**DAYANAND ACADEMY OF MANAGEMENT STUDIES**

**A PROJECT REPORT**

**ON**

**(AWS SERVICES)**

**BCA**

**[SESSION-2024-27]**

Submitted by : Submitted to:

Gauri Shukla Mr. Abhinav Dwivedi

Divyanshi Pathak (Cloud Computing

Aanjaney Pandey Teacher)

Ayush Singh Sengar

**PROJECT**

**REPORT**

**TITLE: Cloud-Based Projects-**

**Image Processing, Website Hosting,**

**And System Monitoring using Grafana**

Table Of Contents

1. Introduction
   1. Overview
   2. Projects Covered
2. Problem Statement
   1. Objective
   2. Technologies Use
3. Implementation
   1. Project 1: Serverless Image Processing
      1. Aim
      2. Prerequisites
      3. Working Steps
      4. Result
   2. Project 2: Deploying a static website on AWS
      1. Aim
      2. Prerequisites
      3. Working Steps
      4. Result
   3. Project 3: Grafana Integration for Linux Server
      1. Aim
      2. Prerequisites
      3. Working Steps
      4. Result
   4. Conclusion

INTRODUCTION

1.1 **Overview**

This project contains three useful tasks based on cloud. Each task uses basic cloud tools to solve a real-world problem.

1.2 **Projects Covered**

a. **Project 1:** Serverless Image Processing

b. **Project 2:** Static Website Hosting on AWS

c. **Project 3:** Grafana Integration for monitoring Linux server

Problem Statement

**2.1 Objective**

To build beginner-friendly cloud projects:

1. Automatically resize images uploaded to S3 bucket.
2. Host a simple static website on AWS
3. Monitoring CPU usage on a Linux server using Grafana

**2.2 Technologies Used**

a. AWS Lambda, S3

b. Node.js, Sharp

c. HTML

d. Prometheus, Grafana

e. EC2 (Ubuntu Linux)

Implementation

* 1. **Project 1: Serverless Image Processing**
     1. **Aim:** Create a serverless image processing application that automatically resizes and optimizes images uploaded to an S3 Bucket.
     2. **Prerequisites:**

**a.** An AWS Account.

1. Thorough Knowledge about AWS Services like S3(for storage), LAMBDA(for serverless functions), IAM(for setting permissions and role)
2. Basic programming knowledge, particularly with Node.js, since the lambda function uses Node.js to process the image.
3. A ZIP file containing the image processing code(already prepared and ready to upload to lambda).
   * 1. **Working Steps:**
4. **Sign in to AWS Console**

Open the AWS console using your credentials.

