Table of Contents

[Table of Contents 1](#_Toc1041051991)

[1. Network Diagram of Infrastructure 2](#_Toc1579300212)

[2. Detailed Description of Solution 3](#_Toc2019597097)

[Web Application in VPC on AWS Cloud 4](#_Toc1073058818)

[Serverless Application 4](#_Toc347319866)

[3. Problems Encountered and Resolutions 5](#_Toc734310439)

[4. How the requirements are implemented and met 6](#_Toc494319838)

[Requirement and Solution Analysis 1 7](#_Toc692283844)

[Requirement and Solution Analysis 2 8](#_Toc1321605144)

[5. Code for Lambda Functions 9](#_Toc1917207491)

[Thumbnails and Images 10](#_Toc1428707563)

[DynamoDB and Form 11](#_Toc1985533677)

[Get Customer Data 11](#_Toc248916005)

[Add Customer Data 12](#_Toc339858643)

[Notification System 12](#_Toc622650923)

[Thumbnail Notification 12](#_Toc2108051166)

[DynamoDB Record Notification 12](#_Toc1004827139)

[6. Testing Methods and Results of Working Solution 13](#_Toc513696350)

[Mandatory 14](#_Toc1954458394)

[Auto Scaling & Web server 14](#_Toc1634593594)

[CloudWatch Alarm & SNS notification 17](#_Toc249991460)

[Additional 22](#_Toc605638537)

[Thumbnail Creation 22](#_Toc1455748366)

[7. Incorporation of AWS Well-Architected Framework 27](#_Toc238924014)

[Operational Excellence 28](#_Toc1830414500)

[Security 28](#_Toc1071169738)

[Reliability 28](#_Toc897032654)

[Performance Efficiency 29](#_Toc1380447360)

[Cost Optimisation 29](#_Toc1889187526)

[Sustainability 30](#_Toc304767226)

[8. Explanation on Cloud Monitoring Plan 30](#_Toc984203424)

[CloudWatch Dashboard 31](#_Toc758620758)

[CloudWatch Alarms 32](#_Toc1852308073)

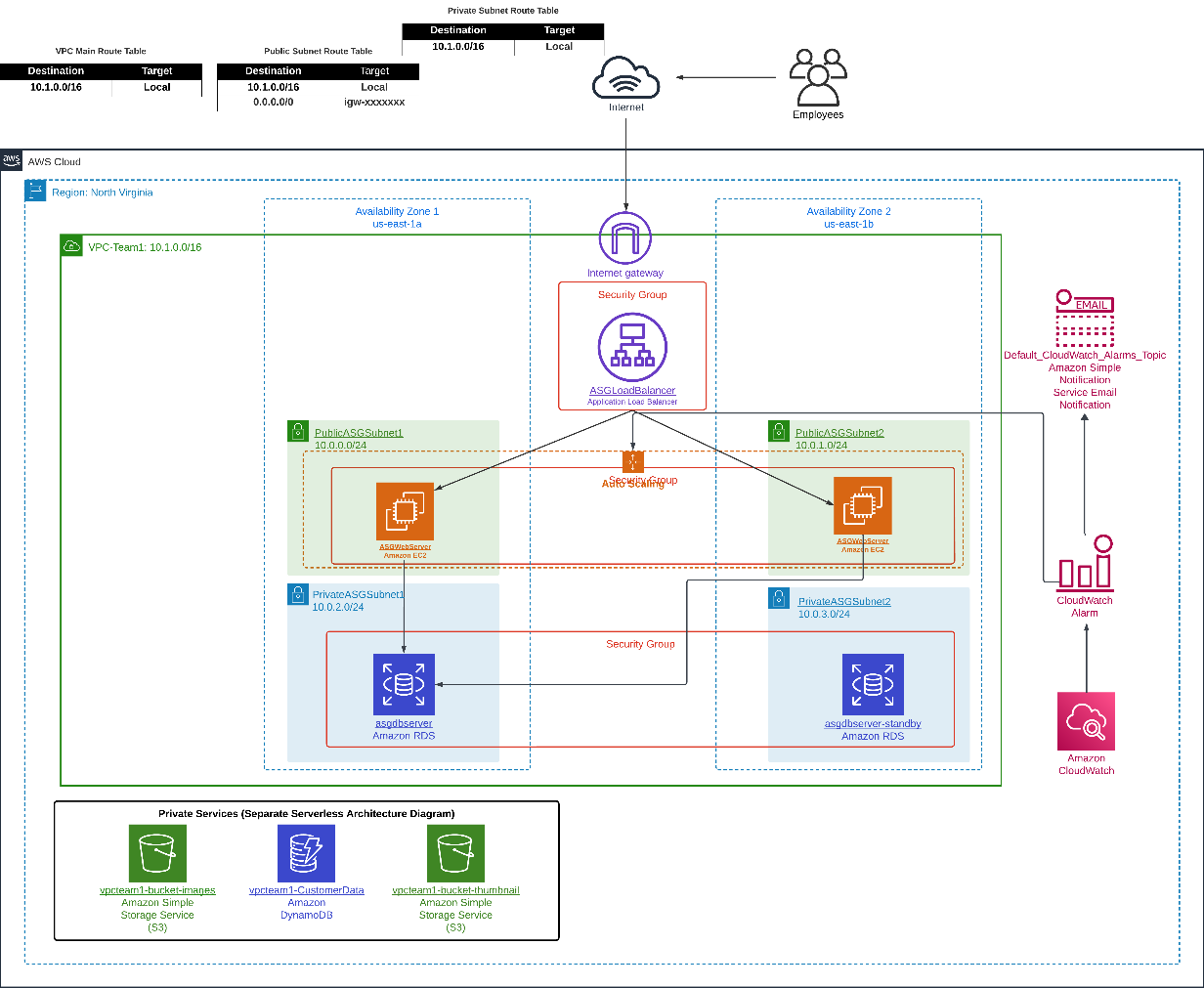
[9. Relevant Screenshots and Diagrams 32](#_Toc768517546)

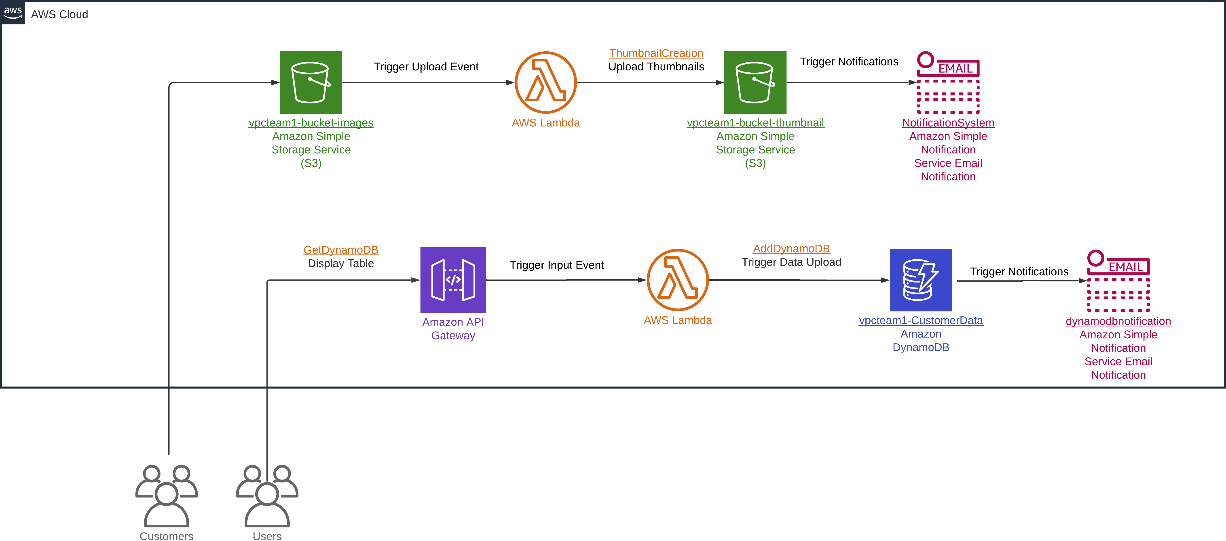
[10. References 37](#_Toc1088015152)

**Google Drive Link:** <https://drive.google.com/drive/folders/1hyffwp1jSyZLe0aoZZaSAqRrbRH-_YWq?usp=sharing>

# 1. Network Diagram of Infrastructure

(HongKai)





# 2. Detailed Description of Solution

(HongKai)

## Web Application in VPC on AWS Cloud

**Region and Availability Zones**

The architecture is deployed in the North Virginia region (us-east-1) and utilises two availability zones (us-east-1a) and (us-east-1b).

**VPC (Virtual Private Cloud) and Subnets**

A VPC with the CIDR Address 10.1.0.0/16 is created and is divided into four subnets. Public Subnet 1 and Private Subnet 1 in Availability Zone 1, with Public Subnet 2 and Private Subnet 2 in Availability Zone 2.

**Public Subnets**

At the minimum, each public subnet contains an EC2 instance to host the web application. The EC2 instances are part of an Auto Scaling Group and behind an Elastic Load Balancer.

**Private Subnets**

Each private subnet contains an RDS instance, the main RDS instance in Private Subnet 1 and the Standby RDS instance in Private Subnet 2.

**Security**

Along with the Private Subnets, security groups were implemented to control inbound and outbound traffic to the resources, which included Elastic Load Balancer, EC2 Instances and the RDS instances.

**Networking**

An Internet Gateway is attached to the VPC to allow internet resources and as mentioned above, an Elastic Load Balancer was used to distribute incoming application traffic across the EC2 Instances.

**Monitoring and Notifications**

Amazon CloudWatch is used for monitoring the resources through their customised dashboard and for setting up alarms. The Amazon Simple Notification Service is used to send email notifications based on the CloudWatch Alarms.

## Serverless Application

**Thumbnail Creation**

When customers upload files, they are stored in an S3 bucket (vpcteam1-bucket-images). This triggers an AWS Lambda function to create thumbnails of the uploaded files, where these thumbnails will be stored in another S3 bucket (vpcteam1-bucket-thumbnail).

**Customer Data Handling**

When customer data is submitted through the HTML form, the integration with Amazon API Gateway which handles HTTP requests, triggers the corresponding Lambda function to store the data in DynamoDB Table (vpcteam1-CustomerData).

**Notifications**

The S3 event of the thumbnail uploads or the addition of customer data to the NoSQL database, triggers AWS Simple Notification Service to send email notifications to the intended recipients.

# 3. Problems Encountered and Resolutions

**Problem 1: Disagreement on EC2 Instance Placement**

Initially when drawing up the AWS architecture diagram before the implementation, our team faced a disagreement on whether the EC2 instances should be placed in private or public subnets. This was driven by our passion for cloud computing and desire to achieve the best results for our assignment, with the debate centring around balancing security and accessibility.

**Solution 1**

Since we started the assignment relatively early, it was no cause for concern. With a lot of time on our hands, we each re-examined the assignment’s context and requirements, conducted individual research on best practices before discussing our findings. Ultimately, after considering factors such as security and accessibility we reached a consensus to place the EC2 instances in public subnets, with the use of security groups to control traffic.

**Problem 2: Only 1 Lambda Function can be Connected to an S3 Bucket**

After finishing the set-up of the lambda function and successfully getting it working, I planned on creating another Lambda Function to call on the Notification System set up but noticed that the system did not allow it. After some debugging, I found out that it was because an S3 Bucket only allowed the linking of 1 S3 trigger which is implemented inside the Lambda Function.

**Solution 2**

To solve this issue, I searched online for other methods but came up empty handed. The solutions provided were over complicated, and thus I opted to just include the calling of the Notification System inside my already created thumbnailCreation lambda function.

**Problem 3: Coordination problems**

When implementing our plans into the AWS Cloud, we were not able to effectively collaborate with each other as the software we used, Learner Lab did not have any collaboration options.

**Solution 3**

By utilising numerous methods such as updating each other actively on our changes and calling each other on platforms such as Teams, we were able to collaborate effectively even in the absence of such features on the platform.

# 4. How the requirements are implemented and met

## Requirement and Solution Analysis 1

**Requirement Analysis:**

**Host existing web application in VPC (Virtual Private Cloud) on AWS Cloud**

* Highly available resilient, auto-scaling and secure VPC
* Web application servers built on general purpose servers (Min:2, Max:6, Desired:2)
* Secure, resilient, scalable, load-balanced, and accessible from the Internet.
* Design based on AWS well-architected design principles: security, reliability, performance efficiency and cost optimisation.
* Customised CloudWatch Dashboard, along with alarms and notification systems
* Linux or Windows operating system and RDS can be used to store its data

**Solution Analysis:**

To meet the requirements for hosting the web application in a VPC, we configured a VPC with the CIDR Address 10.1.0.0/16 as Team 1. Ensuring high availability, we distributed our public and private subnets across two availability zones (AZs). This design approach mitigates the risk of single points of failure.

**Web Application Servers**

We chose EC2 instances of the type t2.micro for the web application servers. This decision was driven by the need to optimise costs while still meeting the requirement for general-purpose servers. To handle varying loads, we implemented auto-scaling for these EC2 instances. The auto-scaling group is configured to maintain a minimum of 2 instances, scale up to 4 instances during high demand, and scale back down to 2 instances during periods of low demand.

**Accessibility and Load Balancing**

Both the EC2 instances spread across two public subnets and the internet gateway attached to the VPC allows the web application to accessible from the Internet. To enhance scalability and distribute network traffic evenly across the EC2 instances, we deployed an Elastic Load Balancer (ELB). The combination of an auto-scaling group and the ELB ensures the web application remains responsive and performs well under varying load conditions.

**Database Configuration**

For data storage, we utilised Amazon RDS with a primary and a standby instance, placed in separate private subnets across two AZs. This setup ensures high availability and data redundancy, adhering to the principle that everything should be designed to handle failure.

**Security Measures**

We also acknowledged that security is a paramount concern. We established security groups for the Elastic Load Balancer, EC2 instances, and RDS instances, defining strict inbound and outbound rules to control access.

**Monitoring and Notifications**

To enable effective monitoring and management of AWS resources, we utilised AWS CloudWatch. We created a customized CloudWatch dashboard to track critical metrics. CloudWatch alarms were set up to trigger actions, such as scaling the web application automatically when CPU usage exceeds a threshold. Additionally, Amazon Simple Notification Service (SNS) was integrated to send email notifications to the intended recipients when alarms are triggered, allowing them to stay informed.

## Requirement and Solution Analysis 2

**Requirement Analysis:**

**Serverless Application Implementation**

* Create a NoSQL database to store customer data
* Create a HTML form page to capture and post customer data to the NoSQL database
* Create a notification system (via SMS or email)
* Create a serverless function to create thumbnails for uploaded pictures

**Solution Analysis:**

For the serverless application, we utilised several AWS services to meet the specified requirements efficiently.

