R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai

Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade

An ISO 21001:2018 Certified Institution

DEPARTMENTOFCOMPUTER SCIENCE AND ENGINEERING

LABMANUAL

24CS908 – CLOUD ARCHITECTING

(LAB INTEGRATED)

Academic Year	**:**	**2025-2026**
Regulations	**:**	**2022**
Batch	**:**	**2024-2028**
Year / Semester	**:**	**III / V**

Prepared By,

Mr. V. M. Jemin, ASP/CSE

Ms.S.Shanthasheela AP / CSE

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai

Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade

An ISO 21001:2018 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABMANUAL

24CS908 – CLOUD ARCHITECTING

(LAB INTEGRATED)

Academic Year	**:**	**2025-2026**
Regulations	**:**	**2022**
Batch	**:**	**2023-2027**
Year / Semester	**:**	**III / V**

Prepared By

Mr. V. M. Jemin, ASP/CSE

Ms.S.Shanthasheela AP / CSE

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

Institute Vision and Mission

Vision

To be knowledge hub of providing quality technical education and promoting research for building up of our nation and its contribution for the betterment of humanity

Mission

- To make the best use of state-of-the-art infrastructure to ensure quality technical education
- To develop industrial collaborations to promote innovation and research capabilities
- To inculcate values and ethics to serve humanity

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

DEPARTMENTOFCOMPUTER SCIENCE AND ENGINEERING

Vision

To be a source of knowledge in the field of Computer Science and Engineering to cater to the growing need of industry and society

Mission

- To avail state-of-the art infrastructure for adopting cutting edge technologies and encouraging research activities.
- To promote industrial collaborations for professional competency.
- To nurture social responsibility and ethics to become worthy citizens

ProgrammeEducationalObjectives(PEOs)

Graduates of Computer Science and Engineering Program will

1. Become a globally competent professional in all spheres and pursue higher education world over.
2. Successfully carry forward domain knowledge in computing and allied areas to solve complex real world engineering problems.
3. Continuously upgrade their technical knowledge and expertise to keep pace with the technological revolution.
4. Serve the humanity with social responsibility combined with ethics.

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

24908	CLOUD ARCHITECTING	L	T	P	C
		2	0	2	3

COURSEOBJECTIVES:

TheCourse will enable the learners:

- To make architectural decisions based on AWS architectural principles and best practices.
- To describe the features and benefits of Amazon EC2 instances, and compare and contrast managed and unmanaged database services.
- To create a secure and scalable AWS network environment with VPC and configure IAM for improved security and efficiency.
- To use AWS services to make infrastructure scalable, reliable, and highly available.
- To use AWS managed services to enable greater flexibility and resiliency in an infrastructure.

UNIT I	INTRODUCING CLOUD ARCHITECTING AND STORAGE LAYER	6+6

Cloud architecting - The AWS Well-Architected Framework - AWS global infrastructure - Amazon S3 - Amazon S3 Versioning - Storing data in Amazon S3 - Moving data to and from Amazon S3 - Amazon S3 Transfer Acceleration - Choosing Regions for your architecture.

List of Exercise/Experiments

1. Creating a Static Website for the Café.
2. Configure an S3 bucket to automatically encrypt all uploaded objects.
3. Set up a cross-region replication configuration for an S3 bucket.

UNIT II	COMPUTE LAYER AND DATABASE LAYER	6+6

Adding compute with Amazon EC2 - Choosing an Amazon Machine Image (AMI) to launch an Amazon EC2 instance - Selecting an Amazon EC2 instance type - Using user data to configure an EC2 instance - Adding storage to an Amazon EC2 instance - Amazon EC2 pricing options - Amazon EC2 considerations - Database layer considerations - Amazon Relational Database Service (Amazon RDS) - Amazon DynamoDB - Database security controls - Migrating data into AWS databases.