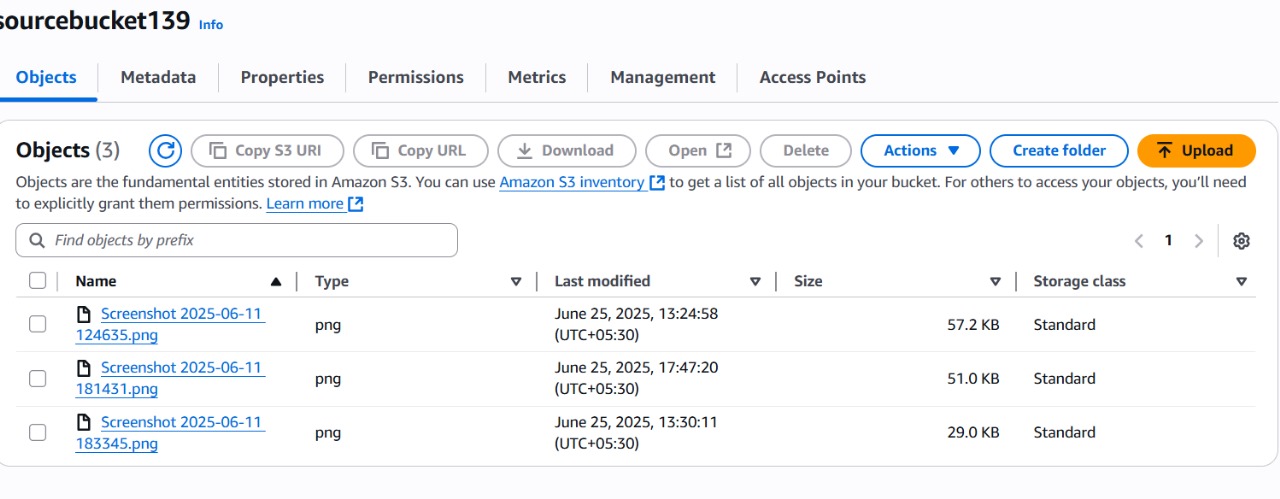
1. **Create two S3 Buckets** :
2. Go to S3 Service.
3. Create a source bucket to receive original images
4. Create a destination bucket where resized images will be stored.
5. Make sure each bucket name is globally unique.
6. **Create IAM Policy and Role:**
7. Go to IAM service and create a new policy.
8. In the policy, allow permissions for reading from the source bucket, writing to the destination bucket and creating logs.
9. After creating the policy, attach it to a new IAM role.
10. When creating the role, chose Lambda as the use case.
11. **Create a Lambda Function**:
12. Go to Lambda service and click on Create Function.
13. Choose “Author from scratch”, set a name, select Node.js 18.x, as runtime, and x86\_64 as architecture.
14. Under permissions, choose the existing IAM role created earlier.
15. After creating the function, upload the ZIP file containing your image processing code.

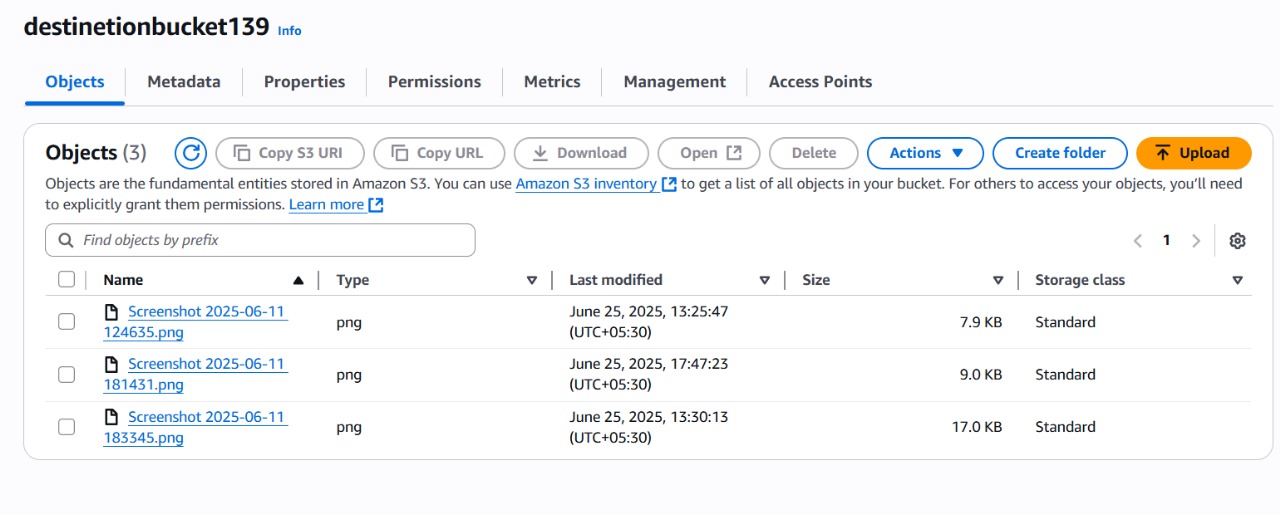


1. **Add a trigger to Lambda**:
2. In the lambda function page, go to Add Trigger.
3. Choose S3 as the trigger source and select the source bucket.
4. This will trigger the lambda function automatically whenever an image is uploaded to the source bucket.
5. **Test the setup the Application:**
6. Upload an image file to the source S3 bucket.
7. Wait a few seconds, and then check the destination bucket.
8. If the setup is correct, you will find a resized version of the image in the destination bucket.

