



**Dhirubhai Ambani  
University  
Technology**

Formerly DA-IICT

# **IT457 Cloud Computing Final Report**

**Date:** December 2, 2025



**CloudLens**

Serverless Content Moderation & Processing Pipeline

**Group 34**

- 202412015 Swara Chokshi
- 202412083 Nitesh Sachade
- 202412109 Stuti Shah
- 202412117 Kishan Thanki

<b>1. ABSTRACT</b>	<b>3</b>
<b>2. PROBLEM STATEMENT</b>	<b>3</b>
<b>3. PROJECT OVERVIEW</b>	<b>4</b>
3.1 PROJECT GOALS	4
3.2 PROJECT SCOPE	4
<b>4. CLOUDLENS ARCHITECTURE</b>	<b>6</b>
4.1 (3 - Tier) Architecture	6
4.2 Flow 1: Frontend Upload (User Interaction & Secure Upload)	6
4.3 Flow 2: Backend Processing (Event-Driven Pipeline)	7
<b>5. PLATFORMS &amp; TECHNOLOGIES USED</b>	<b>8</b>
5.1 Serverless Architecture	8
5.2 Amazon S3 – Object Storage + Static Website Hosting	9
5.3 AWS Lambda – Compute Engine for Core Processing	9
5.4 Amazon Rekognition – AI-Based Image Moderation	10
5.5 Amazon API Gateway – Secure API Front Door	11
5.6 AWS IAM – Identity and Access Management	11
5.7 Amazon CloudWatch – Logging, Monitoring & Dashboarding	12
<b>6. AWS SERVICES USED</b>	<b>12</b>
6.1 Amazon S3 (Simple Storage Service)	12
6.2 AWS Lambda	13
6.4 Amazon Rekognition	14
6.5 Amazon CloudWatch	15
6.7 S3 Static Website Hosting	16
6.8 AWS Lambda Layers	17
6.10 S3 Event Notifications	18
6.11 CORS Configuration (S3 + API Gateway)	19
<b>7. FRONTEND</b>	<b>20</b>
7.1 Upload Image(s)	20
7.2 Watermarked Image	21
7.3 Uploaded Image for Thumbnail is Inappropriate	22
<b>8. DASHBOARD SCREENSHOTS</b>	<b>23</b>
8.1 Lambda Function clouldens-processor	23
8.2 Lambda Function clouldens-get-upload-url	25
8.3 Buckets Metrics & Logs	26
<b>9. MEMBER CONTRIBUTIONS</b>	<b>29</b>
• Member 1 (Nitesh): Backend: Moderation & Logic	29
• Member 2 (Swara): Infrastructure, Security & Monitoring	29
• Member 3 (Kishan): Backend: Image Processing	29
• Member 4 (Stuti): Cloud-Integrated Frontend & API	29
<b>10. CONCLUSION</b>	<b>30</b>

## 1. ABSTRACT

This project implements a fully serverless, event-driven cloud application on Amazon Web Services (AWS) that automates image moderation and processing. Using Amazon S3, AWS Lambda, Amazon Rekognition, and API Gateway, the solution enables users to upload images through a static web interface and select the desired processing action: Thumbnail generation or image watermarking. The system validates images for inappropriate content using Rekognition by default for every image uploaded, ensures secure uploads via presigned URLs, and processes images using the Pillow library. Output images are stored in dedicated S3 buckets depending on the action performed and if any inappropriate image is uploaded so a warning is shown to the user and image is moved to quarantine bucket and user gets option to download processed image and if multiple images then user gets option to download as zip folder. The architecture highlights cloud-native principles such as scalability, resource elasticity, least-privilege security, and integrated monitoring through Amazon CloudWatch. Designed to operate within the AWS Free Tier or beginner cloud credits, the project provides an efficient, cost-effective demonstration of serverless computing and event-driven workflows.

## 2. PROBLEM STATEMENT

With the increasing volume of user-generated content on modern platforms, organizations require automated, scalable, and cost-efficient systems to moderate and process images in real time. Traditional server-based systems often suffer from high operational costs, limited scalability, and security challenges, especially when handling untrusted uploads.

There is a need for a **serverless cloud solution** that:

- Automatically moderates images for unsafe or inappropriate content.
- Supports image processing tasks like thumbnailing and watermarking.
- Enables secure upload mechanisms for untrusted public clients.
- Eliminates infrastructure management overhead.
- Stays cost-efficient and ideally operates within AWS Free Tier limits.

This project addresses these challenges by leveraging AWS serverless components to build an event-driven, highly scalable, secure, and low-cost image moderation and processing pipeline.

## 3. PROJECT OVERVIEW

### 3.1 PROJECT GOALS

#### 3.1.1 PRIMARY GOAL

To design, implement, test, package, deploy, and monitor a **serverless cloud application** on AWS that automatically **moderates** uploaded images for inappropriate content using Amazon Rekognition, and performs image **processing** (thumbnails, watermarking) based on user-specified metadata, ensuring all operations stay within the **AWS Free Tier** limits or utilize initial account credits.

#### 3.1.2 SECONDARY GOALS

- Demonstrate a practical understanding of event-driven architectures using S3 triggers and Lambda.
- Implement secure cloud practices including least-privilege IAM roles, S3 presigned URLs for uploads, and appropriate S3 bucket policies.
- Utilize serverless compute (AWS Lambda) to showcase elastic, scalable, and cost-effective processing.
- Successfully integrate multiple AWS services: S3, Lambda, Rekognition, and API Gateway.
- Implement application monitoring using a custom Amazon CloudWatch Dashboard.
- Gain hands-on experience in deploying and testing a multi-component cloud application.

### 3.2 PROJECT SCOPE

#### 3.2.1 In Scope

- **Frontend:** A basic HTML/CSS/JavaScript single-page application hosted via S3 Static Website Hosting.
  - Allows user selection of image files (.jpg, .png).
  - Allows users to select multiple images at once for batch processing.
  - Allows user selection of processing action (Thumbnail, Watermark), to be passed as S3 object metadata.
  - Allows users to access processed image(s) and also download option for image and if multiple images are there then user can download zip file

containing all processed images.

- The safety section shows if any of the image(s) were found inappropriate and removed automatically.
- **API Layer:** An Amazon API Gateway HTTP API endpoint.
  - Triggers a Lambda function to generate upload URLs.
  - Requires CORS configuration.
- **Upload Mechanism:** An AWS Lambda function that generates time-limited S3 presigned URLs for direct browser uploads.
- **Core Infrastructure:**
  - An S3 bucket configured to receive uploads via presigned URLs and trigger the main Lambda function. Requires CORS configuration.
  - An S3 Event Notification on the uploads bucket triggering the ModerationProcessor Lambda.
- **Backend Processing:**
  - An AWS Lambda function containing the primary application logic.
  - Reads action metadata from the triggering S3 object.
    - If unsafe: Moves original image to a dedicated S3 quarantine bucket.
    - If safe: Proceeds to complete selected action.
  - **Thumbnailing:** Uses the Pillow library to create a custom size thumbnail. Saves thumbnails to a dedicated S3 public-thumbnails bucket and also provides processed images to users on a website which is downloadable .
  - **Watermarking:** Uses the Pillow library to add a custom text watermark which we take as text input from the user. Saves watermarked images to a dedicated S3 watermarked-output bucket and also provides downloadable images.
- **Security:** Appropriate IAM Roles and Policies for both Lambda functions ensuring least-privilege access.
- **Monitoring:** A custom Amazon CloudWatch Dashboard displaying key Lambda metrics.
- **Constraint:** All AWS services utilized must operate within the limits of the AWS Free Tier or be covered by the initial \$100-\$200 AWS credits provided to new accounts.

### 3.2.2 Out of Scope

- User authentication or authorization system for the frontend.
- Processing of video files or non-image file formats.
- Advanced image editing features (e.g., filters, cropping beyond thumbnails).
- Persistent storage of moderation results or image metadata in a database (e.g.,

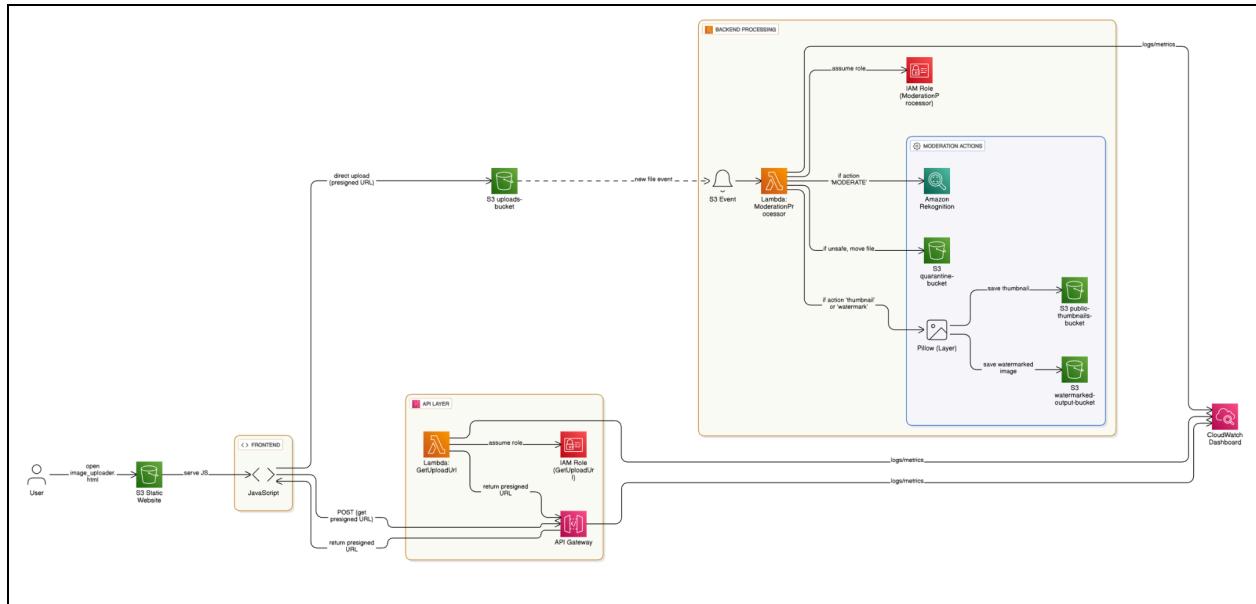
DynamoDB).