**File Storage**

We created three Amazon S3 buckets: ‘vpcteam1-bucket-images’ for storing file uploads, ‘vpcteam1-bucket-thumbnail’ for storing created thumbnails, and ‘vpcteam1-websiteform-bucket’ for the form to add customer data. S3 offers 99.999999999% (11 nines) data durability and 99.99% availability, making it a highly reliable choice for object storage.

**NoSQL Database**

We used Amazon DynamoDB to implement the NoSQL database. The DynamoDB table named CustomerDataTable1 includes attributes for the customer's first name, last name, and email address. We chose the email address as the partition key, operating under the assumption that each customer is associated with a unique email address. A sort key was not necessary, as there are no multiple related items per partition key in this use case.

**HTML Form**

An HTML form page was created to capture customer data. The form is integrated with Amazon API Gateway in the form of REST API, which handles HTTP requests and triggers the corresponding Lambda function.

**Serverless Functions**

Multiple AWS Lambda functions were created to handle different tasks:

1. Thumbnail Creation: This Lambda function triggers when a file is uploaded to the `vpcteam1-bucket-images` S3 bucket. It processes the image and stores the generated thumbnail in the ` vpcteam1-bucket-thumbnail` S3 bucket. It also triggers the Notification system.

2. Customer Data Handling: This Lambda function is invoked when customer data is submitted through the HTML form. It processes the data and stores it in DynamoDB.

3. Customer Data Display: This Lambda function is triggered to show all records within the customer database and displays them in a certain format. It links and get the information required from the DynamoDB.

**Notification System**

To keep the intended recipients promptly alerted to events, we set up Amazon Simple Notification Service (SNS) for email notifications. SNS sends notifications to the stated email account when new data is added to DynamoDB or when a file is uploaded to S3.

**Security Measures**

Access Control Lists (ACLs) were implemented to allow users to access certain web pages in the S3 buckets.

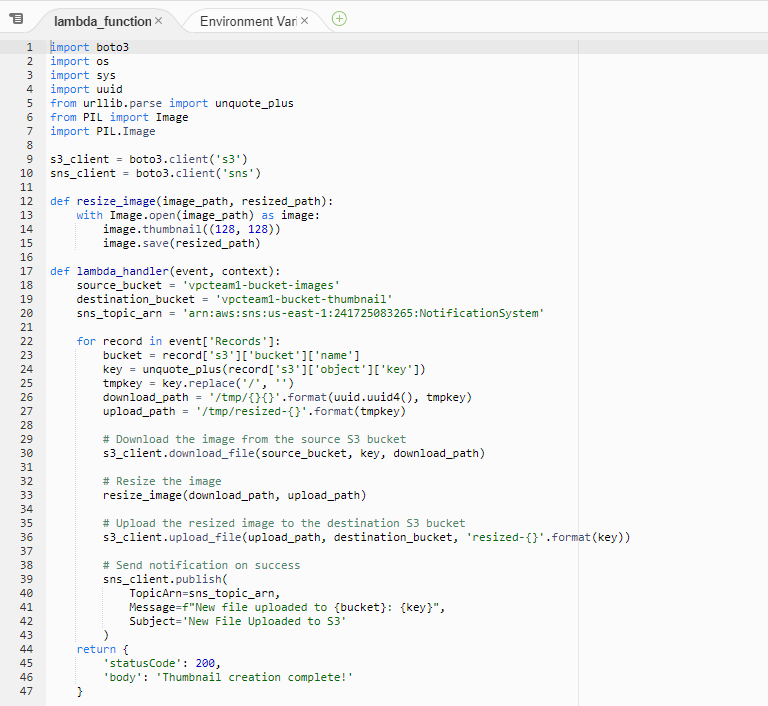
# 5. Code for Lambda Functions

(Louis and ChangZe)

ChangZe: Thumbnails

Louis: DynamoDB

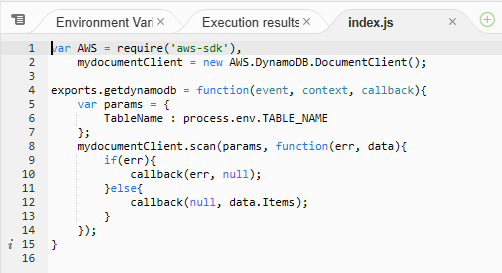
## Thumbnails and Images



**\*Code file can be found in the Google Drive Link in the Appendix\***

DynamoDB and Form

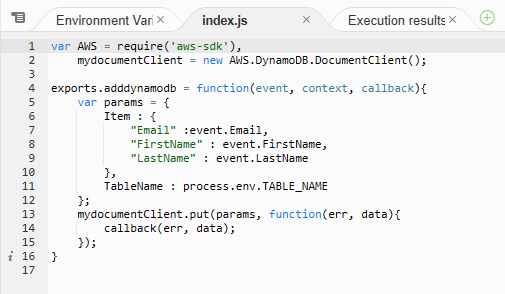
### Get Customer Data



**\*Code file can be found in the Google Drive Link in the Appendix\***

The code provided is used to gather the data of the items within the DynamoDB which is named ‘vpcteam1-CustomerData’.

Add Customer Data



**\*Code file can be found in the Google Drive Link in the Appendix\***

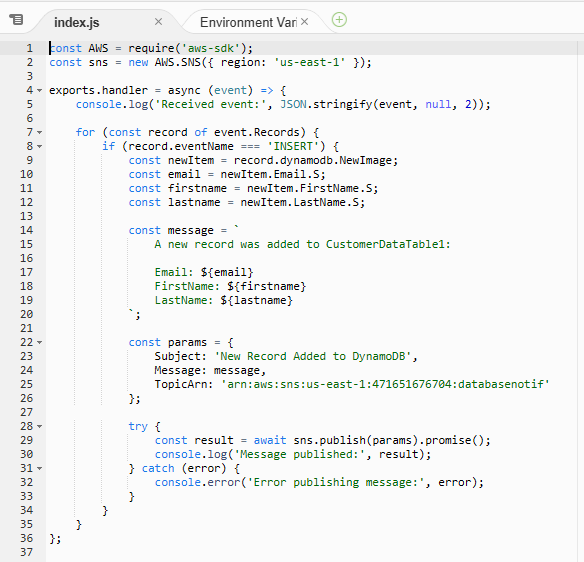
The code provided is used to append records into the DynamoDB, ‘vpcteam1-CustomerData’.

## Notification System

Thumbnail Notification

**\*Notification system is placed together with the resizing code\***

### DynamoDB Record Notification



**\* Please note that for this image code, some variables defer from the actual code\***

**\*Actual Code file can be found in the Google Drive Link in the Appendix\***

The code provided is used to send an email to the subscribed email account specified during the creation of the simple notification service.

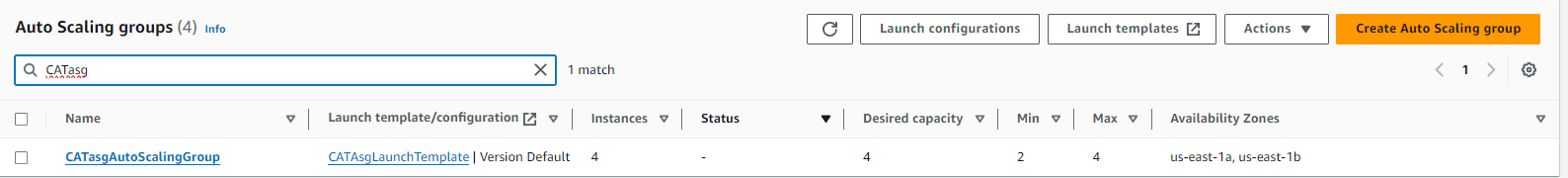
# 6. Testing Methods and Results of Working Solution

## Mandatory

(Elvis)

### Auto Scaling & Web server

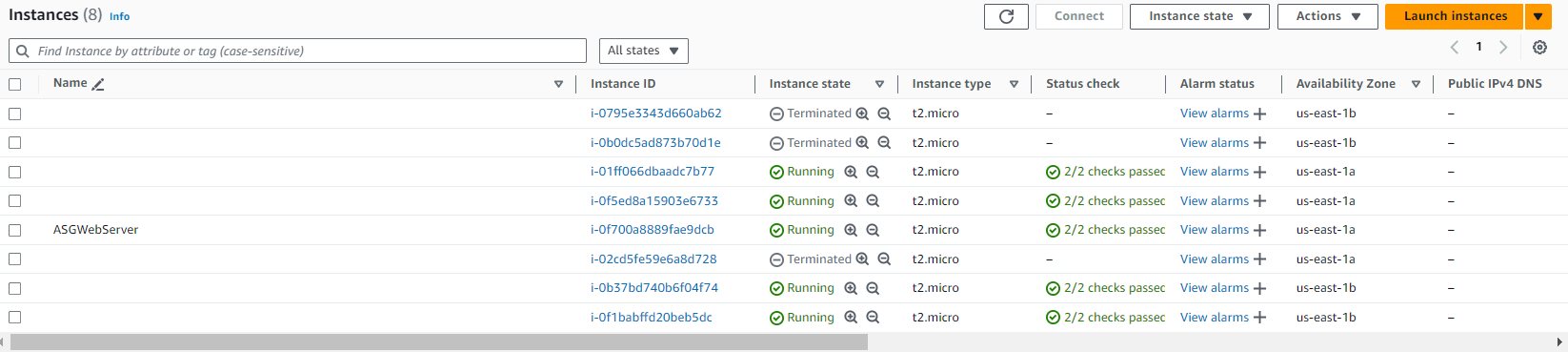
\*\*Note that the implementation video saved in google drive\*\*



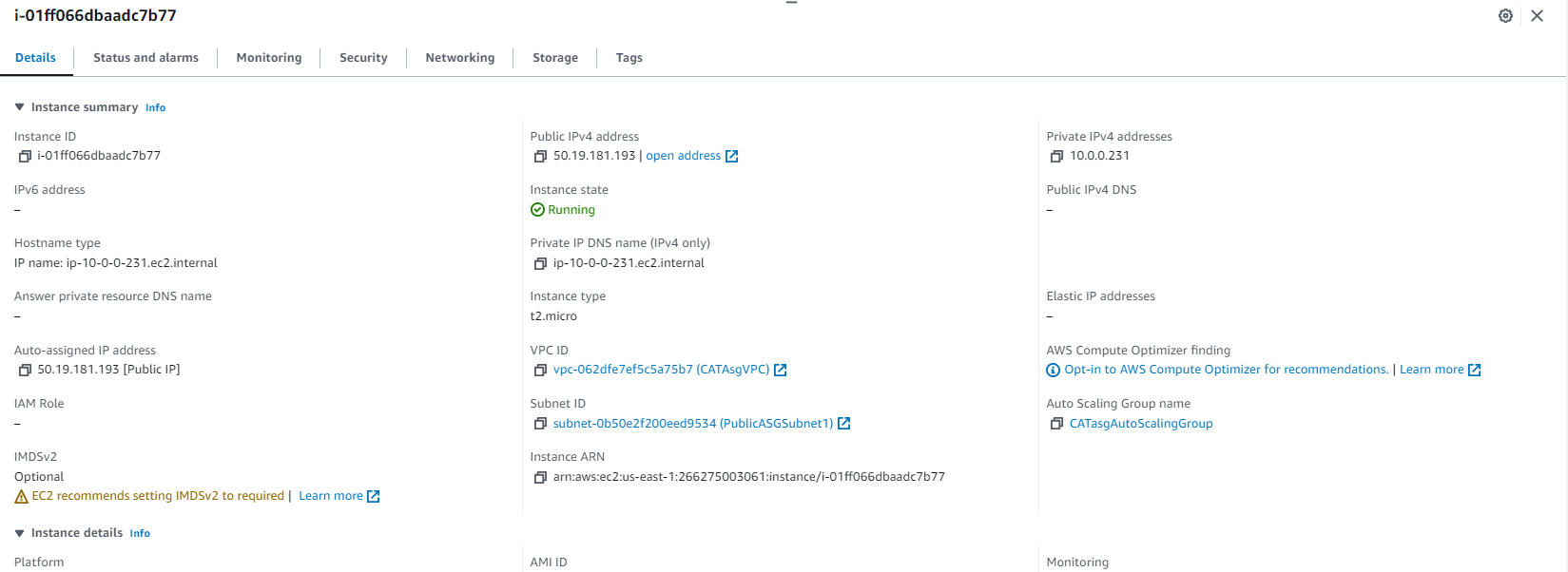
The auto scaling group used is CATasgAutoScalingGroup.

