary access is handled using:

- ❖ AWS IAM roles
- ❖ Lambda environment variables (non-sensitive)
- ❖ No API keys or embedded credentials.

5.4 Networking

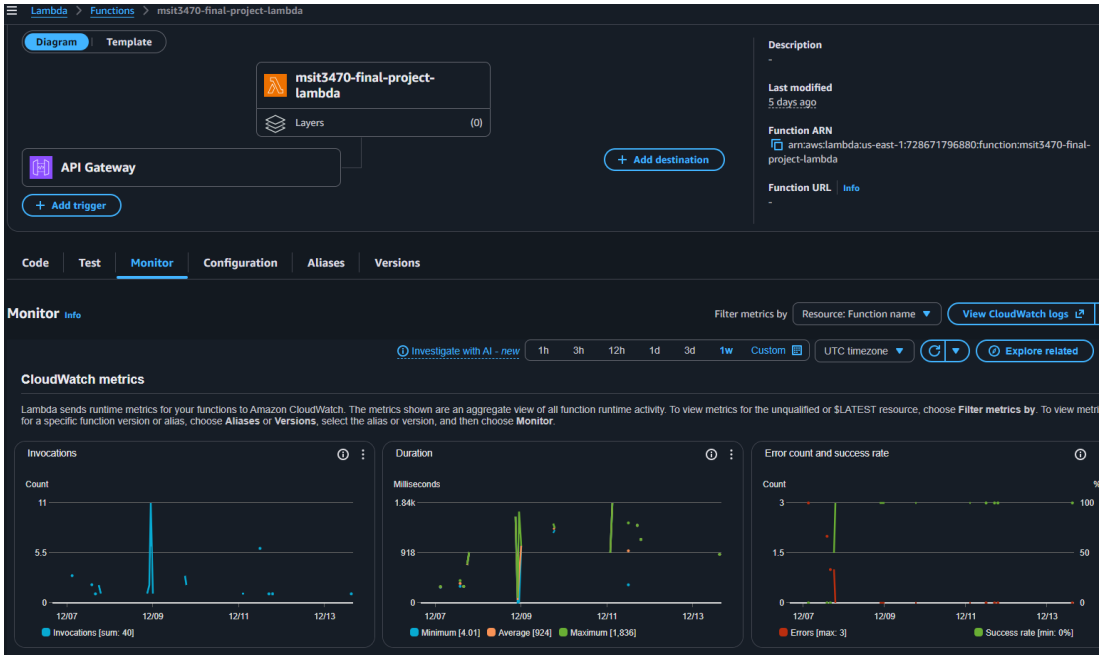
- ❖ All communication occurs via AWS-managed HTTPS endpoints.
- ❖ No exposed ports.

6 Monitoring plan and SLO:

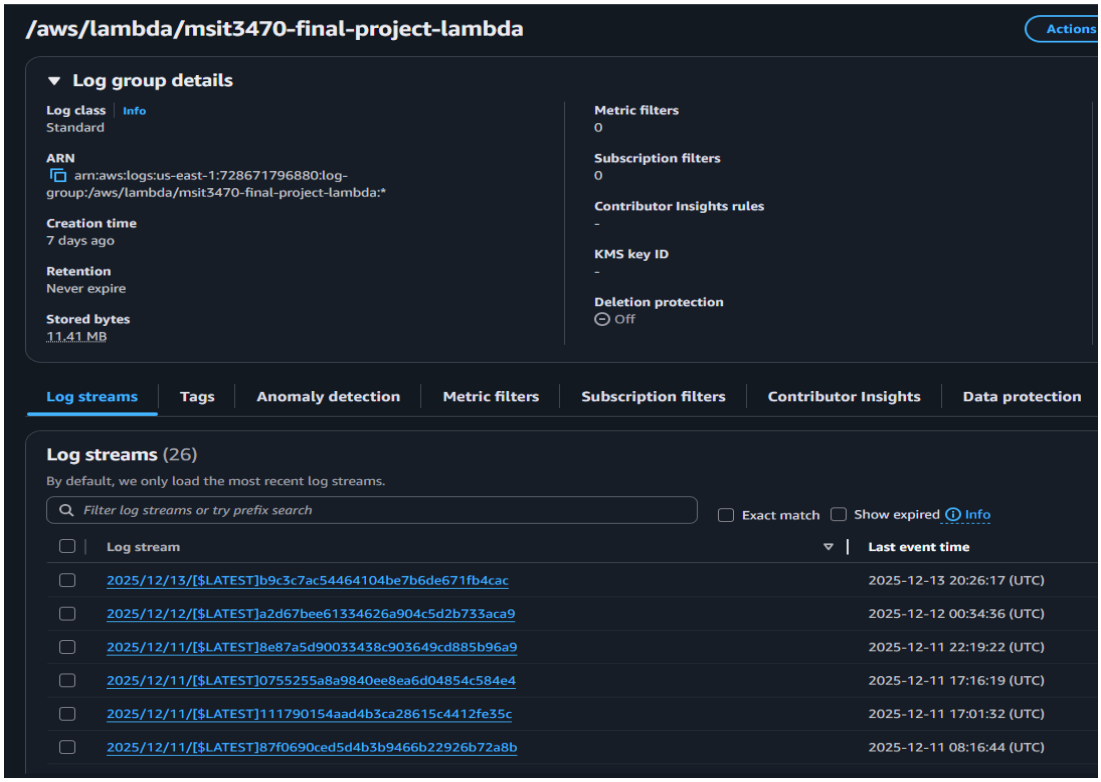
6.1 Monitoring Plan

Monitoring is handled through Amazon CloudWatch. Lambda execution logs and metrics are used to track latency, errors, and availability.