- Complex, granular error handling and retry logic for all potential failure points.
- Automated CI/CD pipeline for deployments. Manual deployment via AWS Console or CLI is acceptable.
- Advanced security measures (e.g., AWS WAF, detailed S3 Bucket Policies beyond basic access control and CORS).

## 4. CLOUDLENS ARCHITECTURE

The application is a **Serverless Event-Driven Pipeline**. It consists of two distinct, decoupled flows: the frontend upload (Flow 1) and the backend processing (Flow 2).

### 4.1 (3 - Tier) Architecture



### 4.2 Flow 1: Frontend Upload (User Interaction & Secure Upload)

This flow securely gets the file from the user's browser into S3 without exposing any credentials.

1. **User (Browser):** The user opens the **image\_uploader.html** page, which is hosted publicly from an **S3 Static Website Hosting** bucket.
2. **Frontend (JavaScript):** The user selects a file and an action. The JS code then makes a **POST** request to our **API Gateway** endpoint.
3. **API Gateway:** This service acts as the secure front door. It receives the **POST**

request and triggers **Lambda Function 1**.

4. **Lambda 1 (GetUploadUrlLambda)**: This function's only job is security. It uses its **IAM Role** (which has s3:PutObject permission) to generate a secure, temporary **S3 Presigned URL**.
5. **Return Flow**: The Presigned URL is sent back through API Gateway to the frontend JavaScript.
6. **Direct Upload**: The frontend JavaScript uses this unique URL to upload the image file directly to the private **S3 uploads-bucket**. The browser checks the bucket's **CORS policy** (which we configured to allow uploads from our S3 website URL) and completes the upload.

### 4.3 Flow 2: Backend Processing (Event-Driven Pipeline)

This flow starts automatically the moment the upload in Flow 1 finishes.

1. **S3 (uploads-bucket)**: The new file upload is detected.
2. **S3 Event Notification**: An event is immediately sent to trigger **Lambda Function 2 (ModerationProcessor)**.
3. Lambda 2 (ModerationProcessor): This is the "brain" of the application. It uses its IAM Role to:
  - **By default**
    1. Call Amazon Rekognition (DetectModerationLabels).
    2. If Unsafe: Move the original file to the S3 quarantine-bucket and return the respective response.
    3. If Safe: Proceed to the selected action.
  - **If action == 'thumbnail'**:
    1. Use the Pillow (Layer) to resize the image according to the provided size.
    2. Save the new thumbnail to the S3 public-thumbnails-bucket and show and provide a download option on the website to the user.
  - **If action == 'watermark'**:
    1. Use the Pillow (Layer) to add a watermark which was entered by the user.
    2. Save the new image to the S3 watermarked-output-bucket and show and provide a download option on the website to the user.
4. **CloudWatch (Monitoring)**: Throughout this entire process, both Lambda

functions and API Gateway are continuously sending logs and metrics, which are collected in CloudWatch and displayed on our **CloudWatch Dashboard**.

## 5. PLATFORMS & TECHNOLOGIES USED

The CloudLens application is built entirely on AWS using a serverless-first approach. Each platform and technology was selected based on clear technical requirements, cost considerations, scalability needs, and the nature of our problem (untrusted uploads, real-time moderation, low maintenance).

This section explains **why** each service was chosen, the **benefits it provides**, and the **problem it solves** in our architecture.

### 5.1 Serverless Architecture

We adopted a **serverless architecture** because the application workload is fully event-driven and does not require long-running servers. Manual provisioning, configuration, patching, and scaling of traditional virtual machines (EC2/VMs) would add unnecessary complexity and cost.

#### Why Serverless?

- **Automatic Scaling:** Lambda automatically adjusts capacity based on the number of uploads, ideal for unpredictable traffic.
- **Zero Server Maintenance:** No operating system updates, patching, or uptime management required.
- **Pay-per-Use:** We only pay when a function is executed or when storage/API calls are made.
- **Built for Event-Driven Workflows:** S3 upload → triggers → moderation → processing flow fits perfectly into serverless design.

#### Need for Serverless

The workload consists of short, image-based tasks that run only when triggered. Thus, a fully serverless model ensures maximum efficiency and zero idle costs.

## 5.2 Amazon S3 – Object Storage + Static Website Hosting

### Why S3 for Storage?

- S3 is a reliable, scalable, and cost-effective storage service for large volumes of images.
- Supports **direct browser uploads using presigned URLs**, making it secure for untrusted public users.
- Provides event notifications that trigger downstream processing automatically.
- Stores original uploads, processed images (thumbnails/watermarked), and unsafe images in separate buckets.

### Why S3 Static Website Hosting for Frontend?

- Simple HTML/CSS/JS frontend does not require a backend server.
- Fast global access with S3 + CloudFront (optional).
- Extremely low cost (only storage charges).
- Fully serverless hosting with zero maintenance.

### Benefit

S3 centralizes all static content and user uploads under the same serverless environment and integrates seamlessly with other AWS services.

## 5.3 AWS Lambda – Compute Engine for Core Processing

### Why Lambda instead of EC2?

Lambda	EC2
Auto-scaled	Must manually scale
Pay only per request	Pay 24/7 even when idle

No server management	Requires OS maintenance
Fast deployment	Longer setup time
Event-driven	Not natively event-triggered

Because our functions only run when an image arrives, Lambda is an ideal choice.

### Lambda's Roles in CloudLens

- Generate presigned S3 upload URLs
- Moderate images using Rekognition
- Process images using Pillow (Lambda Layer)

### Motivation

Lambda dramatically simplifies architecture, reduces cost, and eliminates server configuration overhead.

## 5.4 Amazon Rekognition – AI-Based Image Moderation

### Why Rekognition Instead of Building a Custom ML Model?

Building a custom CV/ML model would require:

- Dataset collection
- Model training
- Infrastructure for inference
- Continuous tuning and retraining

Rekognition provides:

- Pre-trained moderation models
- High accuracy for explicit, unsafe, or inappropriate content
- Near real-time inference
- Fully managed service

## Need

The goal of the project is not ML development but efficient moderation. Rekognition allows us to integrate industry-grade AI moderation instantly.

## 5.5 Amazon API Gateway – Secure API Front Door

### Why API Gateway?

- Acts as a secure interface between frontend and Lambda.
- Prevents exposing Lambda publicly.
- Enforces HTTP method validation and CORS restrictions.
- Handles throttling, access control, and request routing.
- Integrates natively with Lambda.

### Why Needed for This Project?

The frontend must request a **temporary presigned URL** for secure uploads. Exposing S3 directly would be unsafe.

API Gateway ensures:

- No user ever receives AWS credentials
- Only signed, time-limited upload URLs are issued
- Frontend can safely upload images without exposing private buckets

## 5.6 AWS IAM – Identity and Access Management

### Why IAM?

- Enforces **least privilege** permissions for Lambda and S3.
- Prevents unauthorized access to buckets.
- Ensures presigned URLs only allow controlled uploads.

## Need

Because the application deals with public uploads, strict security boundaries are essential.

## 5.7 Amazon CloudWatch – Logging, Monitoring & Dashboarding

### Why CloudWatch?

CloudWatch gives complete visibility into:

- Lambda invocation metrics
- Errors, throttles, and cold starts
- Rekognition API call logs
- API Gateway traffic

### Need

Real-time monitoring helps verify correct system behavior, detect failures, and demonstrate operational insights for the project evaluation.