List of Exercise/Experiments:

1. Creating a Dynamic Website for the Café.
2. Creating an Amazon RDS database.
3. Migrating a Database to Amazon RDS.
4. Create a web application that stores data in a managed database using EC2 instances and Amazon RDS.

UNIT III	CREATING AND CONNECTING NETWORKS	6+6

Creating an AWS networking environment - Connecting your AWS networking environment to the internet - Securing your AWS networking environment - Connecting your remote network with AWS Site-to-Site VPN - Connecting your remote network with AWS Direct Connect - Connecting virtual private clouds (VPCs) in AWS with VPC peering - Scaling your VPC network with AWS Transit Gateway - AWS Transit Gateway - Connecting your VPC to supported AWS services. Securing User and Application Access: Account users and AWS Identity and Access Management (IAM) - Organizing users - Federating users - Multiple accounts.

List of Exercise/Experiments:

1. Creating a Virtual Private Cloud.
2. Creating a VPC Networking Environment for the Café.
3. Creating a VPC Peering Connection.
4. Configure a VPC with subnets, an internet gateway, route tables, and a security group, and connect an on-premises network to the VPC.

UNIT IV	RESILIENT CLOUD ARCHITECTURE	6+6

Scaling your compute resources - Scaling your databases - Designing an environment that's highly available – Monitoring - Reasons to automate - Automating your infrastructure - Automating deployments - AWS Elastic Beanstalk - Overview of caching - Edge caching - Caching web sessions - Caching databases.

List of Exercise/Experiments:

1. Controlling Account Access by Using IAM.
2. Creating Scaling Policies for Amazon EC2 Auto Scaling.
3. Creating a Highly Available Web Application.
4. Creating a Scalable and Highly Available Environment for the Café.
5. Streaming Dynamic Content Using Amazon CloudFront.

UNIT V	BUILDING DECOUPLED ARCHITECTURES, MICROSERVICES AND SERVERLESS ARCHITECTURE	6+6

Decoupling your architecture - Decoupling with Amazon Simple Queue Service (Amazon SQS) - Decoupling with Amazon Simple Notification Service (Amazon SNS) - Sending messages between cloud applications and on-premises with Amazon MQ. Introducing microservices - Building microservice applications with AWS container services - Introducing serverless architectures - Building serverless architectures with AWS Lambda - Extending serverless architectures with Amazon API Gateway - Orchestrating microservices with AWS Step Functions - Disaster planning strategies - Disaster recover patterns.

List of Exercise/Experiments:

1. Breaking a Monolithic Node.js Application into Microservices.
2. Implementing a Serverless Architecture on AWS.
3. Implementing a Serverless Architecture for the Café.
4. Creating an AWS Lambda Function and explore using AWS Lambda with Amazon S3.

TOTAL:30 +30=60 PERIODS

OUTCOMES:

After completing the course, students will have the ability to

CO1: Explain cloud architecture principles and AWS storage solutions.
CO2: Deploy and manage AWS compute and database resources securely.
CO3: Design and configure secure AWS networks using VPC and IAM.
CO4: Implement scalable and resilient AWS architectures with high availability.
CO5: Build decoupled and serverless applications using AWS services like Lambda.
CO6: Develop disaster recovery strategies for AWS environments.

REFERENCES:

1. AWS Certified Solutions Architect Official Study Guide by Joe Baron, Hisham Baz, Tim Bixler
2. Architecting the Cloud by Michael Kavis.
3. AWS Documentation (amazon.com)
4. AWS Skill Builder
5. AWS Academy Cloud Architecting Course - https://www.awsacademy.com/vforcesite/LMS_Login

SOFTWARE REQUIREME

- AWS accounts for each group
- Access to AWS Management Console
- Learning materials on AWS Well-Architected Framework, Amazon S3, andrelated concepts
- Computers with internet access

COURSE OUTCOMES	
CO1	Explain cloud architecture principles and AWS storage solutions.
CO2	Deploy and manage AWS compute and database resources securely
CO3	Design and configure secure AWS networks using VPC and IAM.
CO4	Implement scalable and resilient AWS architectures with high availability
CO5	Build decoupled and serverless applications using AWS services like lamda
CO6	Develop disaster recovery strategies for AWS environments.