The testing steps are as follows:

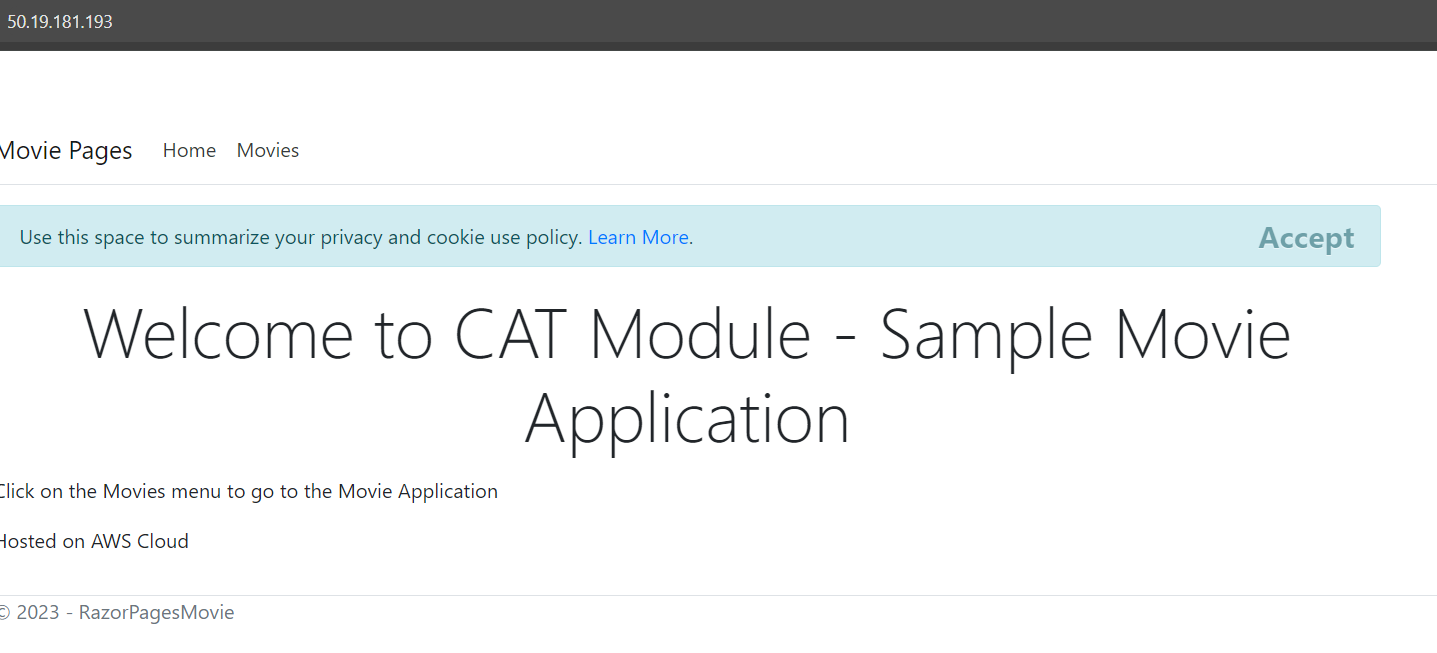
1. Launch Auto-Scaling group and wait for EC2 instances to be deployed

2. Once deployed, click on any EC2 instance to obtain the Public IPv4 Address.

In this context, Public IPv4 Address is 50.19.181.193



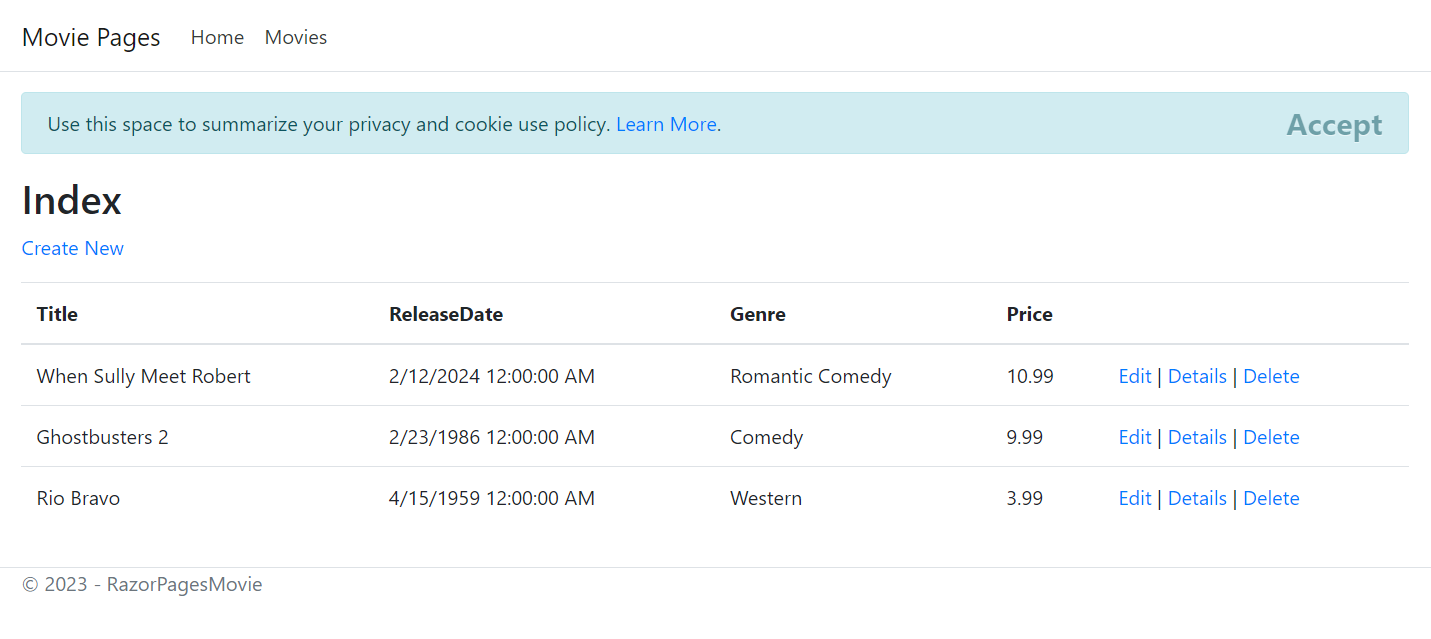
3. Paste the address into a web browser and ensure that Web Application is loaded with all functionalities working.



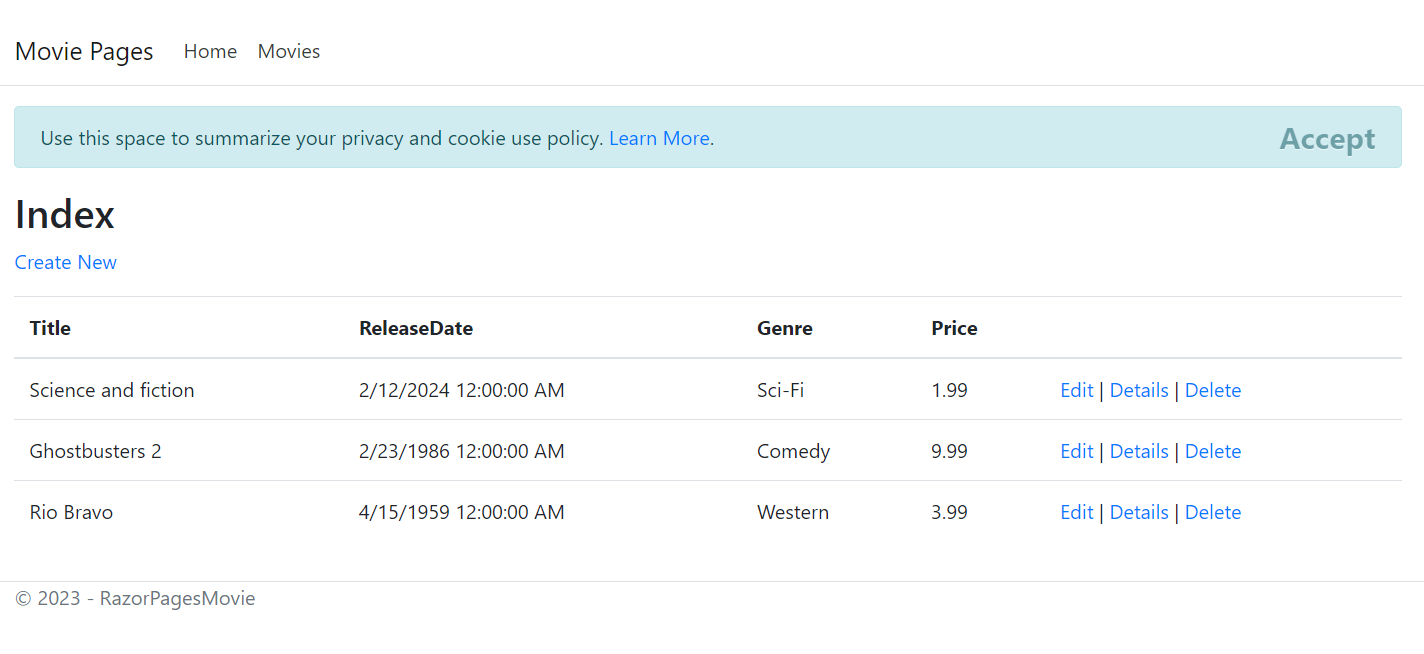
4. If Web Application is loaded, EC2 Windows server is successfully connected to the Web Application

5. Hover over Edit, Details, or Delete to verify if the website is operational.

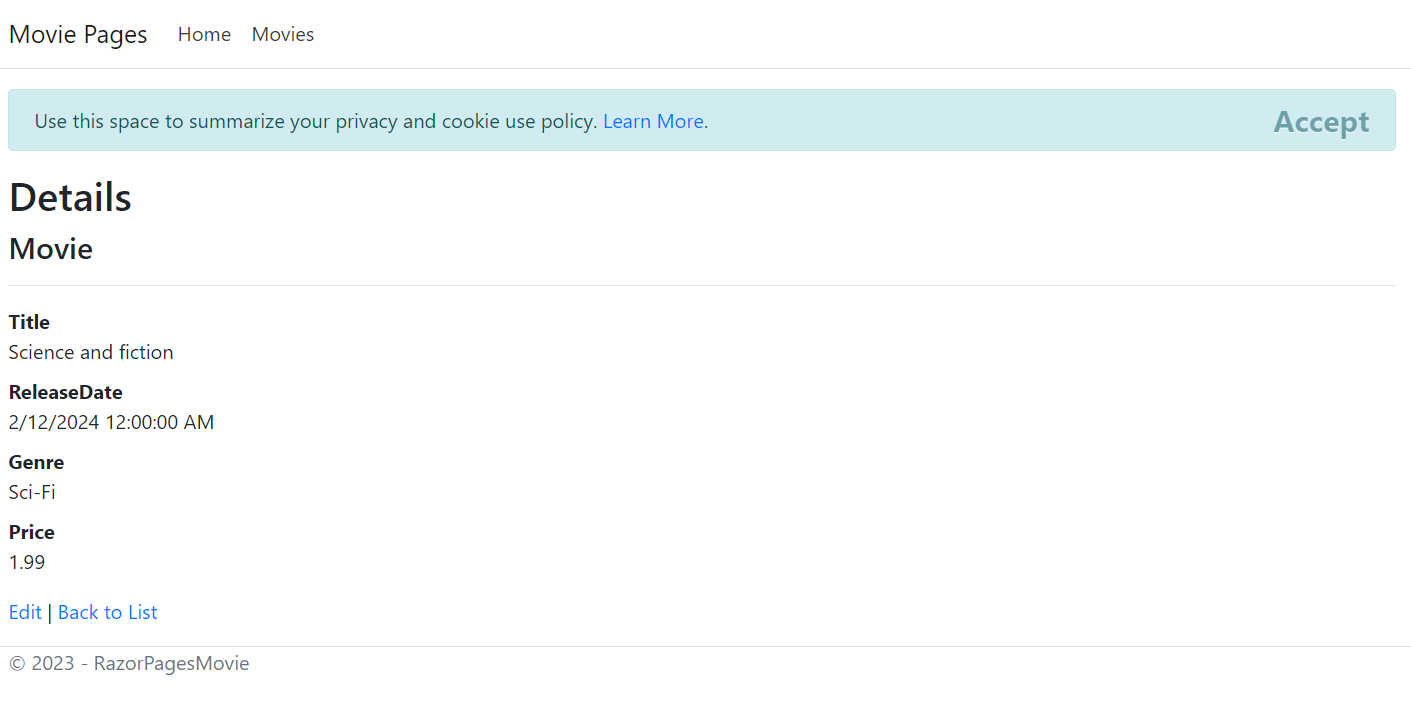
Before Edit



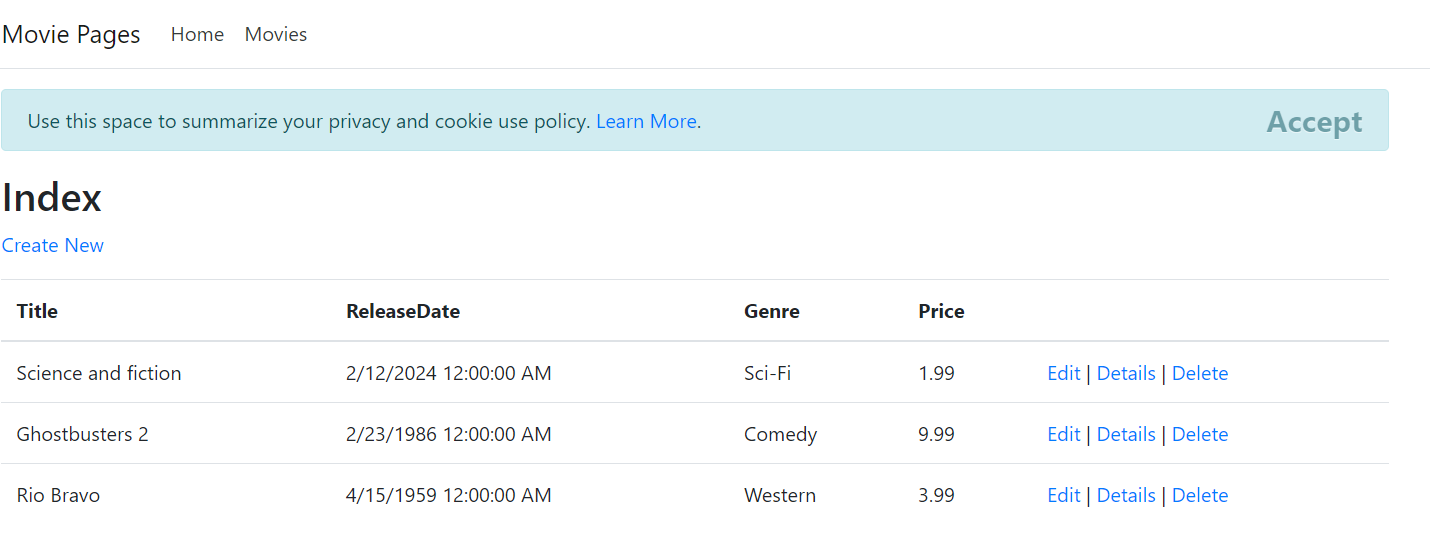
After Edit

The title has been changed to "Science and Fiction," the genre to "Sci-Fi," and the price to $1.99.