## 6. AWS SERVICES USED

### 6.1 Amazon S3 (Simple Storage Service)

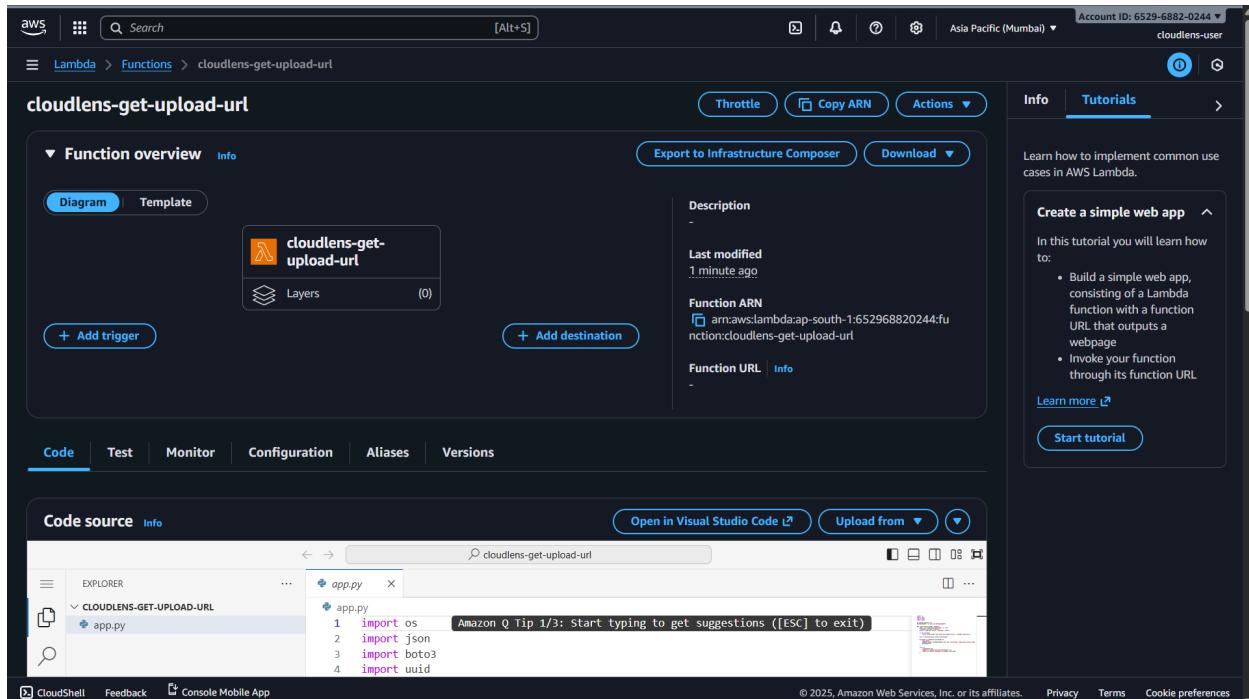
- Used to store uploaded images, processed outputs (thumbnails/watermarked), and unsafe images in separate dedicated buckets.

The screenshot shows the AWS S3 Buckets page. On the left, there's a navigation sidebar with options like Buckets, Access management and security, and Storage management and insights. The main area displays a table of General purpose buckets, each with a name, AWS Region (Asia Pacific (Mumbai)), and Creation date (November 20, 2025). To the right, there are two callout boxes: one for 'Account snapshot' which provides visibility into storage usage and activity trends, and another for 'External access summary - new' which helps identify bucket permissions from other AWS accounts.

Name	AWS Region	Creation date
cloudlens-g34-quarantine	Asia Pacific (Mumbai) ap-south-1	November 20, 2025, 13:14:51 (UTC+05:30)
cloudlens-g34-static	Asia Pacific (Mumbai) ap-south-1	November 20, 2025, 13:14:53 (UTC+05:30)
cloudlens-g34-thumbnails	Asia Pacific (Mumbai) ap-south-1	November 20, 2025, 13:14:46 (UTC+05:30)
cloudlens-g34-uploads	Asia Pacific (Mumbai) ap-south-1	November 20, 2025, 13:14:44 (UTC+05:30)
cloudlens-g34-watermarked	Asia Pacific (Mumbai) ap-south-1	November 20, 2025, 13:14:48 (UTC+05:30)

## 6.2 AWS Lambda

- Serverless computer used for generating presigned URLs and processing uploaded images (moderation + editing).



The screenshot shows the AWS Lambda console for the 'cloudlens-get-upload-url' function. The function overview section includes a thumbnail of the Lambda icon, a description field (empty), and a last modified time of '1 minute ago'. The Function ARN is listed as `arn:aws:lambda:ap-south-1:652968820244:function:cloudlens-get-upload-url`. The Function URL is also present. On the right, there's a 'Tutorials' sidebar with a 'Create a simple web app' section and a 'Start tutorial' button.

**Code source** tab:

```
app.py
1 import os
2 import json
3 import boto3
4 import uuid
```

**Code source** tab:

```
app.py
1 import os
2 import json
3 import io
4 from PIL import Image, ImageDraw, ImageFont
```

## 6.3 Amazon API Gateway (HTTP API)

- Provides a secure public API endpoint that triggers the Lambda function to generate presigned S3 upload URLs.

The screenshot shows the AWS API Gateway console. At the top, there's a green success message: "Successfully created API cloudlens-upload-api (cyxp8zul8c)". Below it, the "APIs (1/1)" section displays a single API entry:

Name	Description	ID	Protocol	API endpoint type	Created
cloudlens-upload-api		cyxp8zul8c	HTTP	Regional	2025-11-20

At the bottom of the page, there are links for CloudShell, Feedback, Console Mobile App, and standard footer links for Privacy, Terms, and Cookie preferences.

## 6.4 Amazon Rekognition

- Performs automated image moderation to detect unsafe or inappropriate content.

The screenshot shows the AWS CloudWatch Log groups interface. The left sidebar shows navigation options like Alarms, AI Operations, GenAI Observability, Application Signals (APM), Infrastructure Monitoring, Logs, Log groups, Log Anomalies, Live Tail, Logs Insights (New), Contributor Insights, and Metrics. The main area is titled "Log events" and shows log entries for a Lambda function named "cloudlens-processor".

Time	Event
2025/12/01/[\$LATEST]833142eb8c87491ba404f6752a0126f0	INIT_START Runtime Version: python:3.13.v74    Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:c7da947c7c5e914eae22806e9f241ceb83a56a93ebcae5588c99b18c38636ce
	START RequestId: 01273756-2d41-4cac-8d9a-bc4326ffdd82 Version: \$LATEST
	Processing: watermark/443ffeb6-c7ef-4bf5-a348-1bd8dd80faee_1000430771.jpg   Action: watermark   Meta: {'custom-text': 'Warning'}
	⚠️ UNSAFE CONTENT DETECTED: watermark/443ffeb6-c7ef-4bf5-a348-1bd8dd80faee_1000430771.jpg
	END RequestId: 01273756-2d41-4cac-8d9a-bc4326ffdd82
	REPORT RequestId: 01273756-2d41-4cac-8d9a-bc326ffdd82 Duration: 383.10 ms    Billed Duration: 1013 ms    Memory Size: 512 MB    Max Memory Used: 102 MB Init Duration: 629.78 ms

At the bottom of the page, there are links for CloudShell, Feedback, Console Mobile App, and standard footer links for Privacy, Terms, and Cookie preferences.

## 6.5 Amazon CloudWatch

- Collects logs and metrics from Lambda and API Gateway, and displays them through a custom monitoring dashboard.

The screenshot shows the AWS CloudWatch Log streams page. The left sidebar has a 'Logs' section with 'Log groups'. The main area shows a table of log streams with columns for 'Log stream' and 'Last event time'. The log streams are listed in descending order of last event time.

Log stream	Last event time
2025/12/01/[SLATEST]8335142eb8c87491ba404f6752a0126f0	2025-12-01 10:25:04 (UTC)
2025/12/01/[SLATEST]36e90220e98d47eea96a8f312e669a9a	2025-12-01 10:17:07 (UTC)
2025/12/01/[SLATEST]05f15a61802649c68b5cf9b597c78294	2025-12-01 10:03:03 (UTC)
2025/11/30/[SLATEST]1cb691b37e3243a81bd88ce412f47cf	2025-11-30 20:36:40 (UTC)
2025/11/30/[SLATEST]28647cd07fc74999815f9071bf439b2b	2025-11-30 20:31:43 (UTC)
2025/11/30/[SLATEST]ed1679ba59024980aef105eaa7b9ddd	2025-11-30 20:23:16 (UTC)
2025/11/30/[SLATEST]8c2861d05d4b484db06b071aae38f61d	2025-11-30 20:21:21 (UTC)
2025/11/30/[SLATEST]13c5f1233f3246a9a2c184cf87296342	2025-11-30 20:03:08 (UTC)
2025/11/30/[SLATEST]a5fe188a14ba4eb69b0b4fc2ef1bc372	2025-11-30 19:53:02 (UTC)
2025/11/30/[SLATEST]3b036e6da0c64075a21b595afe7c9baa	2025-11-30 19:18:46 (UTC)
2025/11/20/[SLATEST]550b1bc695124b2cbf619ca1946f9fde	2025-11-20 18:58:31 (UTC)
2025/11/20/[SLATEST]1c4953ec7eac64277a68b62dec1782693	2025-11-20 18:51:42 (UTC)

## 6.6 AWS IAM (Identity and Access Management)

- Provides fine-grained least-privilege permissions for Lambdas, S3 buckets, and API Gateway.

The screenshot shows the AWS IAM Roles page. The left sidebar has an 'IAM' section. The main area shows a table of roles with columns for 'Role name', 'Trusted entities', and 'Last activity'. Below the table, there are sections for 'Access AWS from your non AWS workloads', 'X.509 Standard', and 'Temporary credentials'.