**3.1.4 Result:**

**The serverless image processing application has been created.**

****

****

**3.2 Project 2: Deploying a static website on AWS**

**3.2.1 Aim**: To deploy a static website using AWS

**3.2.2 Prerequisites:**

1. An AWS Account.
2. Thorough Knowledge about AWS S3 Service.
3. A HTML/CSS/JAVASCRIPT File to upload in the S3 bucket.
   * 1. **Working Steps:**
4. **Sign in** to AWS Console
5. **Create an S3 bucket** with a unique name and disable “Block Public Access”.
6. **Enable Static Website Hosting** in the bucket properties
7. Set hosting type: “Host a static website”
8. Set index.html as Index Document.
9. **Upload HTML** **file** to the bucket.

<https://d.docs.live.net/6e18f0e27fc42b40/Documents/quiz.html>

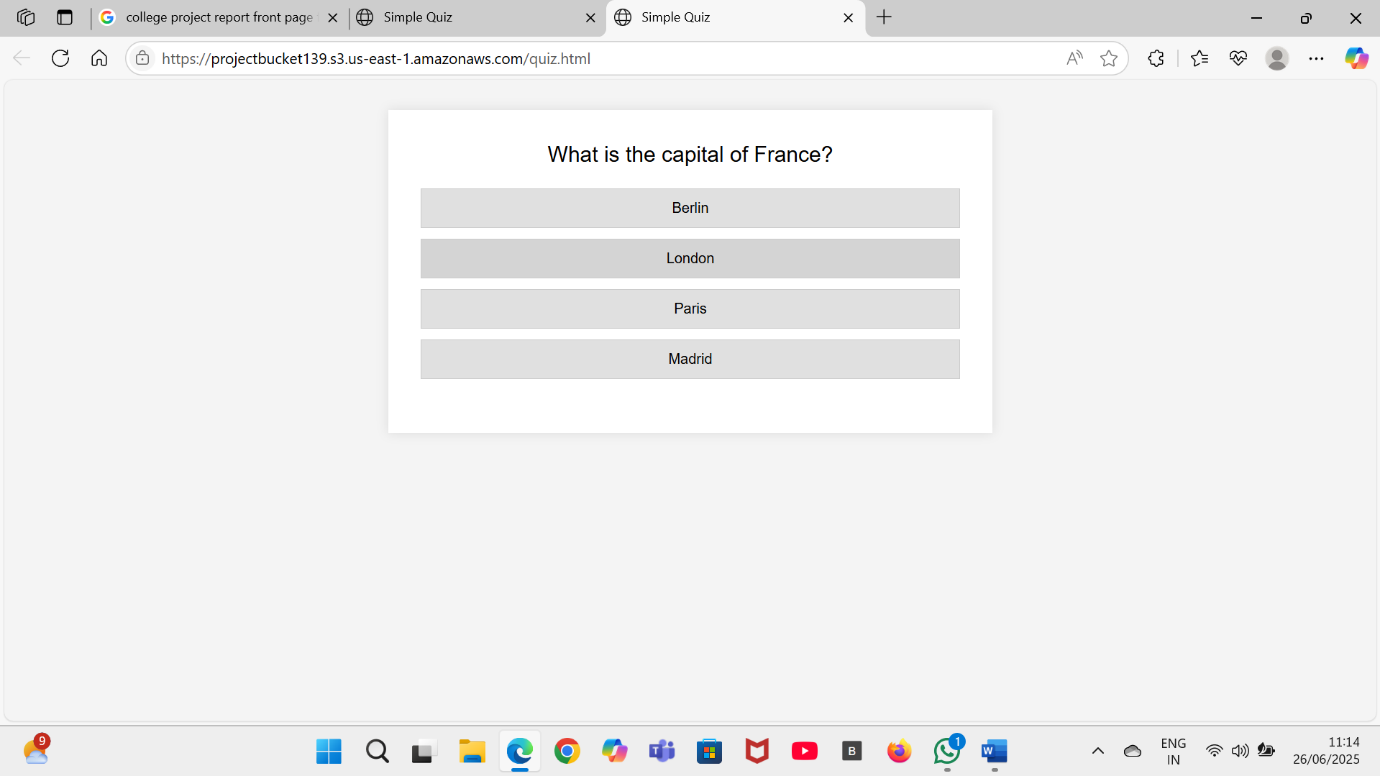
1. **Set Permissions** :