**Details**

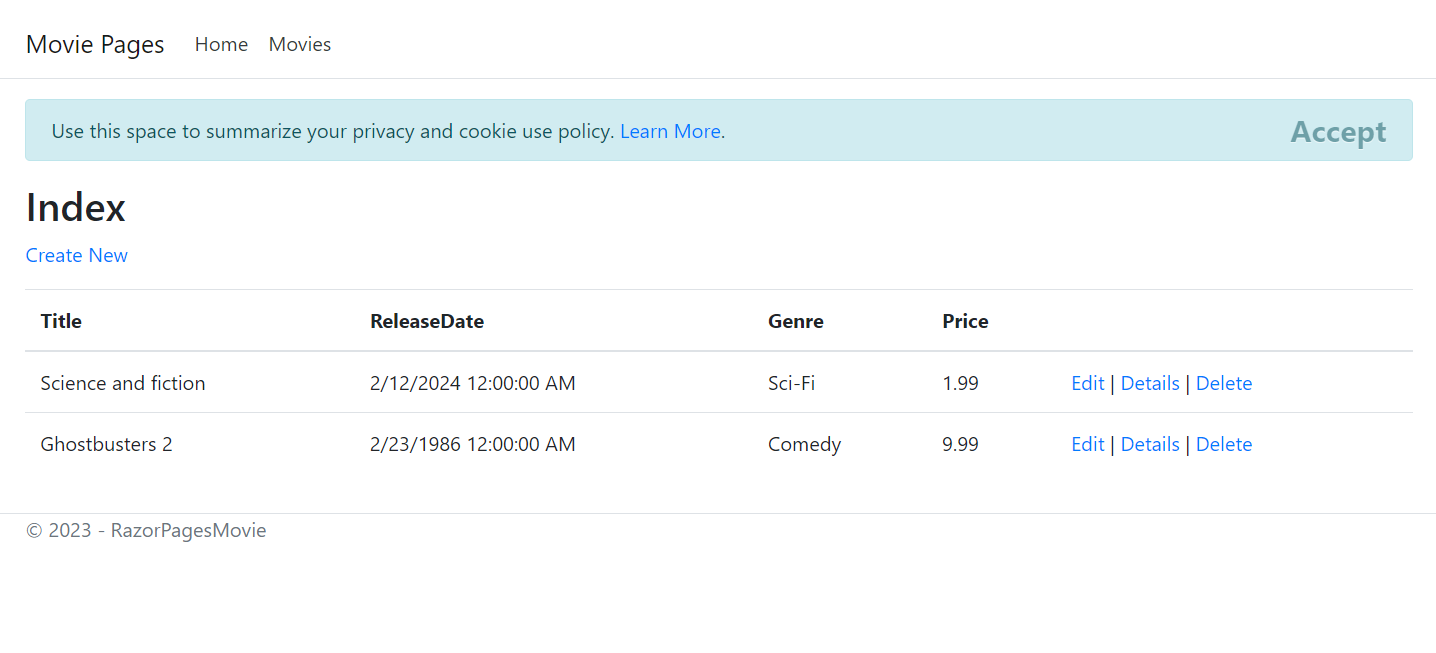


Before Delete



**After Delete**

The movie "Rio Bravo" has been deleted.



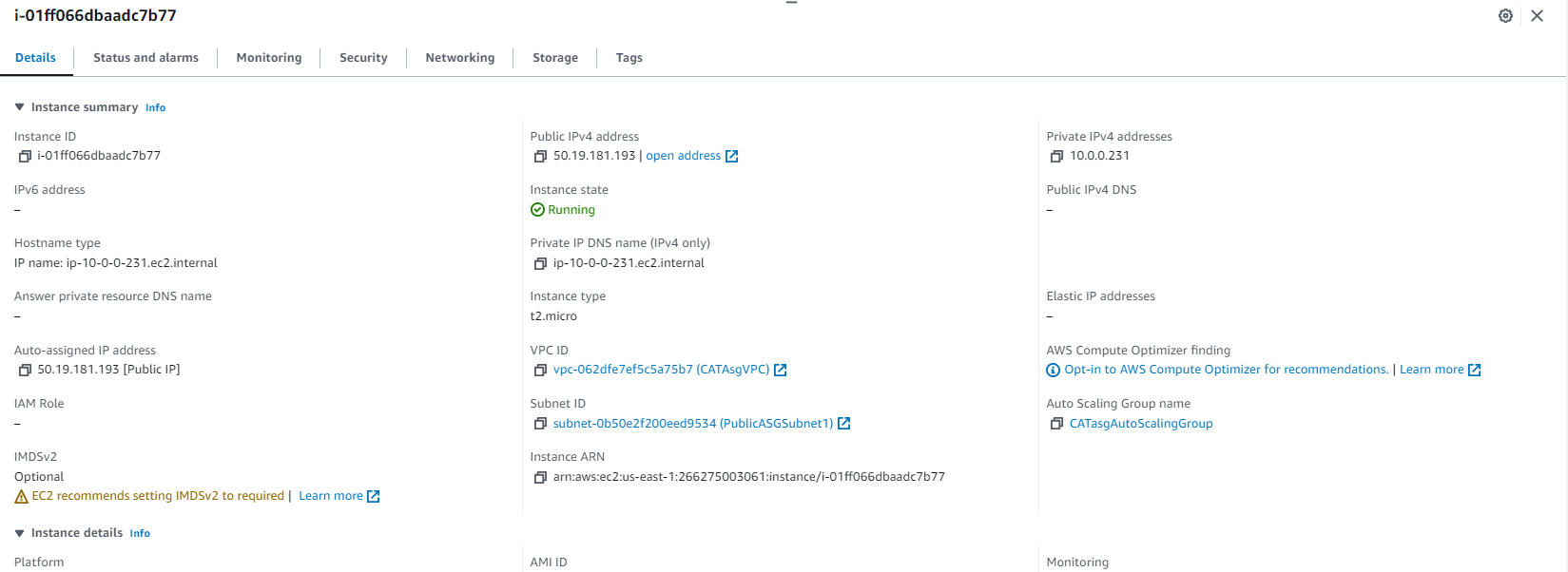
### CloudWatch Alarm & SNS notification

\*\*Note that the implementation video saved in google drive\*\*

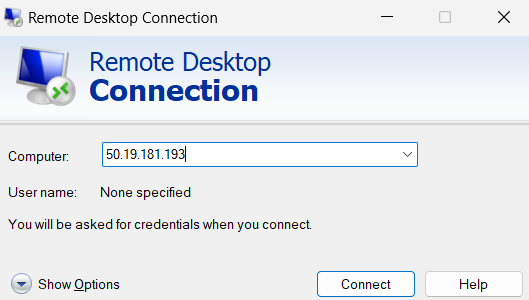
The testing steps are as follows:

1. Launch instances from the auto scaling group.

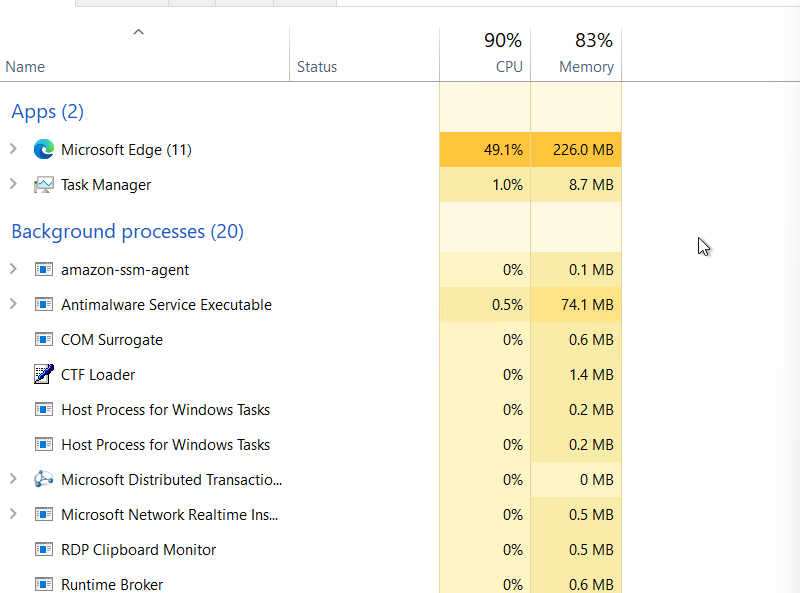
In this context, Public IPv4 Address is 50.19.181.193



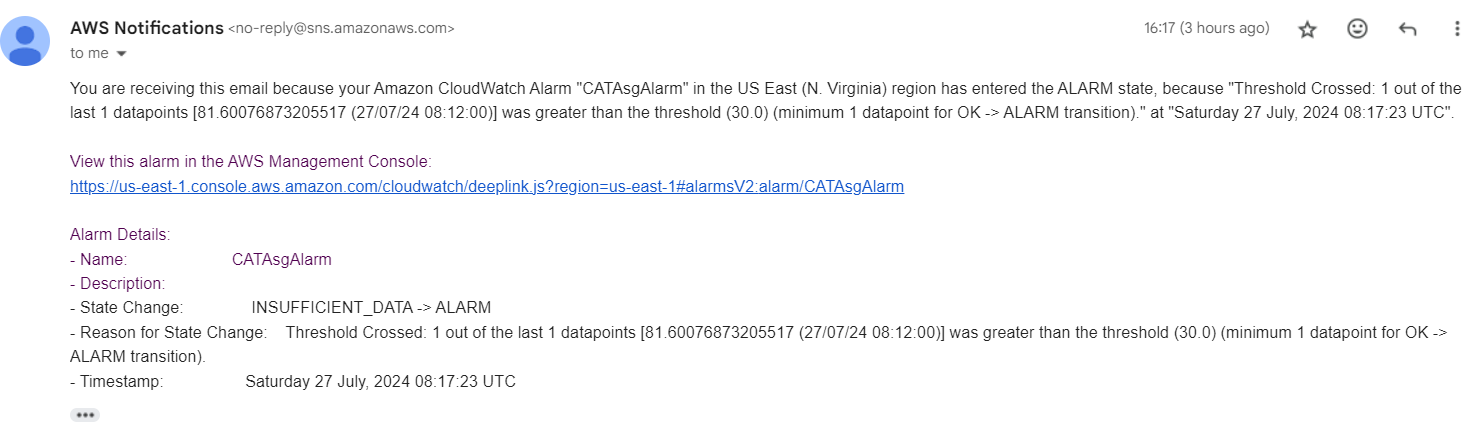
2. Connect to an instance via RDP

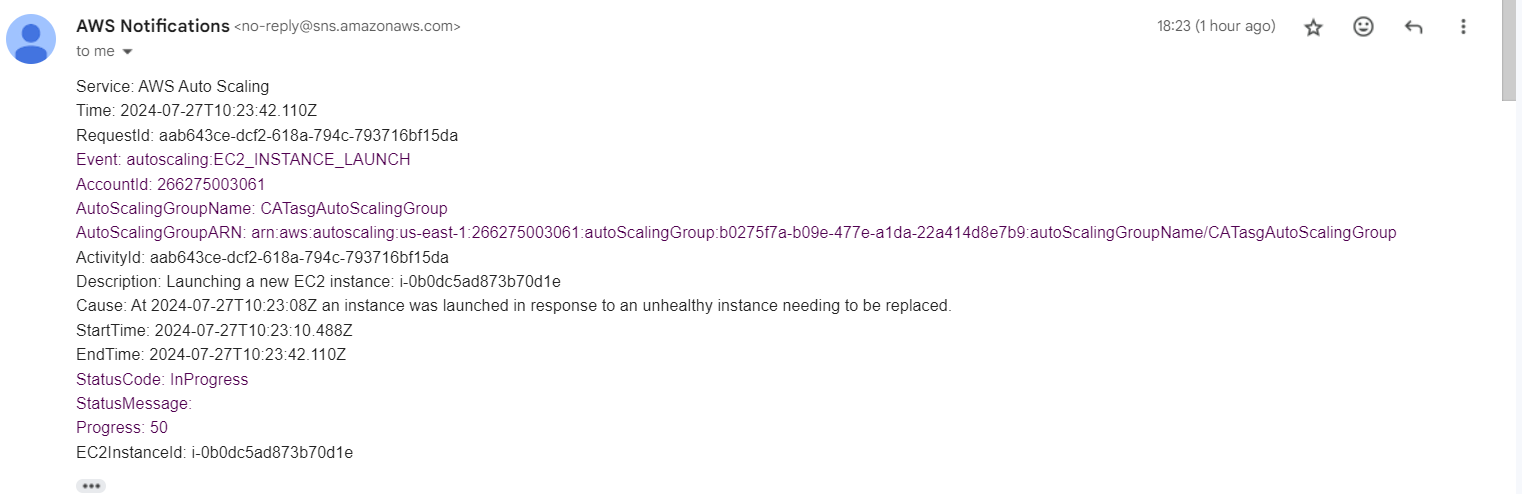


3. Increase the CPU usage of the Windows EC2 instance by starting high CPU usage activities such as playing high quality videos and downloading large files. Make use of Task Manager in the RDP to monitor CPU usage.

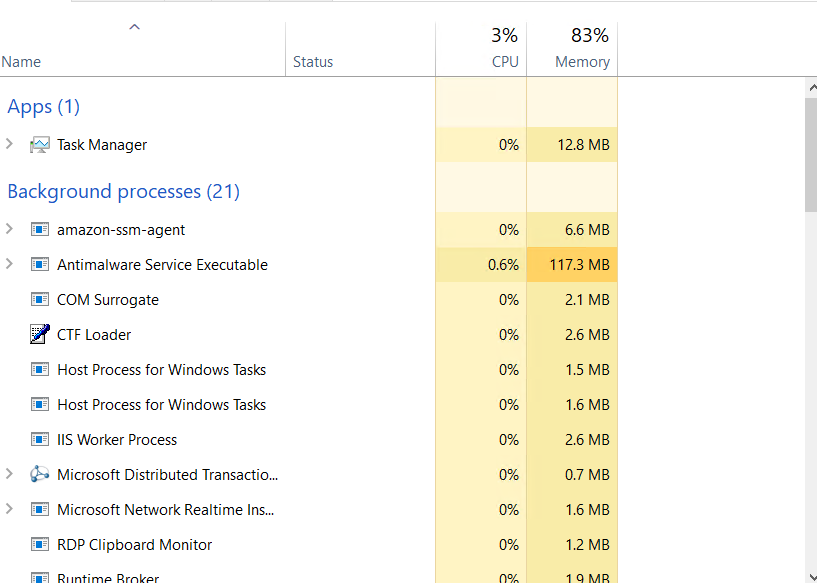


4. CloudWatch alarm for high CPU usage should trigger once CPU usage exceeds 30% for 3 datapoints within 3 minutes.

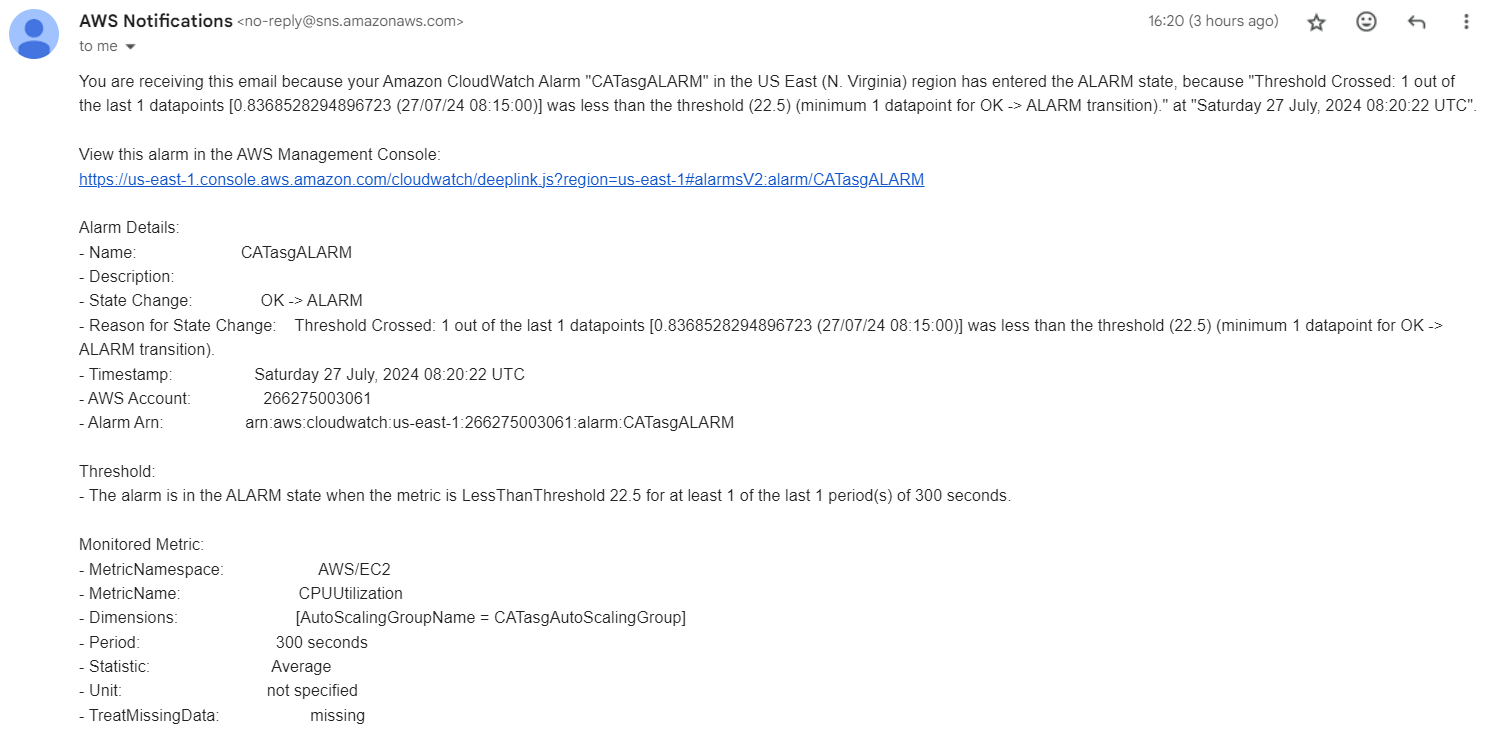
5. New Windows EC2 instances should be launched and SNS should be sent to subscribers of the SNS topic regarding the launching of new instances.