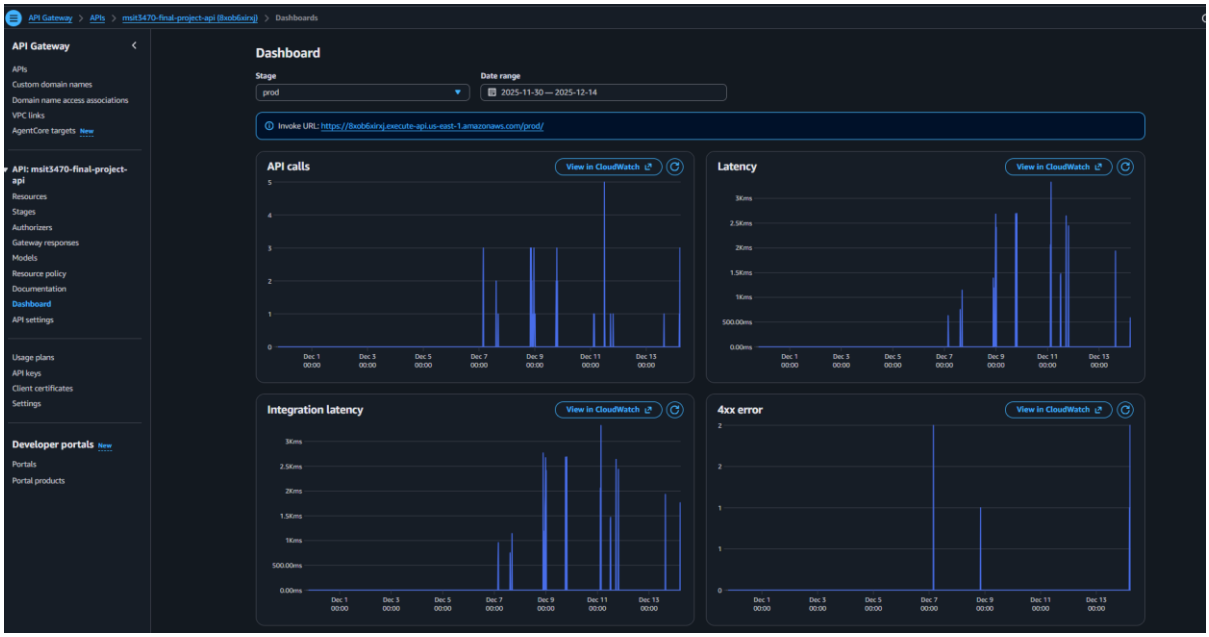
Screenshot 7. Showing CloudWatch metrics for lambda invocation:



Screenshot 8. Showing CloudWatch logs:



Screenshot 9. Showing API Gateway dashboard:



6.2 SLO (Service Level Objective)

99% of resize invocations complete within 2 seconds

Metrics observed via CloudWatch:

- Lambda invocations
- Lambda duration
- Lambda Errors
- API Gateway latency metrics

Alerts (if production)

- High error rate in Lambda
- Lambda duration p95 > 2 seconds
- 4xx/5xx spikes in API Gateway

Why these metrics?

They directly reflect:

- User experience
- System reliability
- Lambda performance under load

7. Reflection

This project successfully demonstrates the core learning outcomes of this course through the design and implementation of a functional real time serverless image resizing application on AWS. By selecting appropriate managed cloud services, the project shows a strong understanding of cloud fundamentals and how different services work together to deliver an end-to-end solution.

AWS Lambda was used as the main compute service to process image resizing requests, which reflects an understanding of serverless computing and event-driven architectures. Amazon S3 was used for storage, with separate buckets for application output, ensuring a clean and secure data flow. Access to storage was carefully controlled using IAM roles and least-privilege policies, and no sensitive credentials were stored in the codebase. This demonstrates proper attention to security and identity management best practices.

Infrastructure as Code was implemented using Terraform, allowing the entire cloud environment to be created, updated, and reproduced consistently. This approach improves reliability and reflects real-world cloud deployment practices. The project also includes a monitoring mindset by leveraging Amazon CloudWatch for logging and troubleshooting, making it easier to observe application behavior and diagnose issues.

Team collaboration was managed using Git feature branches, with clear separation of responsibilities between backend infrastructure and frontend development. This mirrors professional software development workflows and encourages accountability and collaboration.

8. Artifacts Included

- Architecture diagram
- README.md
- Terraform IaC
- Lambda source code
- Frontend source code
- Deployment scripts
- Screenshots and demonstration images (in /project/screenshot folder)