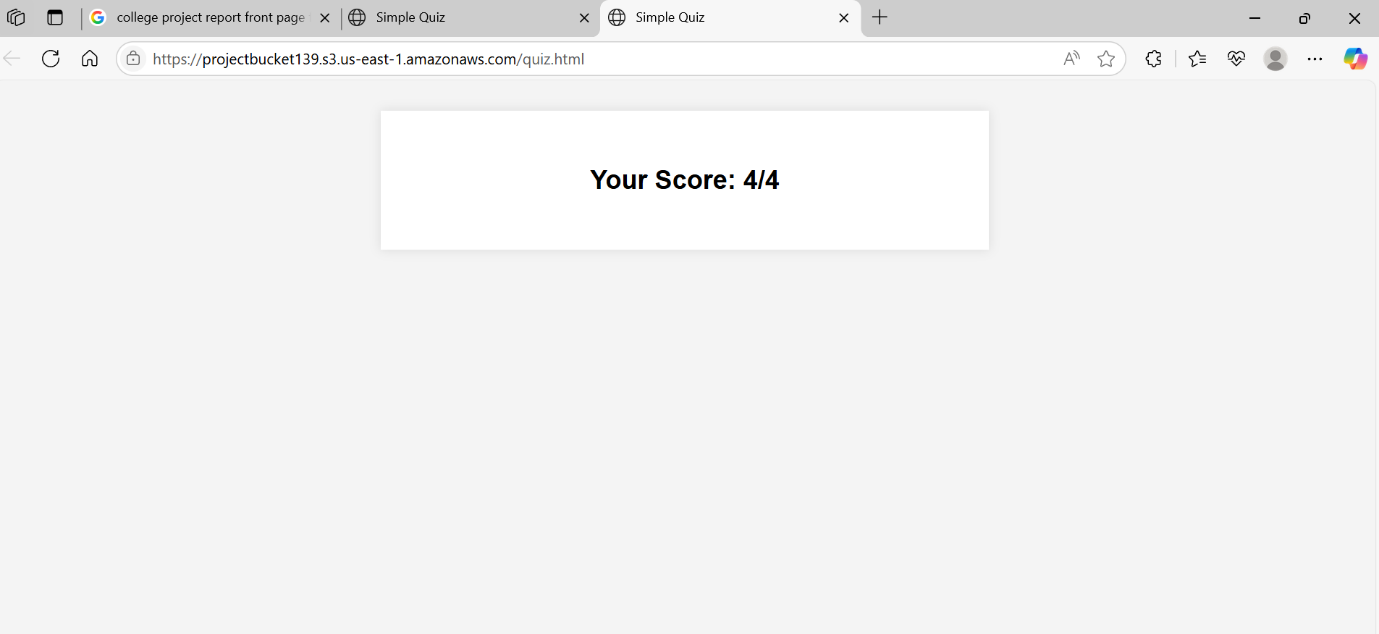
Add a **Bucket Policy** allowing public access (GetObject)