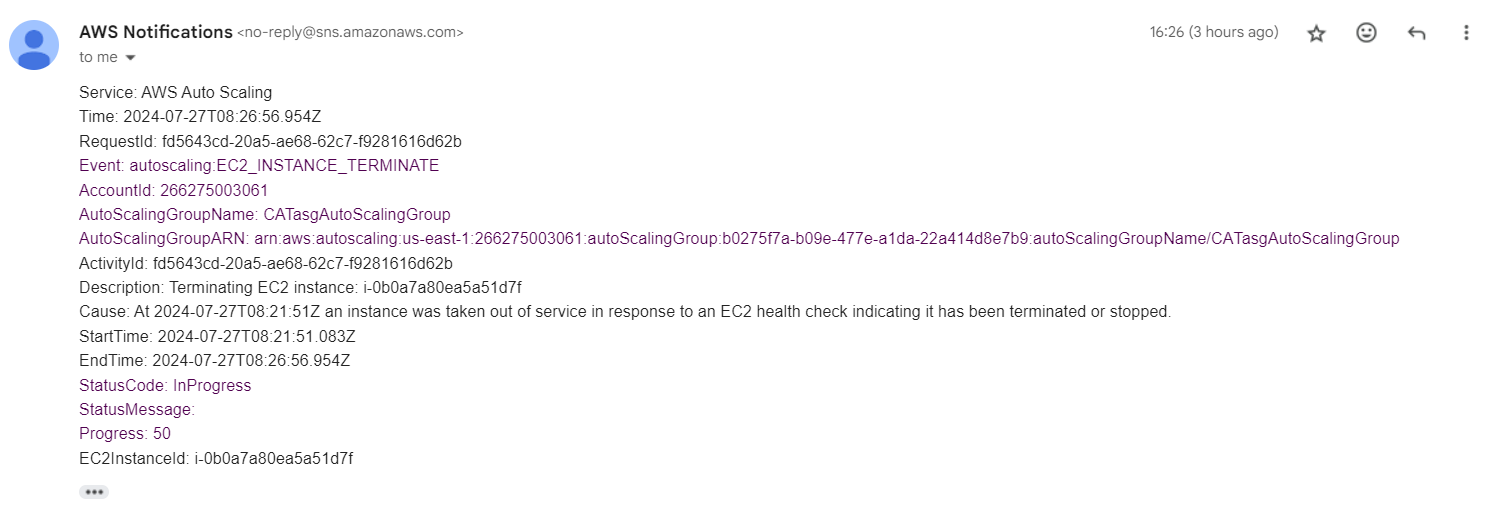
6. Decrease the CPU usage of the Windows EC2 instance by stopping all high CPU usage activities. Make use of Task Manager in the RDP to monitor CPU usage.



7. CloudWatch alarm for low CPU usage should trigger once CPU usage decrease below 22.5% for 15 datapoints within 15 minutes.



8. Windows EC2 instance should be terminated and SNS should be sent to subscribers of the SNS topic regarding the terminated instances.



## Additional

(ChangZe)

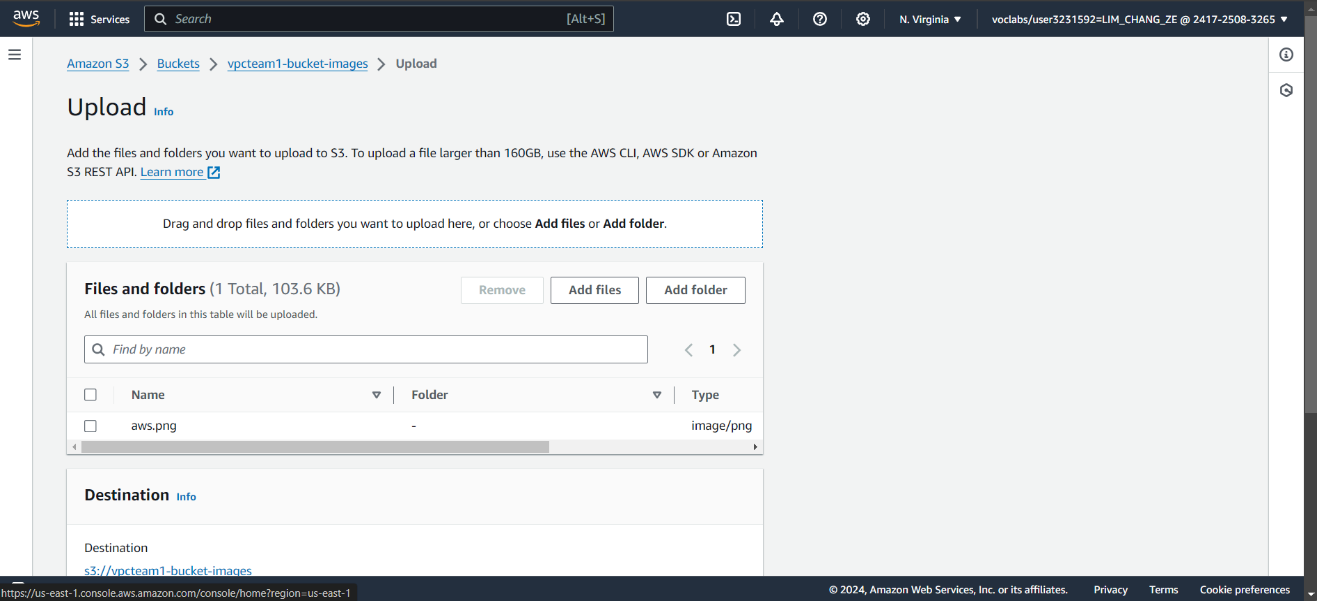
### Thumbnail Creation

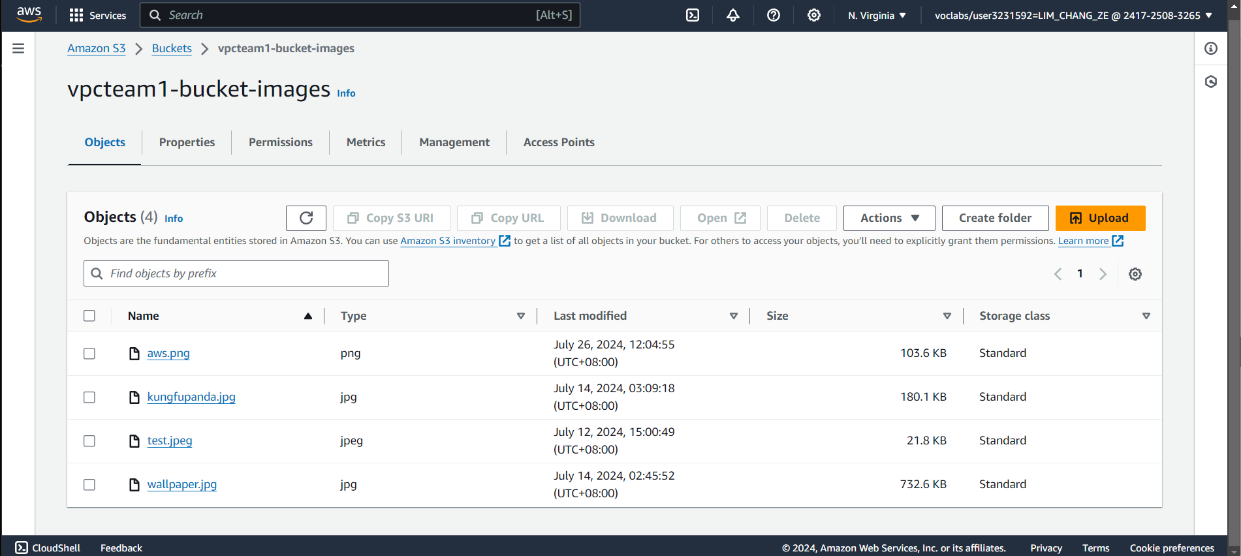
The testing steps are as follows:

* Upload an image to the vpcteam1-bucket-images.
* Check if the image has been created inside vpcteam1-bucket-thumbnail.
* Check for the email from SNS.

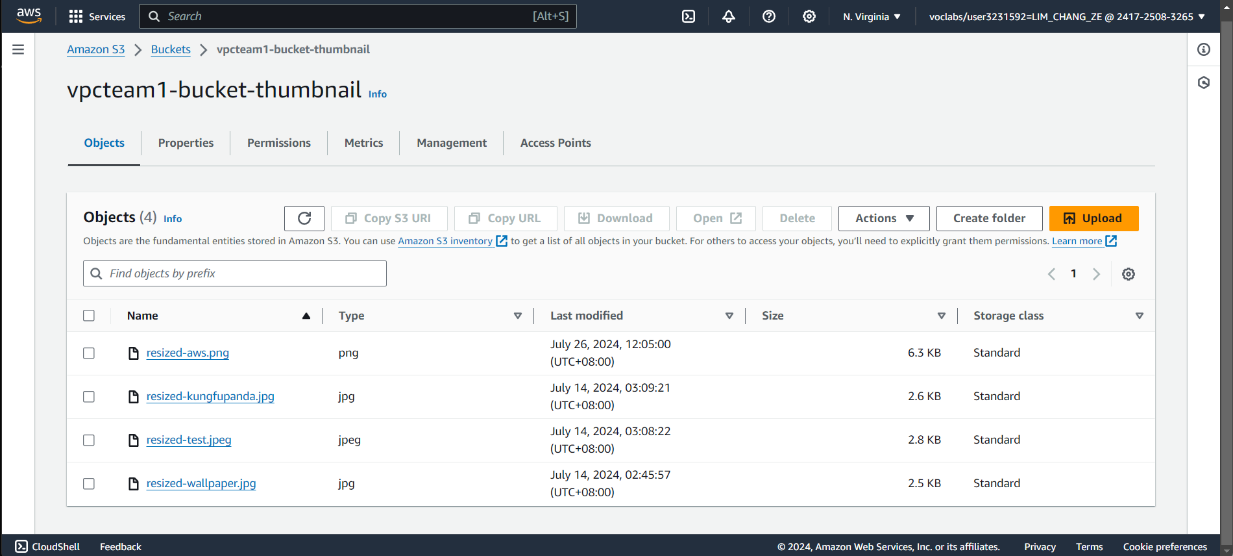


This is the full-sized picture that I will be using as a demonstration. It is named aws.png

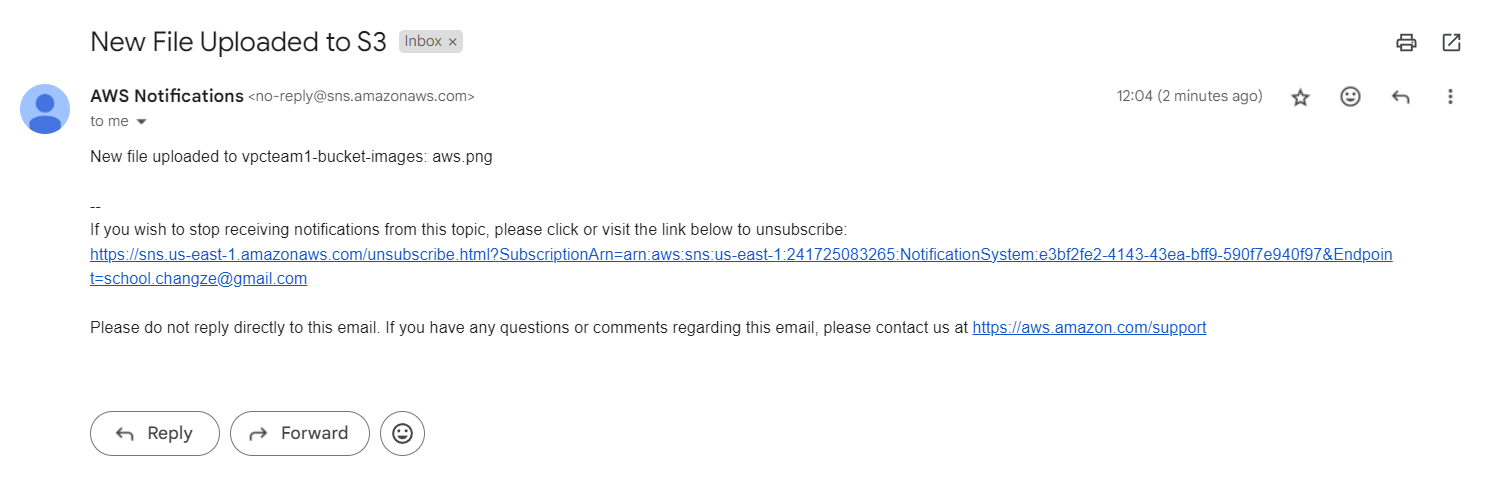




Upload the image. A resized image should be created inside the other S3 bucket.