Role name	Trusted entities	Last activity
AWSServiceRoleForResourceExplorer	AWS Service: resource-explorer-2 (Service-Linker)	14 minutes ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linker)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linker)	-
GetUploadUrlLambdaRole	AWS Service: lambda	-
ProcessingLambdaRole	AWS Service: lambda	-

## 6.7 S3 Static Website Hosting

- Hosts the frontend web interface (HTML/CSS/JS) for user uploads and interactions.

The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > cloudlens-g34-static`. The left sidebar has sections for Buckets, Access management and security, and Storage management and insights. The main area is titled "cloudlens-g34-static Info" and shows the "Objects" tab selected. It lists four objects: `cloudlens_logo.png` (png), `index.html` (html), `script.js` (js), and `style.css` (css). Each object has its name, type, last modified date, size, and storage class (Standard). A search bar and a "Show versions" button are also present.

The screenshot shows the AWS S3 console with the same path as the previous screenshot. The left sidebar includes a "Requester pays" section which is currently disabled. The main area has a "Static website hosting" section. It recommends using AWS Amplify Hosting for static website hosting and provides a link to "Create Amplify app". Below this, there's a "S3 static website hosting" section where "Enabled" is selected. The "Hosting type" is set to "Bucket hosting". At the bottom, the "Bucket website endpoint" is listed as `http://cloudlens-g34-static.s3-website.ap-south-1.amazonaws.com`.

## 6.8 AWS Lambda Layers

- Used to package and attach the Pillow image-processing library for thumbnailing and watermarking.

The image shows two screenshots of the AWS Lambda console. The top screenshot displays the 'Layers' section, showing a single layer named 'cloudlens-pillow-layer' with version 1, which is a 'Pillow layer for CloudLens project' compatible with Python 3.11. The bottom screenshot shows the 'Functions' section for a function named 'cloudlens-processor'. It includes a 'Function overview' tab with a diagram showing the function's layers, and a 'Code source' tab where the code file 'app.py' is visible, containing the following Python code:

```
import os
import boto3
import io
from PIL import Image, ImageDraw, ImageFont
```

## 6.9 Amazon CloudWatch Logs

- Stores detailed execution logs from both Lambda functions for debugging and monitoring.

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Alarms, AI Operations, GenAI Observability, Application Signals (APM), Infrastructure Monitoring, Logs (selected), Log groups, Log Anomalies, Live Tail, Logs Insights, and Metrics. The main content area displays 'Log events' for the path /aws/lambda/cloudlens-get-upload-url on December 1, 2025. The log entries show various Lambda requests with details like RequestId, Duration, Billed Duration, Memory Size, and Max Memory Used. A search bar at the top allows filtering by search terms, and a time range selector provides options for 1m, 30m, 1h, 12h, Custom, and UTC timezone.

## 6.10 S3 Event Notifications

- Automatically triggers the ModerationProcessor Lambda when a new file is uploaded.

The screenshot shows the AWS S3 Bucket Properties page for 'cloudlens-g34-uploads'. A green success message box states 'Successfully created event notification "UploadsToProcessor". Operation successfully completed.' Below this, a blue info message box says 'You can view and configure CloudTrail data events for Amazon S3 bucket object-level operations in the AWS CloudTrail console.' A 'AWS CloudTrail' button is also present. The 'Event notifications' section lists a single rule named 'UploadsToProcessor' for 'All object create events' using 'Amazon EventBridge' as the destination type, with 'Lambda function' as the destination. Other sections visible include 'Transfer acceleration' (disabled) and 'Object Lock' (disabled). The bottom of the page includes standard AWS navigation links: CloudShell, Feedback, Console Mobile App, Privacy, Terms, and Cookie preferences.

## 6.11 CORS Configuration (S3 + API Gateway)

- Ensures secure cross-origin requests between the static website and API endpoints.

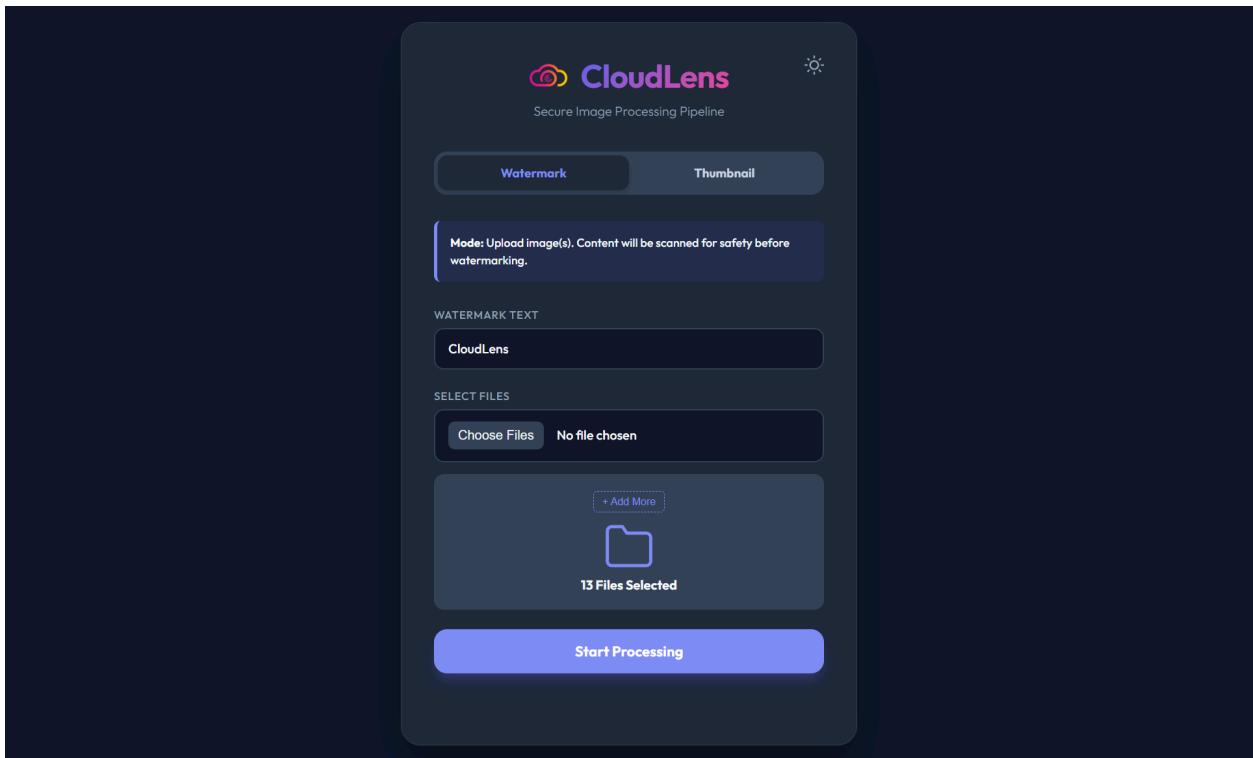
The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > cloudlens-g34-uploads`. In the top navigation bar, the account ID is listed as `6529-6882-0244`, the region is `Asia Pacific (Mumbai)`, and the user is `cloudlens-user`. The main content area is titled "Cross-origin resource sharing (CORS)" and contains a JSON configuration block:

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "PUT",  
      "POST",  
      "GET",  
      "HEAD"  
    ],  
    "AllowedOrigins": [  
      "*"  
    ],  
    "ExposeHeaders": []  
  }  
]
```

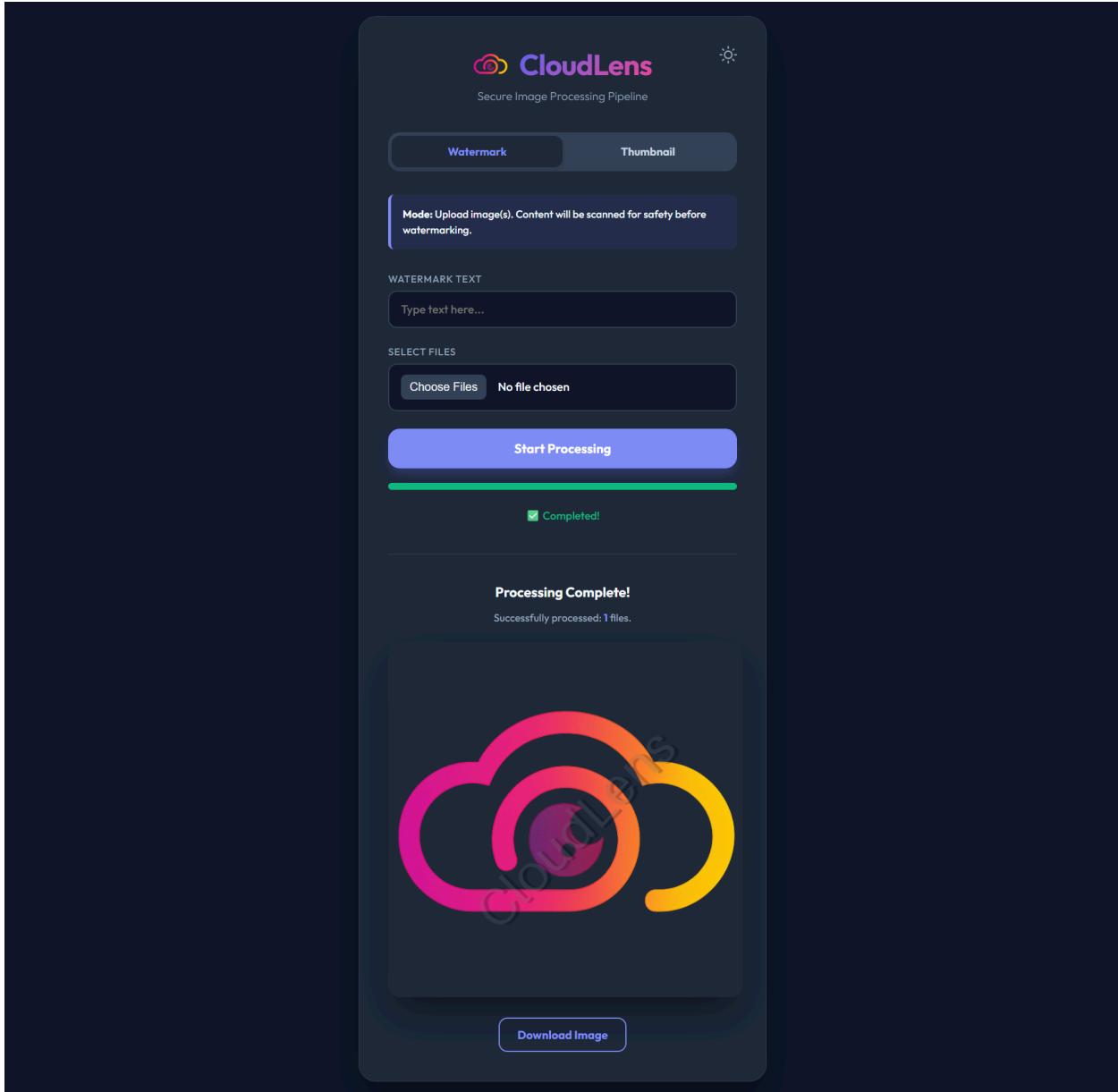
Below the configuration, there are "Edit" and "Copy" buttons. At the bottom of the page, there are links for CloudShell, Feedback, and Console Mobile App, along with copyright information: `© 2025, Amazon Web Services, Inc. or its affiliates.` and links for Privacy, Terms, and Cookie preferences.

## 7. FRONTEND

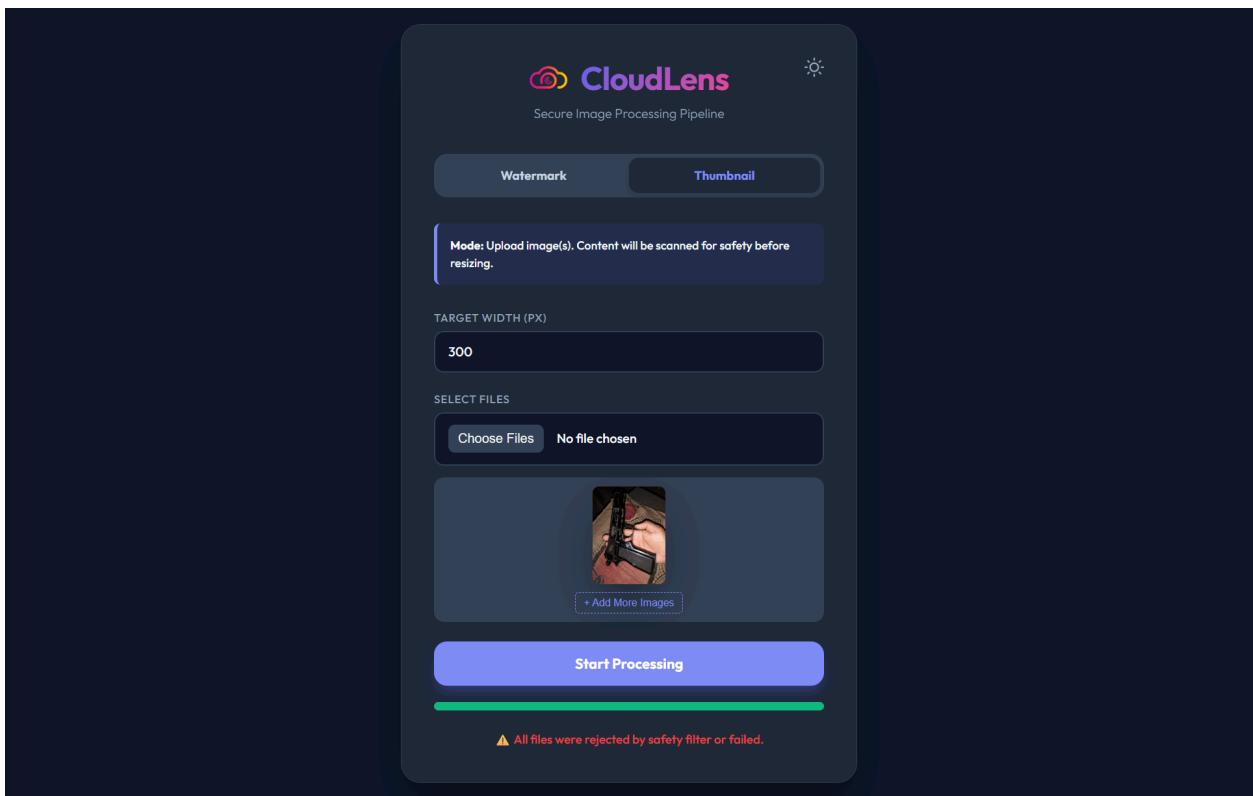
### 7.1 Upload Image(s)



## 7.2 Watermarked Image

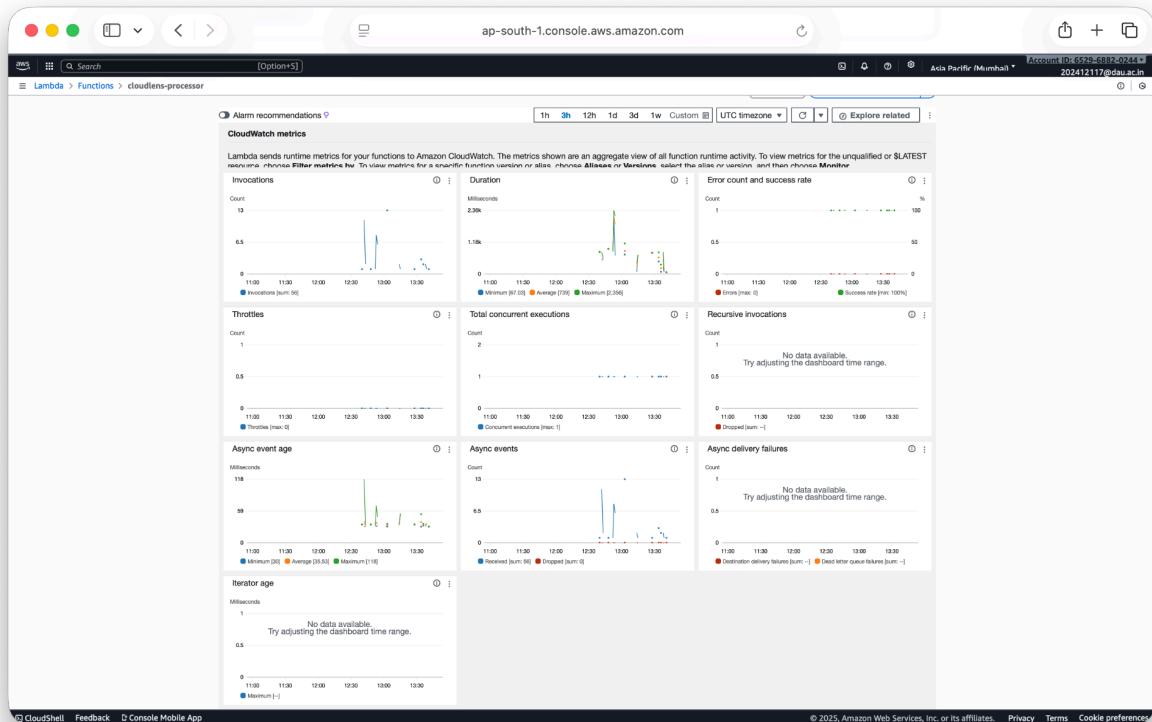


### 7.3 Uploaded Image for Thumbnail is Inappropriate



# 8. DASHBOARD SCREENSHOTS

## 8.1 Lambda Function cloudlens-processor



Screenshot of the AWS Lambda CloudWatch Logs interface for the 'clouditems-processor' function.

**CloudWatch Logs**

Lambda logs all requests handled by your function and automatically stores logs generated by your code through Amazon CloudWatch Logs. To validate your code, instrument it with custom log statements. The following tables list the most recent and most expensive function invocations across all function activity. To view logs for a specific function version or alias, visit the CloudWatch Metrics console.

**Recent invocations**

#	Timestamp	RequestId	LogStream	BilledDuration	MemorySize	MemoryUsedInMB
1	2025-12-02T13:41...	ef93e2f3-2bd3-4877-9527-...	2025/12/02/[SLATEST]603d5277ef6426a...	61.03	68.0	512.0