Course Articulation Matrix

Course Code/ Course Name: 24CS908/ CLOUD ARCHITECTING

CourseOutcome–ProgrammeOutcomeMapping:

Course Outcomes (COs)	ProgrammeOutcomes(POs),ProgrammeSpecificOutcomes(PSOs)													
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
CO1	2	1	-	-	2	-	-	3	-	2	-	2	3	2
CO2	2	2	-	-	2	-	-	2	2	2	-	2	3	2
CO3	2	2	-	-	2	-	-	2	2	2	-	2	3	2
CO4	2	2	3	-	2	-	-	2	2	2	-	2	3	3
CO5	2	2	3	-	2	-	-	-	2	2	-	2	3	3
CO6	2	2	3	-	2	-	-	-	2	2	-	2	3	3

EXPERIMENTS WITH MAPPED CO

S.NO	NAME OF THE EXERCISE	Mapped CO
1	CREATING A STATIC WEBSITE FOR THE CAFÉ	CO1
2	CONFIGURE AN S3 BUCKET TO AUTOMATICALLY ENCRYPT ALL UPLOADED OBJECTS	CO1
3	SET UP A CROSS-REGION REPLICATION CONFIGURATION FOR AN S3BUCKET	CO1
4	CREATING A DYNAMIC WEBSITE FOR THE CAFÉ	CO2
5	CREATING AN AMAZON RDS DATABASE	CO2
6	MIGRATING A DATABASE TO AMAZON RDS	CO2
7	CREATE A WEB APPLICATION THAT STORES DATA IN A MANAGED DATABASE USING EC2 INSTANCES AND AMAZON RDS	CO2
8	CREATING A VIRTUAL PRIVATE CLOUD	CO3
9	CREATING A VPC NETWORKING ENVIRONMENT FOR THE CAFÉ	CO3
10	CREATING A VPC PEERING CONNECTION	CO3
11	CONFIGURE A VPC WITH SUBNETS, AN INTERNET GATEWAY, ROUTE TABLES, AND A SECURITY GROUP, AND CONNECT AN ON-PREMISES NETWORK TO THE VPC	C)3
12	CONTROLLING ACCOUNT ACCESS BY USING IAM	CO4
13	CREATING SCALING POLICIES FOR AMAZON EC2 AUTO SCALING	CO4
14	CREATING A HIGHLY AVAILABLE WEB APPLICATION	CO4
15	CREATING A SCALABLE AND HIGHLY AVAILABLE ENVIRONMENT FOR THE CAFÉ	CO4

16	STREAMING DYNAMIC CONTENT USING AMAZON CLOUDFRONT	CO4
17	BREAKING A MONOLITHIC NODE.JS APPLICATION INTO MICROSERVICES	CO5
18	IMPLEMENTING A SERVERLESS ARCHITECTURE ON AWS	CO5
19	IMPLEMENTING A SERVERLESS ARCHITECTURE FOR THE CAFÉ	CO5
20	CREATING AN AWS LAMBDA FUNCTION AND EXPLORE USING AWS LAMBDA WITH AMAZON S3	CO5

VIVA QUESTIONS

1. Creating a Static Website for the Café

1. What are the core components of a static website, and how are they used in your café website?
2. How did you ensure the café website is responsive and mobile-friendly?
3. What tags or elements did you use in HTML to represent the café menu and contact information?
4. What is the difference between a static and dynamic website, and why did you choose a static one for this café project?
5. How did you optimize images and other assets to improve loading speed on the website?