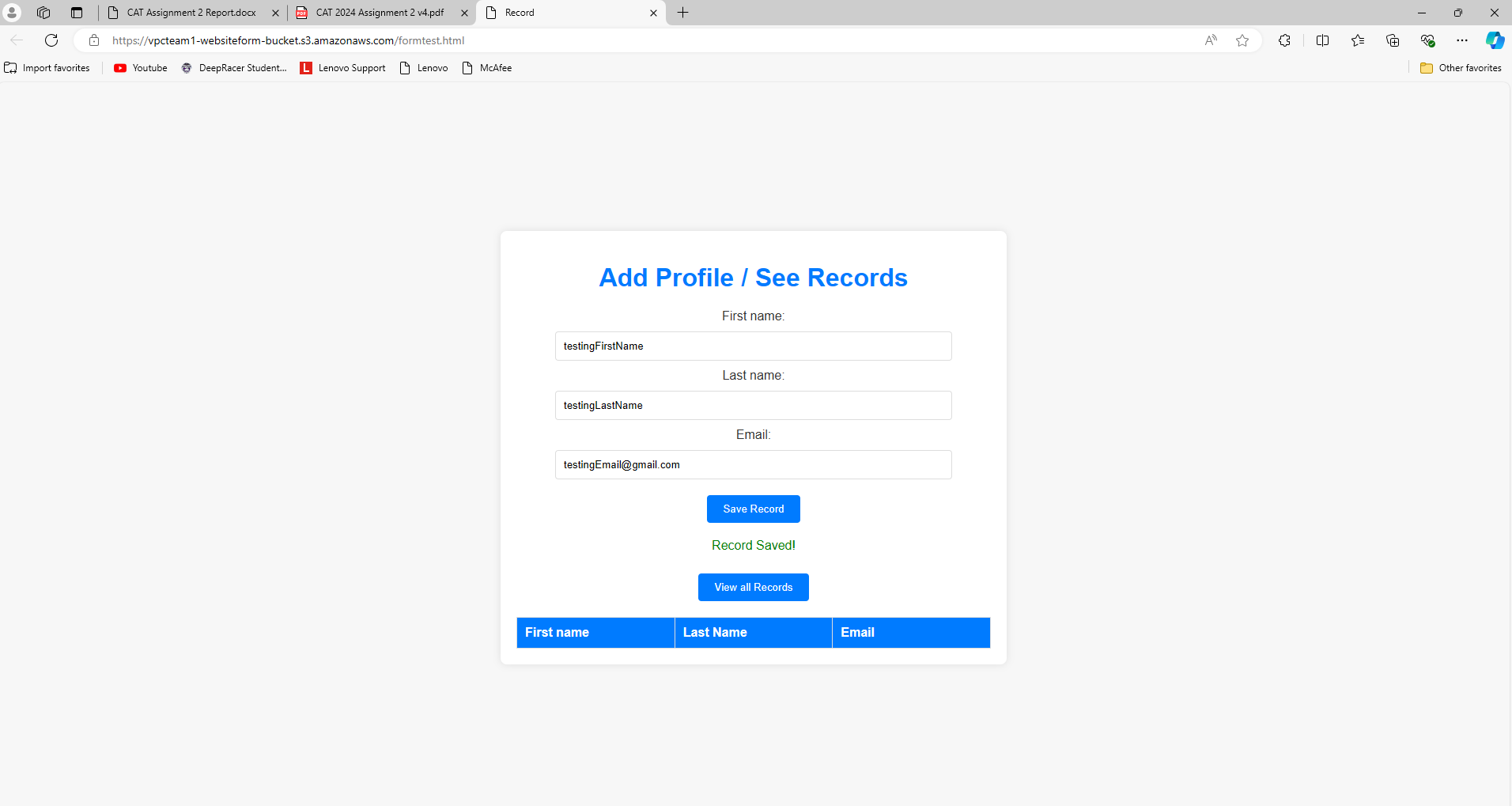
After uploading, an email should be sent to me.

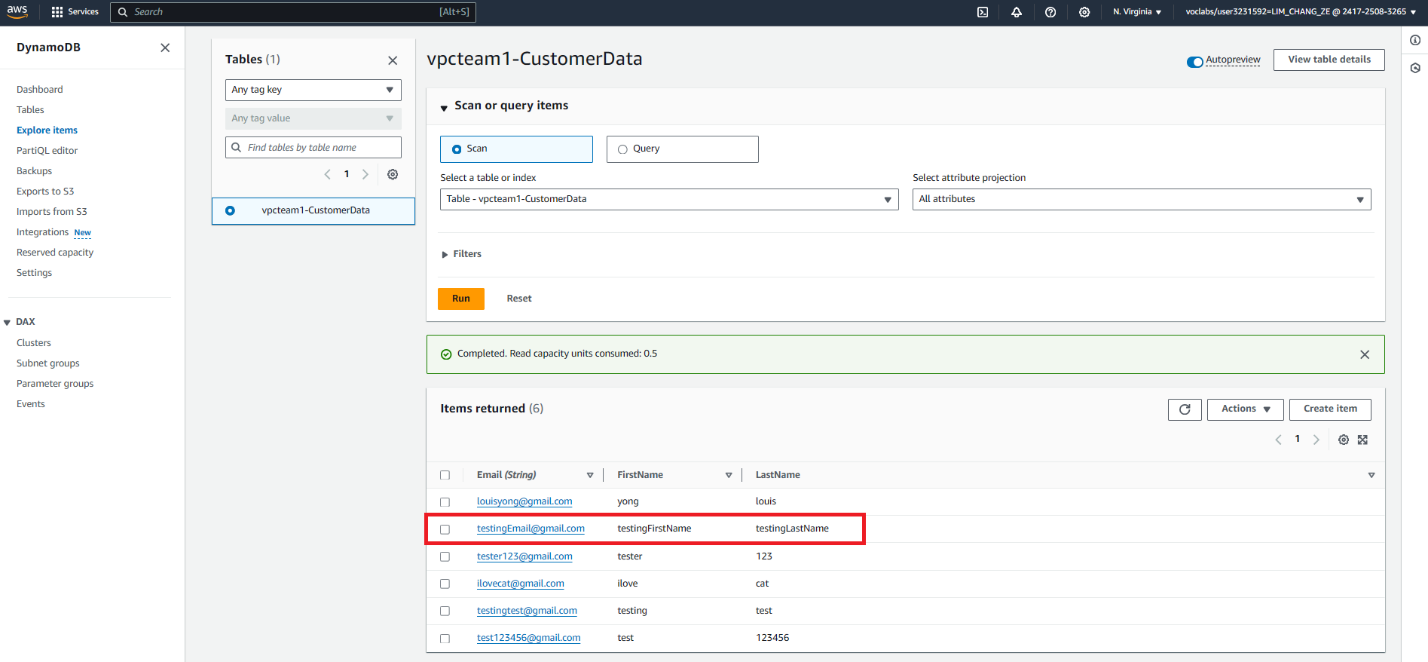


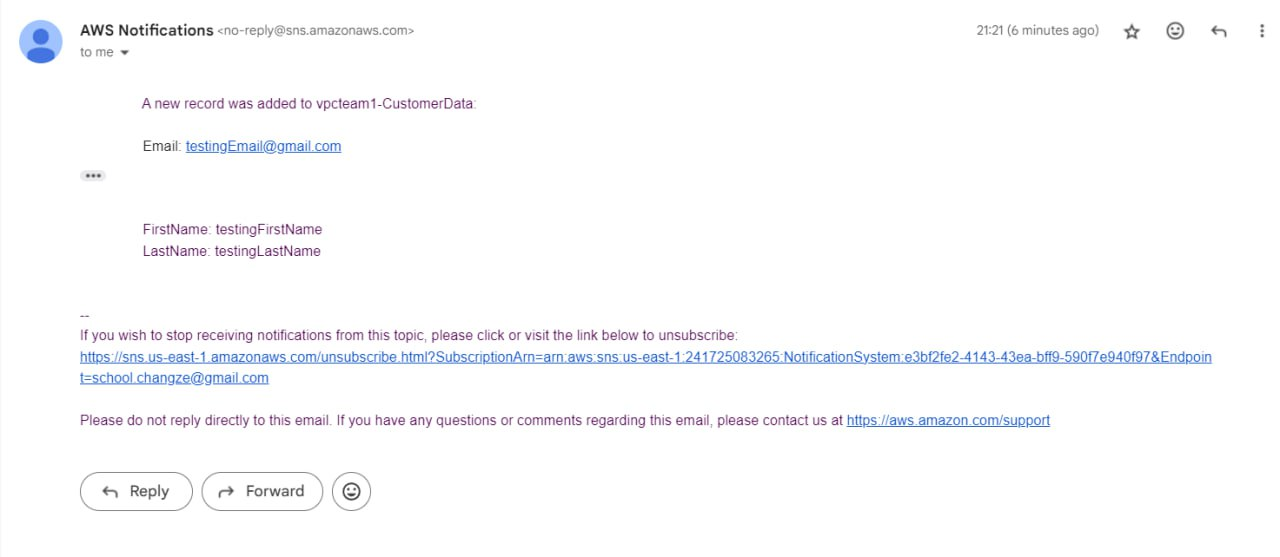
DynamoDB

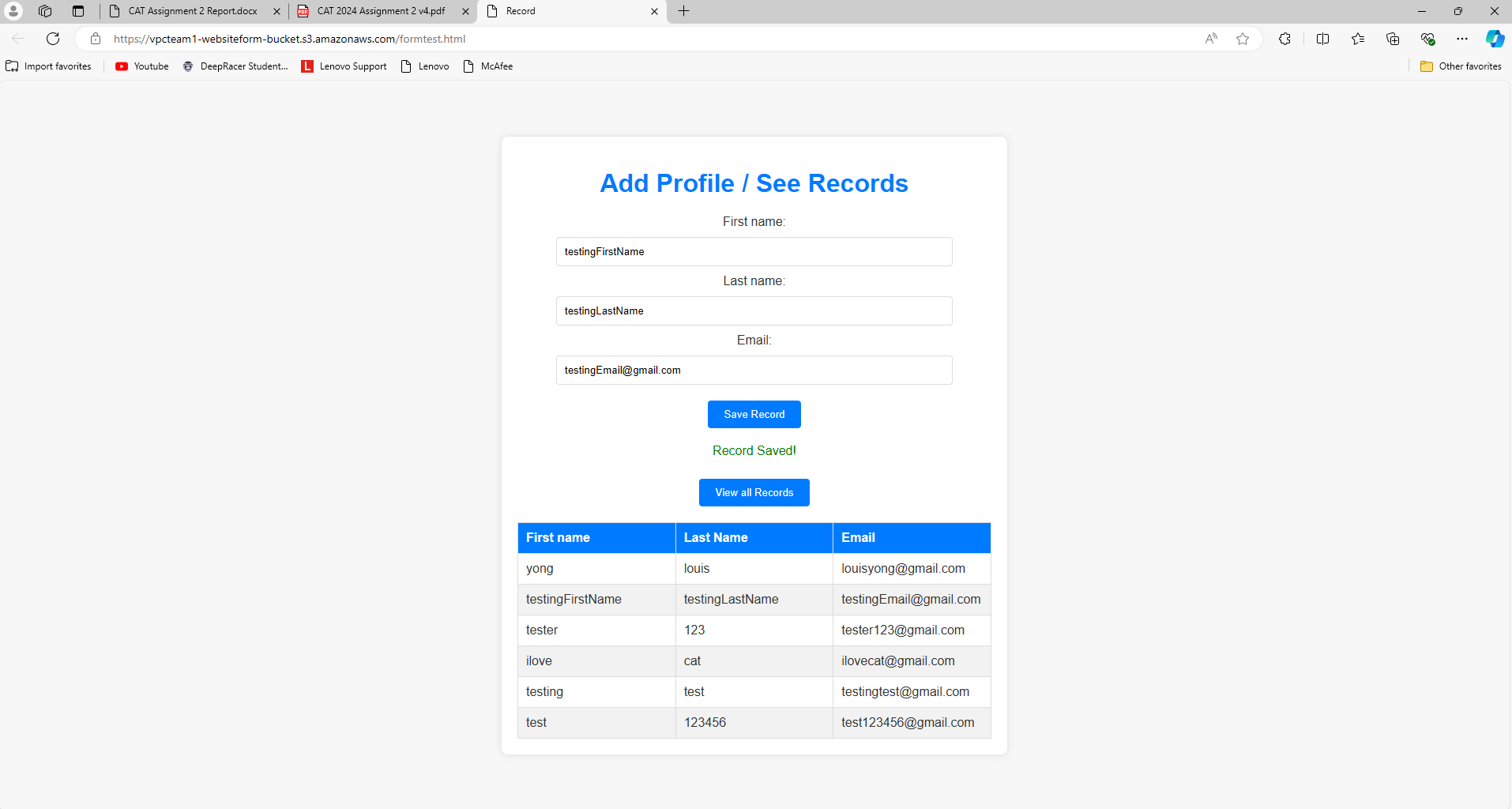
The steps to testing are as follows:

* Open the HTML form using the following link: <https://vpcteam1-websiteform-bucket.s3.amazonaws.com/formtest.html>
* Fill in the fields and submit the form.
* Ensure that the data from the form is added into DynamoDB.
* Ensure email is sent when record has been uploaded into the DynamoDB.
* See the details of all customer records in the database in the form.

Fill in the relevant fields and submit.

Check if data is added.

Check Email for SNS Notification.

Check if records can be displayed.

# 7. Incorporation of AWS Well-Architected Framework

(Louis)

There are six different pillars in the AWS Well-Architected Framework. These pillars are Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimisation, and Sustainability.

## Operational Excellence

Operational Excellence is the ability to support development and run workloads efficiently, allowing us to gain insights into the operations conducted and continuously improve supporting processes to ensure that it delivers business value.

This is how it is applied:

* Implementation of AWS CloudWatch allows us to monitor the state and health states of the EC2 instances and other services.
* CloudWatch alarms are set to notify us about the problems.
* Dashboards are created in CloudWatch to visualise the operational data and key metrics in real-time, allowing us to identify issues and problems quickly.

## Security

Security is the ability to take advantage of cloud technologies to protect data, systems, and assets, ensuring that these intangible items are protected and secure.

This is how it is applied:

* Using access-control lists or ACL, we can allow or prevent unauthorised access to confidential data, while other items that everyone can access such as the website form to enter customer data is allowed public access to anyone.
* Using specific security groups, we can control inbound and outbound network traffic to our instances and subnets, ensuring only authorised traffic is allowed. For example, the network traffic directed into the EC2 instances is only permitted from the HTTP, HTTPS or the RDS port.
* Using specially created key pairs, we can prove our identity when we connect to the EC2 instance(s), ensuring that unauthorised users have no access.