2	2025-12-02T13:41...	ef93e2f3-2bd3-4877-9527-...	2025/12/02/[SLATEST]603d5277ef6426a...	61.03	75.0	512.0
3	2025-12-02T13:38...	25fcf5ec-23df-4bc7-bcd4-e...	2025/12/02/[SLATEST]603d5277ef6426a...	259.64	268.0	512.0
4	2025-12-02T13:38...	732arfdu-f0b8-4972-8143-b...	2025/12/02/[SLATEST]603d5277ef6426a...	886.27	897.0	512.0
5	2025-12-02T13:38...	732arfdu-f0b8-4972-8143-b...	2025/12/02/[SLATEST]603d5277ef6426a...	890.00	949.0	512.0
6	2025-12-02T13:36...	c8a0a0b9-e5fd-47e4-94fc-4...	2025/12/02/[SLATEST]603d5277ef6426a...	81.52	685.0	512.0
7	2025-12-02T13:34...	240b229b-30d0-47a3-959c-0...	2025/12/02/[SLATEST]07c918f18b184...	568.78	561.0	512.0
8	2025-12-02T13:34...	240b229b-30d0-47a3-959c-0...	2025/12/02/[SLATEST]07c918f18b184...	462.00	500.0	512.0
9	2025-12-02T13:34...	d443a3b9-308c-4baf-c936-3...	2025/12/02/[SLATEST]07c918f18b184...	895.34	1444.0	512.0

**Most expensive invocations in GB-seconds (memory assigned \* billed duration)**

#	Timestamp	RequestId	LogStream	BilledDuration	MemorySize	BilledDurationInGBSeconds
1	2025-12-02T12:53...	efca042e-1479-4f76-8e9b-b...	2025/12/02/[SLATEST]32bf5169777c4e6...	2357	512	1.1785
2	2025-12-02T12:53...	c3bf2d67-2e16-4068-971d-b...	2025/12/02/[SLATEST]32bf5169777c4e6...	2188	512	1.094
3	2025-12-02T12:53...	efca042e-1479-4f76-8e9b-b...	2025/12/02/[SLATEST]32bf5169777c4e6...	2148	512	1.075
4	2025-12-02T12:53...	a91a087f-0319-40cf-9921-d...	2025/12/02/[SLATEST]32bf5169777c4e6...	2137	512	1.0685
5	2025-12-02T12:54...	96d877ee-9ecc-4520-af25-8...	2025/12/02/[SLATEST]32bf5169777c4e6...	2133	512	1.0665
6	2025-12-02T12:53...	20553531-0e35-426f-b8d7-d...	2025/12/02/[SLATEST]32bf5169777c4e6...	2122	512	1.061