1. **Test the website** using the provided **S3 endpoint URL**
   * 1. **Result:**

The static website was **successfully hosted** and can be viewed publicly using the S3 URL:

[**https://projectbucket139.s3.us-east-1.amazonaws.com/quiz.html**](https://projectbucket139.s3.us-east-1.amazonaws.com/quiz.html)

****

****

**3.3 Project 3: Grafana Integration for Monitoring Linux Server**

* + 1. **Aim:** Integrate Grafana with Linux server for high CPU utilization and create a graph in Grafana.
    2. **Prerequisites:**

1. An **AWS EC2 instance** (preferably Ubuntu)
2. **Ports open** in the EC2 security group:
3. **Port 22(**SSH**)** for connecting with PuTTY
4. **Port 3000** for accessing Grafana
5. **Port 9090** for Prometheus
6. **Port 9100** for Node Exporter
7. **Basic** **knowledge** of using the terminal and **PuTTY** for connection. And terminal commands.
   * 1. **Working Steps:**
8. **Launching EC2 Instance:**
9. **Login to AWS Console** and launch a new EC2 instance.
10. Create a new EC2 instance with Ubuntu as the os.
11. Generate and download a key pair in .ppk format
12. Set inbound rules to allow necessary ports for Grafana, Prometheus, and Node Exporter
13. **Connect to EC2 Instance:**
14. Use PuTTY to connect to your instance using the publicIPv4 address and the .ppk key.
15. Login as Ubuntu user
16. **Install Node Exporter** for collecting server metrices. Run the given commands:

**cd /opt**

**wget https://github.com/prometheus/node\_exporter/releases/download/v1.8.1/node\_exporter-1.8.1.linux-amd64.tar.gz**

**tar -xvzf node\_exporter-1.8.1.linux-amd64.tar.gz**

**mv node\_exporter-1.8.1.linux-amd64 node\_exporter**

**cd node\_exporter**

**./node\_exporter &**

1. **Install Prometheus** and configure it to monitor both Prometheus and Node Exporter. Run the following commands:

**cd /opt**

**wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz**

**tar -xvzf prometheus-2.52.0.linux-amd64.tar.gz**

**mv prometheus-2.52.0.linux-amd64 prometheus**

**cd prometheus**

**Edit Config:**

**nano prometheus.yml**

**Edit prometheus.yml:**

**scrape\_configs:**

**- job\_name: 'prometheus'**

**static\_configs:**

**- targets: ['localhost:9090']**

**- job\_name: 'node\_exporter'**

**static\_configs:**

**- targets: ['localhost:9100']**

**Run Prometheus:**

**./prometheus &**

1. **Install Grafana**, start its services, and open it in the browser. Follow the given commands:

**sudo apt-get install -y software-properties-common**

**sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"**

**wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -**

**sudo apt-get update**

**sudo apt-get install -y grafana**

**sudo systemctl start grafana-server**

**sudo systemctl enable grafana-server**

**Access Grafana:** http://<your-ec2-ip>:3000

1. **Login to Grafana**, with admin/admin and change the password.
2. **Add Prometheus as Data Source:**
3. In Grafana, go to **Connections>Data Sources**
4. Select Prometheus and enter the URL <http://localhost:9090>
5. Save and test the connection
6. **Create CPU Usage Dashboard:**
7. Go to Dashboards>New Dashboard
8. Add a new panel with the query

node\_cpu\_seconds\_total

1. Run the query to display the CPU usage graph
2. Rename the panel and save the dashboard
3. **Result:**

**A real-time CPU usage graph of the Linux server is successfully created and displayed in Grafana**

****

**Conclusion**

**This project shows how simple cloud tools can be used for useful purposes. We can create an image processing system, a hosted website, and a live CPU monitoring setup using free and easy-to-learn services. Each project is a stepping stone toward learning real-word cloud applications.**