## Reliability

Reliability is the ability of a workload to perform or execute its intended function without error or any inconsistency when expected, which is inclusive of its ability to operate and test the workload entirely.

This is how it is applied:

* For the EC2 instances, auto scaling is implemented to ensure that the instances scale to the maximum capacity when demand reaches a certain threshold. This reduces the total workload of the current EC2 instances when the demand increases, ensuring resiliency and availability.
* EC2 instances and other applications are designed in multiple Availability Zones (AZ) for redundancy. When one of the AZs fails, the other AZs remain available for usage. The applications are ensured to stay fault tolerant even when there is a natural disaster that render the AZs not usable.
* S3 buckets are used instead of other file storage systems as it has cross region replication for data redundancy.
* An additional Relational Database (RDS) is implemented to ensure reliability. When the main RDS is affected, the backup RDS takes over as the main database to ensure fault tolerance.

Performance Efficiency

Performance efficiency is the ability to utilise computing resources with efficiency and meet system requirements while maintaining the current state of efficiency when the demand or technology changes.

This is how it is applied:

* With the implementation of load balancing, inbound traffic towards the EC2 instances is distributed among themselves. When an instance becomes unhealthy, the inbound traffic will be diverted to other healthy instances instead.
* Using serverless architectures, we can avoid managing servers allowing for high performance efficiency. AWS Lambda is used for event-driven computing, allowing for fast and efficient task automation and output.
* The usage of AWS DynamoDB allows for high-performance and scalable database solutions.

Cost Optimisation

Cost optimisation is the ability to run the systems such that they can deliver business value at the lowest price or cost. By building and operating cost-aware workloads, businesses can achieve desired outcomes while minimising costs and maximising their investment.

This is how it is applied:

* Auto scaling decreases the number of EC2 instances when there is lesser demand. This helps to reduce electricity and lower electricity costs as most cloud providers charge based on the total usage instead of maximum capacity.
* With serverless architectures, we can remove the need for managing servers, reducing costs associated with provisioning and maintaining infrastructure.

## Sustainability

Sustainability addresses the long-term environmental, economic, and societal impact of business’ activities, ensuring that the business is sustainable and minimising environmental damage.

This is how it is applied:

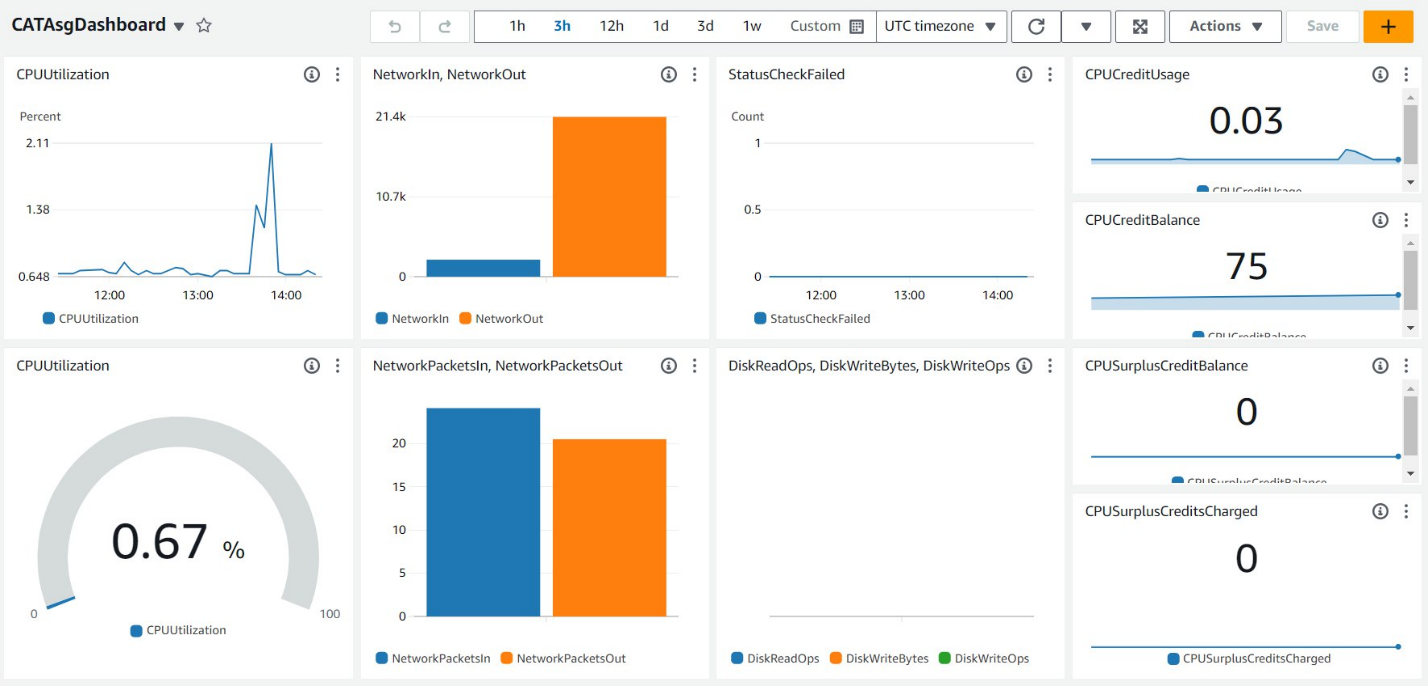
* Auto Scaling is implemented for right-sizing and to ensure that instances are used efficiently and not left to idle, helping to reduce overall energy consumption.
* Efficient code is written to optimise code, minimising computational requirements. This reduces the energy consumption taken to compute and run the code.
* Serverless architecture like AWS Lambda is used to run code when required, eliminating the need for always-on servers which consumes energy.

# 8. Explanation on Cloud Monitoring Plan

(Tej)

To ensure that of the other implemented cloud features are working as they should be, we created a cloud monitoring plan to allow the user to be notified when there is any form of disruption for any service. This plan includes 2 parts, CloudWatch Dashboard and CloudWatch Alarms.

## CloudWatch Dashboard

Above is a sample dashboard that we have created to monitor the EC2 Auto Scaling Group. We selected the metrics from the EC2 Auto Scaling Group that can be easily monitored. Below is an explanation of each dashboard visual:

* Top Left: The graph on the top left is a line chart of the CPU utilisation over time. This chart is useful to examine fluctuations in CPU utilisation in time or to check when the CPU utilisation crosses a certain threshold.
* Top Middle Left: The graph in the top middle left is a bar chart of two metrics, network (in bytes) in and network (in bytes) out. These metrics represent the number of bytes received by the EC2 instances’ network interfaces, and the number of bytes sent out by the EC2 instances’ network interfaces respectively. This chart is useful for comparing the network traffic in and out of the EC2 interfaces.
* Top Middle Right: The graph on the top middle right is a line chart of whether the EC2 instance has failed its status check over time. This graph allows us to check whether the EC2 instance has failed, and at what time the instance has failed.
* Bottom Middle Left: The graph in the bottom middle left is a bar chart of two metrics, network packets in and network packets out. These metrics represent the number of packets received by the EC2 instances’ network interfaces, and the number of packets sent out by the EC2 instances’ network interfaces respectively. This chart is useful for comparing the network traffic in and out of the EC2 interfaces.
* Bottom Middle Right: The graph on the bottom middle right is a bar chart comparing 3 statistics, DiskReadOps, DiskWriteOps and DiskWriteBytes. These 3 graphs compare the amount of disk operation being read and written, as well as the number of bytes written. This helps us check the storage of the system.
* Rightmost Column: The rightmost column consists of 4 Card visuals that show the number of credits (one vCPU running at 100% use rate for 1 minute) the EC2 instances are using, the total number of credits used since the EC2 instance was started and the number of credits that are in surplus that have been accrued and not paid down. These metrics help us make sure our EC2 usage is economical.

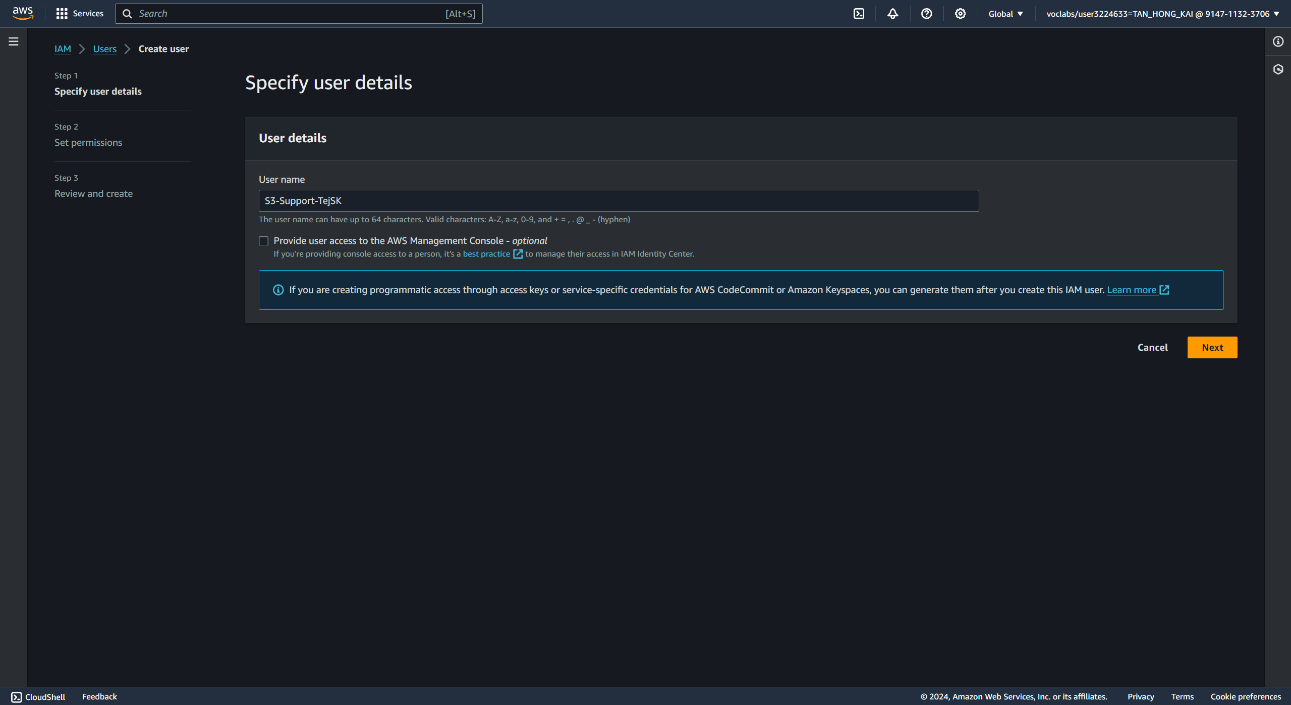
## CloudWatch Alarms

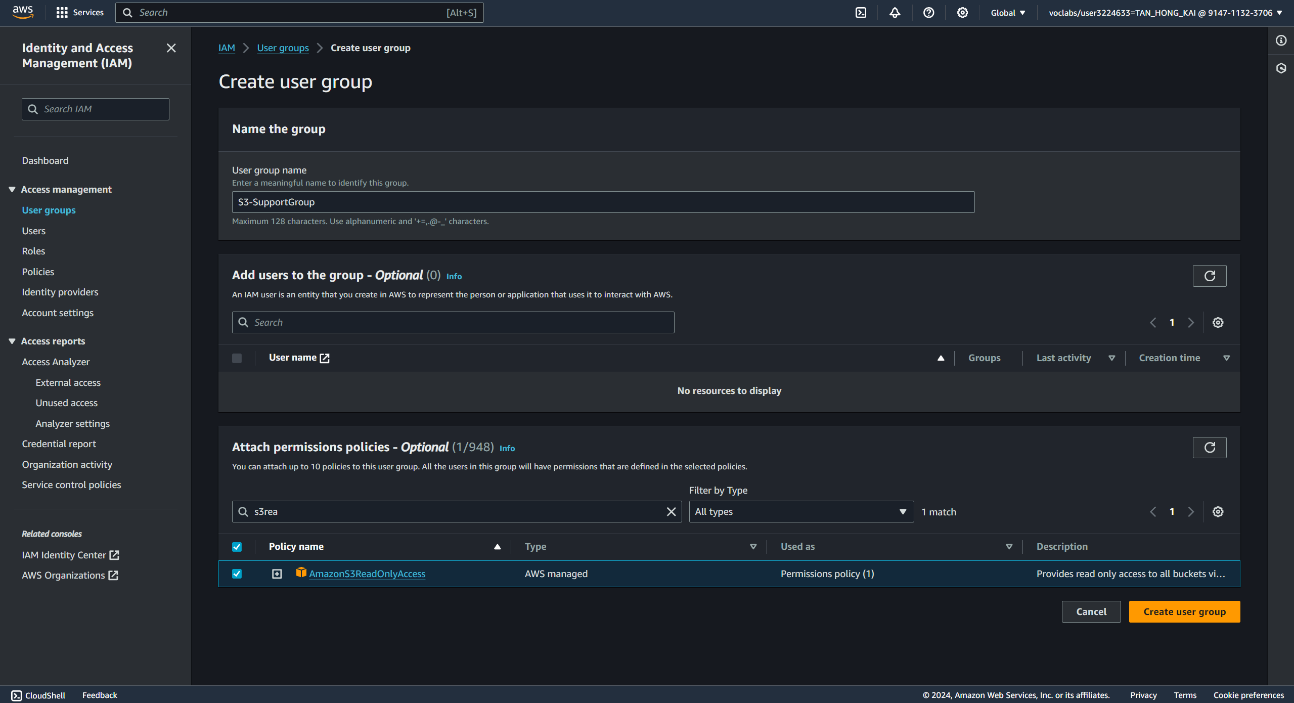
CloudWatch Alarms are used to inform us when a certain threshold on the EC2 instance is or is not being met. We can set this threshold ourselves. We have set up 2 CloudWatch Alarms, one to notify us if the CPU utilisation of the EC2 instance has surpassed 30% utilisation 3 times within a 30-minute timeframe, and another to notify us if the CPU utilisation of the EC2 instance has dropped below 22.5% utilisation 3 times within a 30-minute timeframe. When triggered, each alarm will use Amazon’s SNS system to send an email to an email group. This can be used to notify IT personnel if any alarm is triggered.