7	2025-12-02T12:53...	20553531-0e35-426f-b8d7-d...	2025/12/02/[SLATEST]32bf5169777c4e6...	2118	512	1.060
8	2025-12-02T12:53...	efbb6644-fc15-4d9a-858d-d...	2025/12/02/[SLATEST]32bf5169777c4e6...	2098	512	1.049
9	2025-12-02T12:54...	0e81181b-2e1c-434d-8176-5...	2025/12/02/[SLATEST]32bf5169777c4e6...	2087	512	1.045

**AWS X-Ray**

CloudWatch integrates with AWS X-Ray to provide an end-to-end view of your application. These integrated services provide a service map, which displays your service endpoints and resources as nodes and highlights the traffic between and among the each node and its connections.

**Service map**

No services

Try adjusting the time range

No node selected

Select a node to see its details

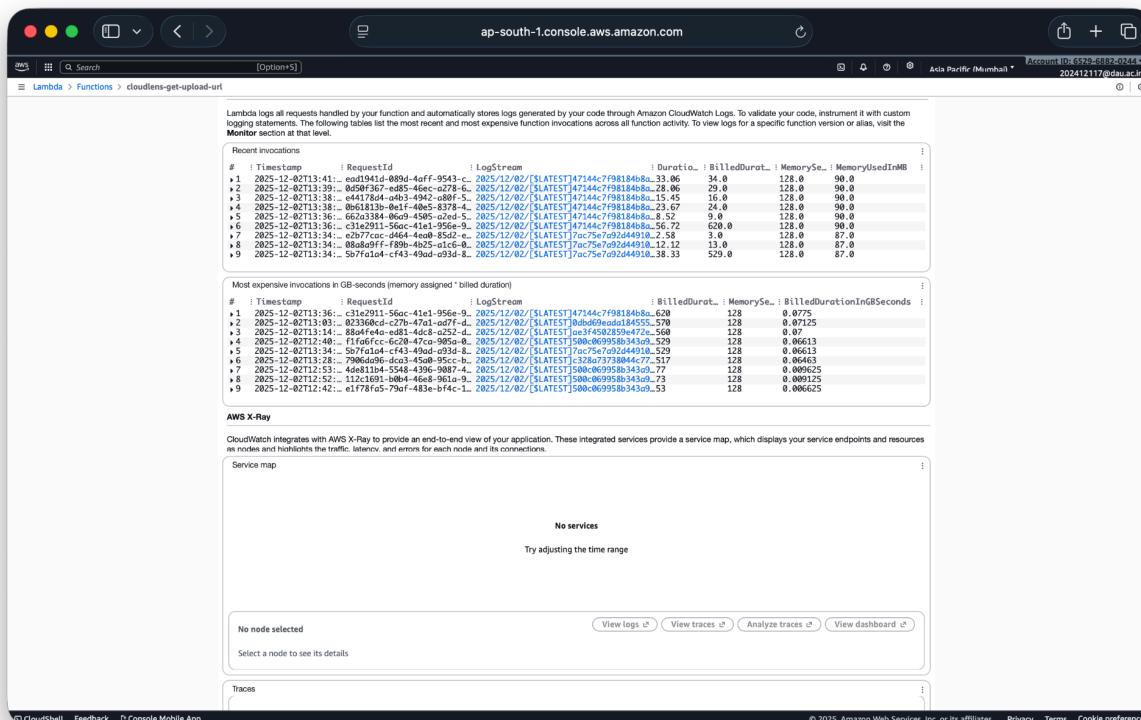
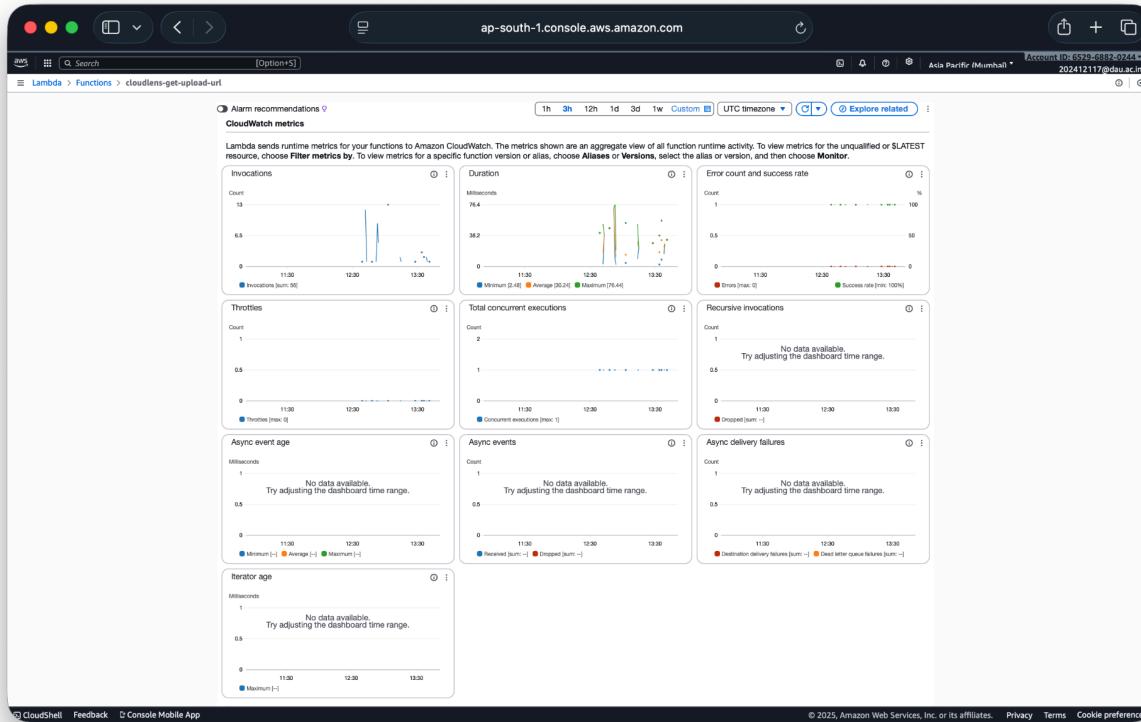
Traces

View log  View traces  Analyze traces  View dashboard

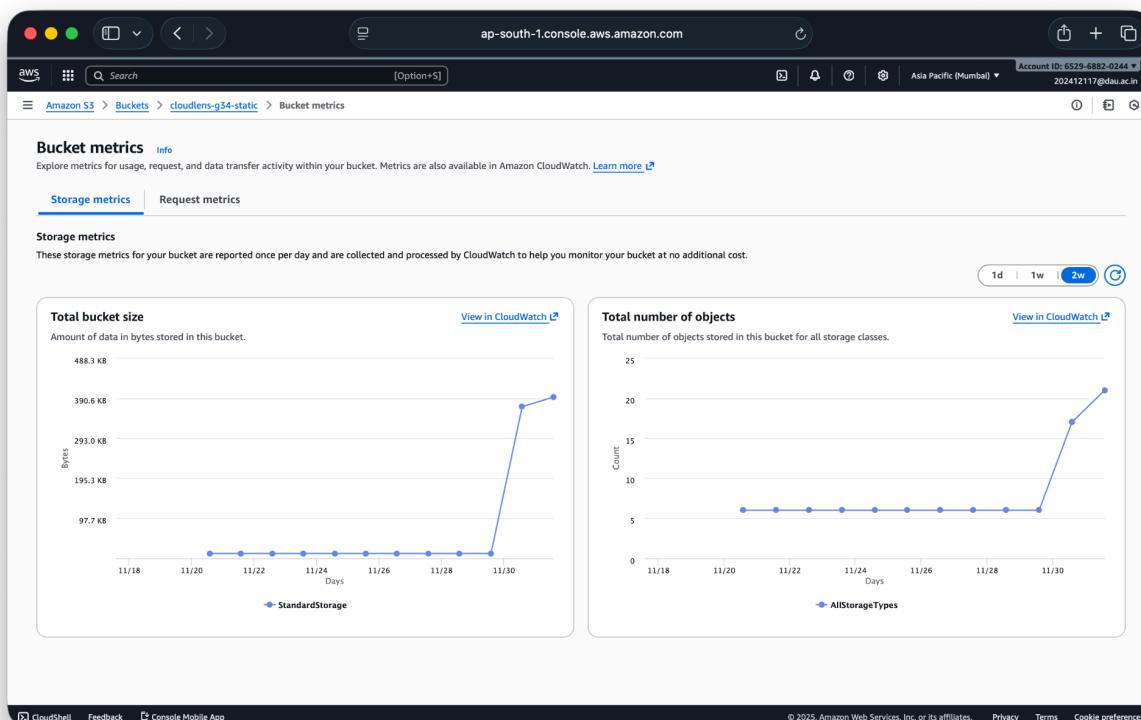
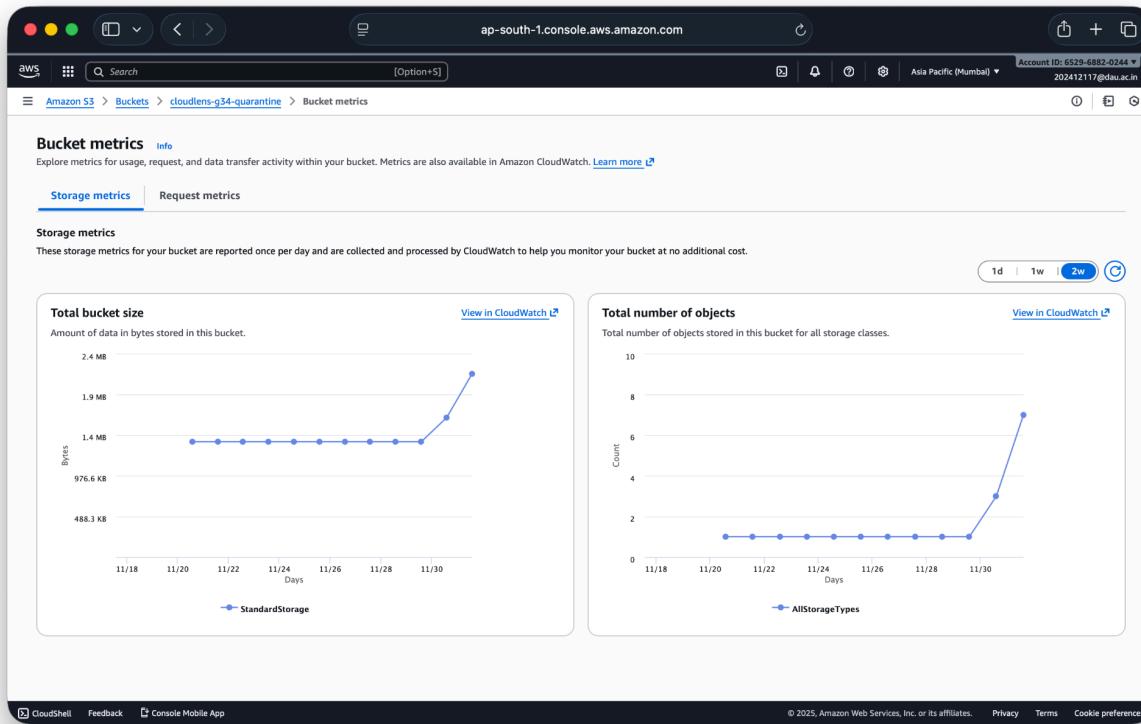
© CloudShell Feedback Console Mobile App

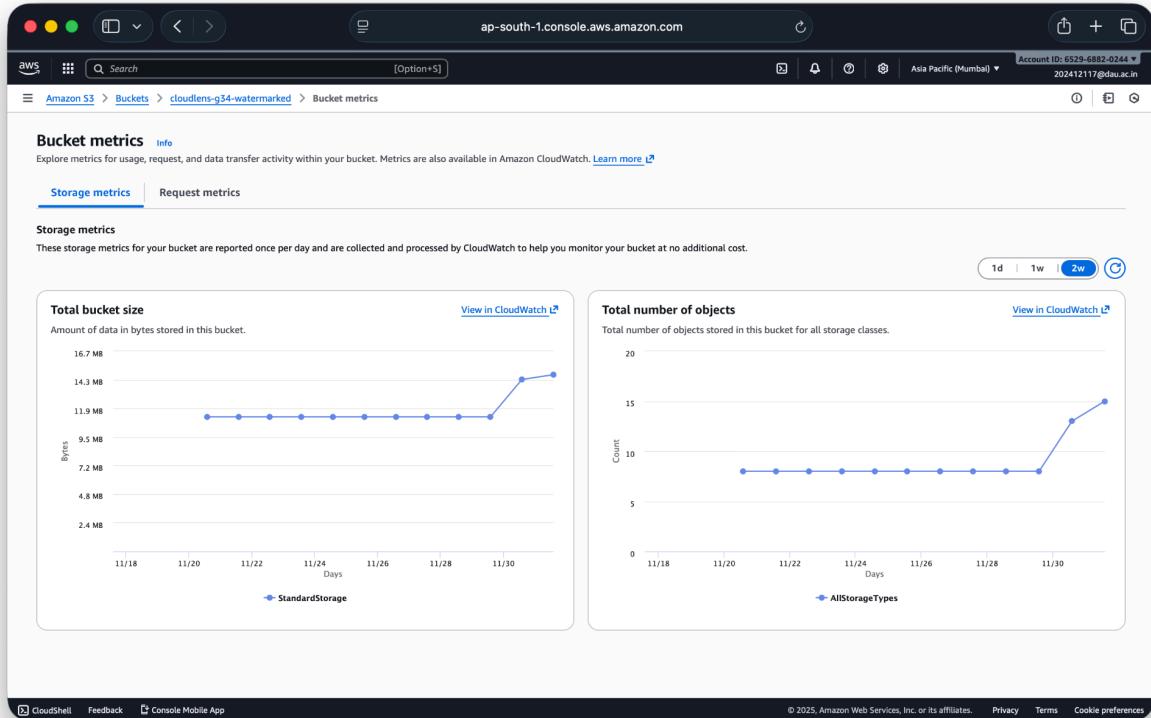
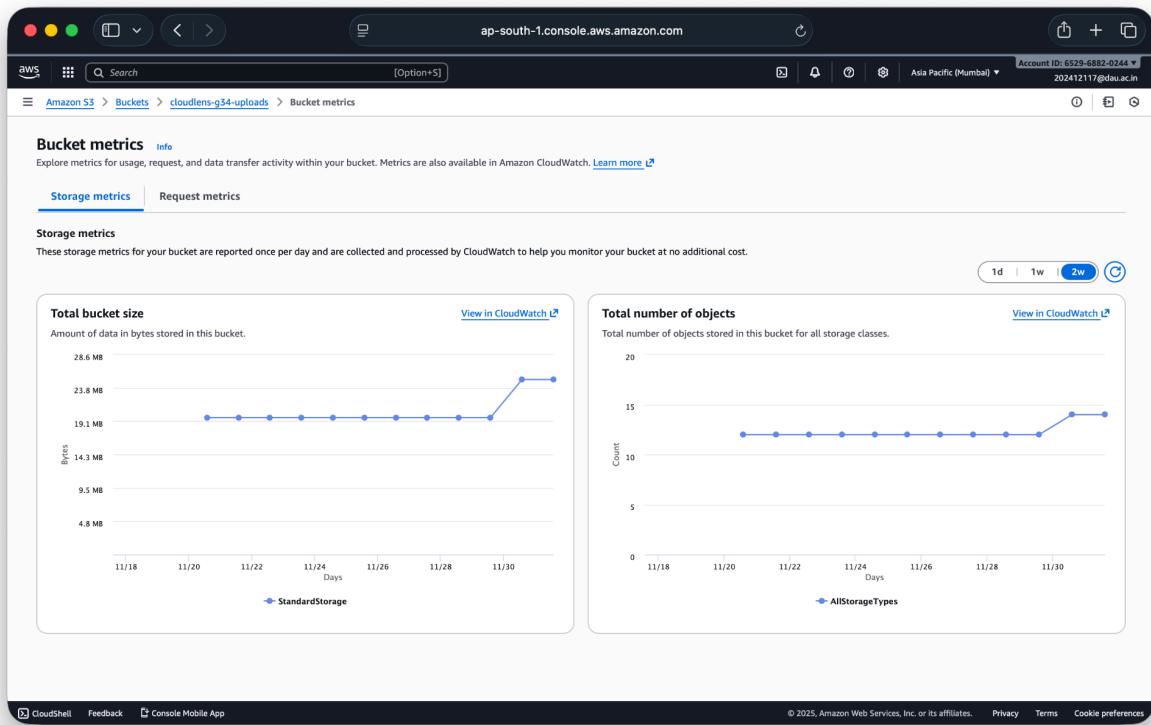
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

## 8.2 Lambda Function cloulens-get-upload-url



## 8.3 Buckets Metrics & Logs





The screenshot shows the AWS Lambda CloudWatch log viewer interface. The top navigation bar includes tabs for 'CloudShell', 'Feedback', and 'Console Mobile App'. The main content area displays log events for the 'cloudlens-processor' function. The logs are timestamped and show various API calls (INIT, START, REPORT, END) and their outcomes. A specific error is highlighted:

```

2025-12-02T13:36:27. INIT:START Runtime Version: python:3.13.-v74 Runtime Version ARN: arn:aws:lambda:ap-south-1:runtime:ce7da97c7c5e914ea022806e9f241cb83d93ebca5588c0918c38636ce
2025-12-02T13:36:28. START RequestId: c8aa0d9-e5fd-47e4-94fe-42351bd07a8 Version: $LATEST
2025-12-02T13:36:28. Error: An error occurred (AccessDenied) when calling the GetObject operation: User: arn:aws:sts::652968820244:assumed-role/ProcessingLambdaRole/cloudlens-processor is not authorized to perform: s3:ListBucket
2025-12-02T13:36:28. END RequestId: c8aa0d9-e5fd-47e4-94fe-42351bd07a8
2025-12-02T13:36:28. REPORT RequestId: c8aa0d9-e5fd-47e4-94fe-42351bd07a8 Duration: 81.52 ms Billed Duration: 685 ms Memory Size: 512 MB Max Memory Used: 101 MB Init Duration: 603.23 ms
2025-12-02T13:36:50. START RequestId: ecd0e3d8-a2e7-4dd1-9337-5d6de959801 Version: $LATEST
2025-12-02T13:36:50. Processing: thumbnail1/d226e058-69c1-4423-0bdd-9f5873f2d83_92990046cfec5c6e752ad3507869de05.jpg | Action: thumbnail | Meta: {'custom-size': '300'}
2025-12-02T13:36:50. ▲ UNSAFE CONTENT DETECTED: thumbnail1/d226e058-69c1-4423-0bdd-9f5873f2d83_92990046cfec5c6e752ad3507869de05.jpg
2025-12-02T13:36:50. END RequestId: ecd0e3d8-a2e7-4dd1-9337-5d6de959801
2025-12-02T13:36:50. REPORT RequestId: ecd0e3d8-a2e7-4dd1-9337-5d6de959801 Duration: 348.19 ms Billed Duration: 349 ms Memory Size: 512 MB Max Memory Used: 103 MB
2025-12-02T13:38:06. START RequestId: 732afdfb-f0b8-4972-8143-87080d4845ec Version: $LATEST
2025-12-02T13:38:06. Processing: thumbnail1/1477c2e9-8924-4d08-be7b-3c8996b42e3_step-03-buckets-created.png | Action: thumbnail | Meta: {'custom-size': '300'}
2025-12-02T13:38:07. ✓ Image Safe. Proceeding to task.
2025-12-02T13:38:07. Creating Thumbnail size 300px
2025-12-02T13:38:07. END RequestId: 732afdfb-f0b8-4972-8143-87080d4845ec
2025-12-02T13:38:07. REPORT RequestId: 732afdfb-f0b8-4972-8143-87080d4845ec Duration: 806.27 ms Billed Duration: 807 ms Memory Size: 512 MB Max Memory Used: 121 MB
2025-12-02T13:38:09. START RequestId: 25cf05ec-23df-4bc7-bcdd-e99ecbf528 Version: $LATEST
2025-12-02T13:38:09. Processing: thumbnail1/133d94b7-6c01-4e65-8407-1h12281d75d4_pool-party-91-070825-e72f47fe6754480d92b69fc29b104eo.jpg | Action: thumbnail | Meta: {'custom-size': '300'}
2025-12-02T13:38:09. ▲ UNSAFE CONTENT DETECTED: thumbnail1/133d94b7-6c01-4e65-8407-1h12281d75d4_pool-party-91-070825-e72f47fe6754480d92b69fc29b104eo.jpg
2025-12-02T13:38:09. END RequestId: 25cf05ec-23df-4bc7-bcdd-e99ecbf528
2025-12-02T13:38:09. REPORT RequestId: 25cf05ec-23df-4bc7-bcdd-e99ecbf528 Duration: 259.64 ms Billed Duration: 260 ms Memory Size: 512 MB Max Memory Used: 121 MB
2025-12-02T13:39:46. START RequestId: a97f808d-1530-4213-881b-6dc307e01659 Version: $LATEST
2025-12-02T13:39:46. Error: An error occurred (AccessDenied) when calling the GetObject operation: User: arn:aws:sts::652968820244:assumed-role/ProcessingLambdaRole/cloudlens-processor is not authorized to perform: s3:ListBucket
2025-12-02T13:39:46. END RequestId: a97f808d-1530-4213-881b-6dc307e01659
2025-12-02T13:39:46. REPORT RequestId: a97f808d-1530-4213-881b-6dc307e01659 Duration: 74.06 ms Billed Duration: 75 ms Memory Size: 512 MB Max Memory Used: 121 MB
2025-12-02T13:41:35. START RequestId: f59de2f3-2bd3-4077-a327-f3bc7514542 Version: $LATEST
2025-12-02T13:41:35. Error: An error occurred (AccessDenied) when calling the GetObject operation: User: arn:aws:sts::652968820244:assumed-role/ProcessingLambdaRole/cloudlens-processor is not authorized to perform: s3:ListBucket
2025-12-02T13:41:35. END RequestId: f59de2f3-2bd3-4077-a327-f3bc7514542 Duration: 67.03 ms Billed Duration: 68 ms Memory Size: 512 MB Max Memory Used: 121 MB
2025-12-02T13:41:35. REPORT RequestId: f59de2f3-2bd3-4077-a327-f3bc7514542 Duration: 67.03 ms Billed Duration: 68 ms Memory Size: 512 MB Max Memory Used: 121 MB

```

No newer events at this moment. Auto retry paused. [Resume](#)

## 9. MEMBER CONTRIBUTIONS

- **Member 1 (Nitesh): Backend: Moderation & Logic**
  - Primary Responsibilities:
    - Develop ModerationProcessor Lambda (Python)
    - Integrate Amazon Rekognition (DetectModerationLabels)
    - Implement safe/unsafe decision logic
    - Log all moderation results to CloudWatch Logs
- **Member 2 (Swara): Infrastructure, Security & Monitoring**
  - Primary Responsibilities:
    - Define all IAM Roles & Policies
    - Create all S3 Buckets
    - Build CloudWatch Dashboard for metrics
    - Compile final report & demo video
- **Member 3 (Kishan): Backend: Image Processing**
  - Primary Responsibilities:
    - Build and attach the Pillow Lambda Layer
    - Implement the thumbnail generation logic
    - Implement the watermark generation logic
    - Save processed images to appropriate S3 buckets
- **Member 4 (Stuti): Cloud-Integrated Frontend & API**
  - Primary Responsibilities:
    - Configure S3 Static Website Hosting
    - Set up API Gateway endpoint & GetUploadUrlLambda
    - Implement all CORS configuration (S3 & API-G)

## 10. CONCLUSION

The CloudLens project successfully demonstrates how serverless technologies on AWS can be combined to build a secure, scalable, and cost-efficient image moderation and processing pipeline. By leveraging services such as Amazon S3, AWS Lambda, Amazon Rekognition, and API Gateway, the system eliminates the need for traditional server management while providing real-time, automated content moderation and image processing. The integration of presigned URLs ensures secure uploads from untrusted clients, and the use of Pillow within a Lambda Layer enables efficient generation of thumbnails and watermarked images.

Throughout the implementation, core cloud principles including event-driven design, least-privilege IAM security, automatic scaling, and detailed monitoring via CloudWatch were applied effectively. The project remained within AWS Free Tier constraints, reinforcing its cost-optimized architecture.

Overall, CloudLens not only fulfills the problem requirement of moderating and transforming user-generated content but also showcases practical industry-level cloud design patterns. The experience enabled the team to gain hands-on understanding of serverless workflows, distributed cloud components, security enforcement, and operational monitoring. This project lays a solid foundation for future enhancements such as user authentication, advanced image processing, database integration, and automated CI/CD pipelines.



CLICK HERE TO GO TO [CLOUDLENS](#)