# 9. Relevant Screenshots and Diagrams

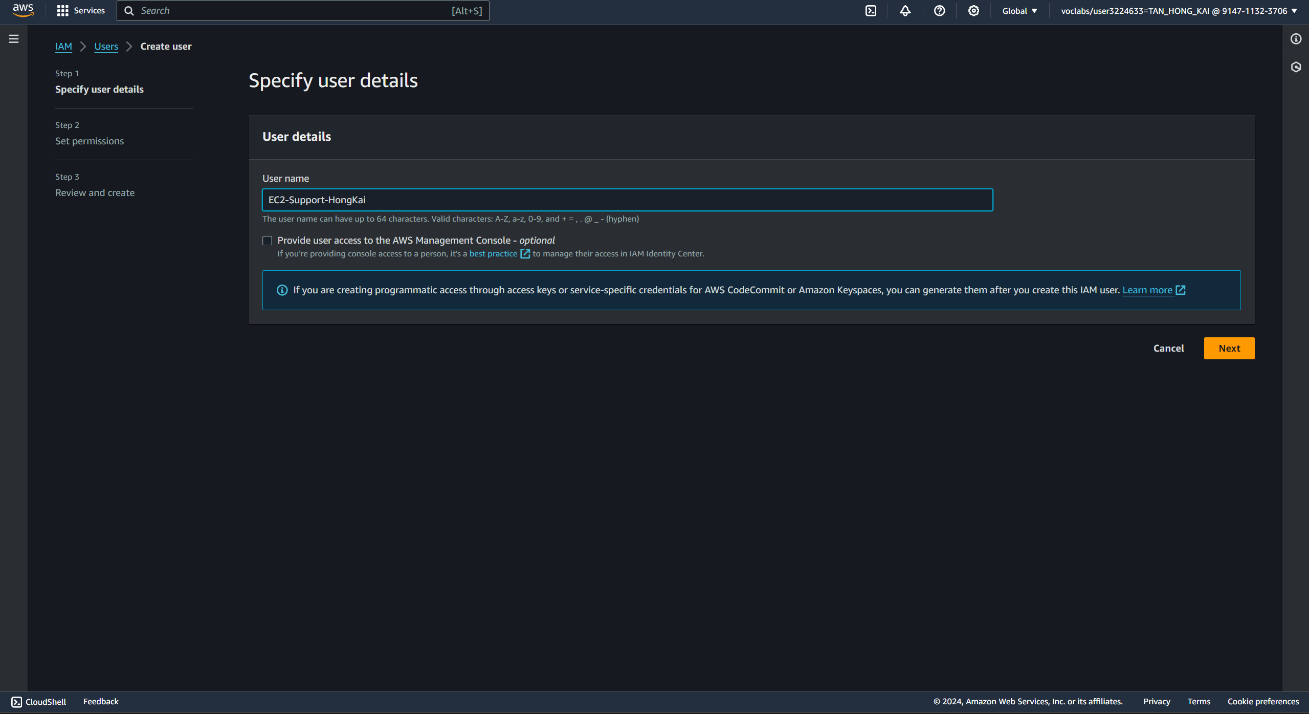
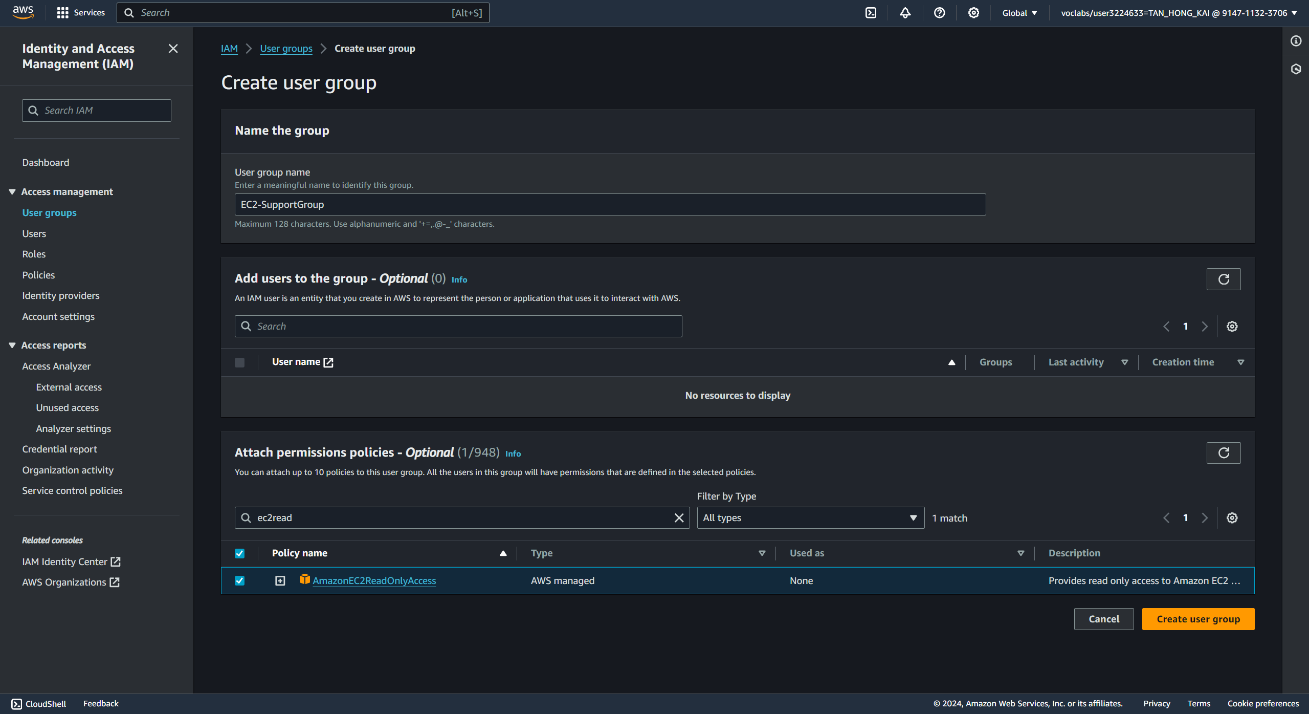
A few of the Hypothetical IAMs we would include

For example, if we have a technical S3-Support person, we can create a user and add the person into the user group





Same goes for the EC2 Support, this should also be done for the RDS Instances, but it was not included in the report.

For EC2 Admins however, appropriate policies should be in place, like the Customer inline JSON code below

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"ec2:Describe\*",

"ec2:StartInstances",

"ec2:StopInstances"

],

"Resource": [

"\*"

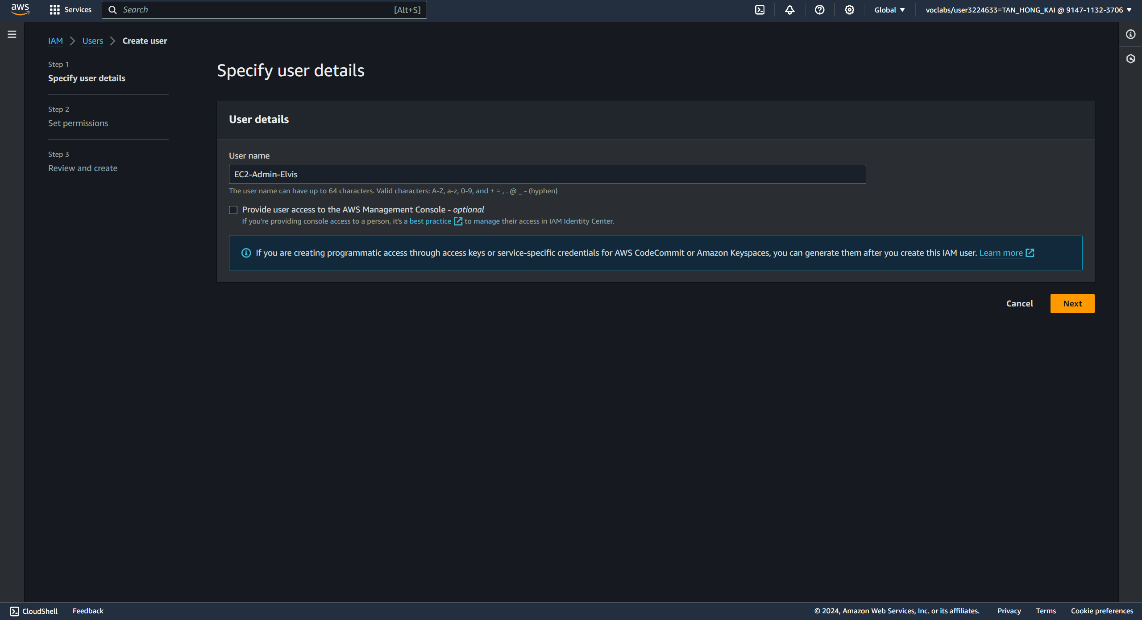
],

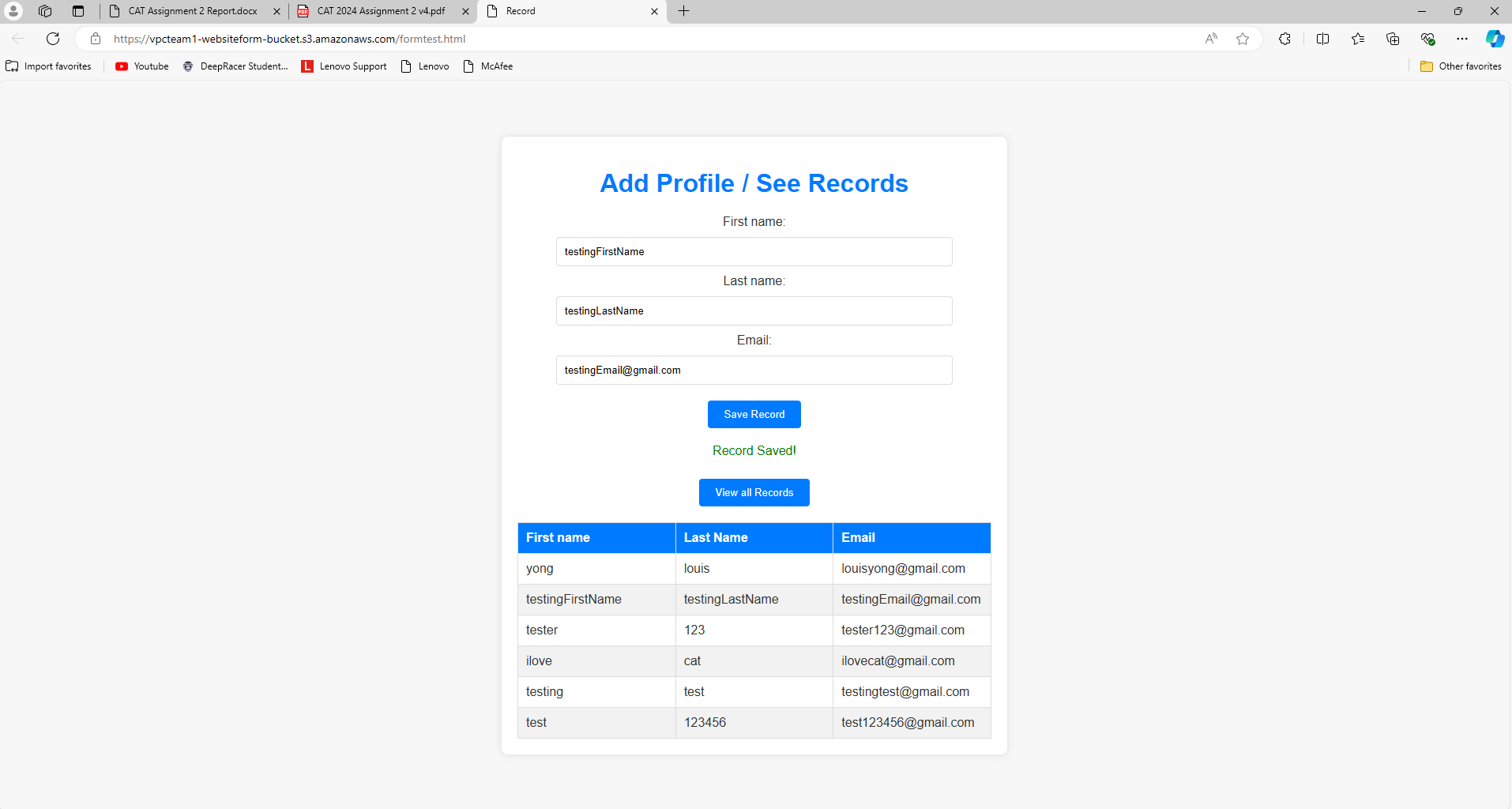
"Effect": "Allow"

}

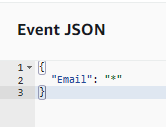
]

}



Customer Form Website: 

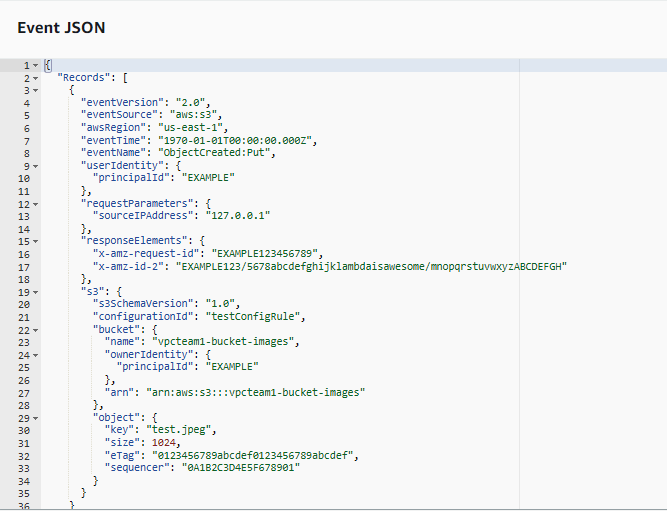
Test Event for GET records:



Test Event for POST records to DynamoDB:



Test event for thumbnail creation:



# 10. References

* The 6 Pillars of the AWS Well-Architected Framework | Amazon Web Services. (2022, March 1). Amazon Web Services. <https://aws.amazon.com/blogs/apn/the-6-pillars-of-the-aws-well-architected-framework/>
* Wikipedia contributors. (2024, July 5). Autoscaling. Wikipedia. <https://en.wikipedia.org/wiki/Autoscaling>
* ChatGPT. (n.d.). <https://chatgpt.com/c/40b0f184-b7cd-46f7-9cf6-93fba820cab4>
* Sarthak. (2018, September 13). How create a serverless website using AWS Lambda, AWS S3, AWS API Gateway and AWS DynamoDB with node.js? Full Tutorial! The Web Spark. <https://sarthaksrivastavablog.wordpress.com/2018/08/22/how-create-a-serverless-website-using-aws-lambda-aws-s3-aws-api-gateway-and-aws-dynamodb-with-node-js-full-tutorial/>
* Safe. (2023, November 13). Basics of AWS Cloud architecture - AWS in Plain English. *Medium*. <https://aws.plainenglish.io/basics-of-aws-cloud-architecture-cf949129b824>