2. Configure an S3 Bucket to Automatically Encrypt All Uploaded Objects

1. What is the purpose of enabling default encryption on an S3 bucket?
2. What are the differences between SSE-S3 and SSE-KMS encryption methods in S3?
3. Does enabling default encryption encrypt existing objects in the S3 bucket? Why or why not?
4. How can you verify that an uploaded object is encrypted in an S3 bucket?
5. What is the advantage of using SSE-KMS over SSE-S3 for encryption in S3?

3. Creation of Views, Synonyms, Sequence, Indexes

1. What is Cross-Region Replication (CRR) in Amazon S3, and why is it used?
2. What are the prerequisites for enabling Cross-Region Replication on an S3 bucket?
3. How do you configure the IAM role or bucket policy for Cross-Region Replication?
4. What happens if you delete an object from the source bucket after replication?
5. How can you verify that Cross-Region Replication is working as expected?

4. Creating a dynamic website for the café

1. What is the difference between a static website and a dynamic website?
2. Which technologies did you use to make the website dynamic?
3. How does the website handle customer orders or feedback dynamically?
4. How did you connect the website to a database, and which database did you use?
5. How does your website ensure data security and user authentication (if any)?

5. Creating an Amazon Rds Database

1. What is Amazon RDS and what are its key features?
2. Which database engines are supported by Amazon RDS?
3. What configuration options must be specified when creating an RDS instance?
4. How does Amazon RDS handle backups and automatic failover?
5. How can you connect an application or client to your RDS database securely?

6. Migrating a Database to Amazon Rds

1. What is Amazon RDS, and what are its main advantages over a self-managed database?
2. What steps are involved in migrating an on-premises database to Amazon RDS?
3. How do you choose the right database engine when setting up RDS?
4. How do you ensure data consistency and minimal downtime during the migration?
5. What backup and security features does Amazon RDS provide for your migrated database?

7. Create a Web Application That Stores Data in a Managed Database Using Ec2 Instances And Amazon Rds

1. What is the role of an EC2 instance in your web application architecture?
2. How does your web application connect to the Amazon RDS database?
3. What are the security best practices for allowing EC2 instances to access an RDS database?
4. How do you handle database credentials in your web application?
5. What are the advantages of using a managed database (RDS) instead of hosting the database on the same EC2 instance?

8. Creating a Virtual Private Cloud

1. W hat is a Virtual Private Cloud (VPC) and why is it used in AWS?
2. What are the main components of a VPC?
3. How do you configure subnets in a VPC and what is the difference between public and private subnets?
4. What is the purpose of an Internet Gateway in a VPC?
5. How do Security Groups and Network ACLs work within a VPC to control traffic?

9. Creating a VPC Networking Environment for the Café

1. Why does the café's web application need a custom VPC instead of using the default VPC?
2. How did you design the subnets for the café's VPC (public vs. private subnets)?
3. What is the role of a route table in your VPC configuration?
4. How do you ensure secure internet access for instances in the public subnet?
5. How do you control access between different subnets within the café's VPC?

10.Creating A VPC Peering Connection

1. What is VPC Peering and why is it used?
2. What are the steps to create a VPC Peering connection between two VPCs?
3. How do route tables need to be updated to enable communication through a VPC Peering connection?
4. Can VPC Peering connections be established between VPCs in different AWS accounts or regions?
5. What are some limitations or security considerations of using VPC Peering?

1. **Configure A VPC With Subnets, An Internet Gateway, Route Tables, And A Security Group, And Connect An On-Premises Network To The VPC**

1. What is the purpose of creating public and private subnets in a VPC?
2. How does an Internet Gateway enable communication between resources in the VPC and the internet?
3. What is the role of route tables in managing traffic flow within a VPC?
4. How do Security Groups differ from Network ACLs in a VPC?
5. What methods can be used to connect an on-premises network to a VPC, and how does a VPN connection work in this scenario?

2. **Controlling Account Access By Using IAM**

1. What is AWS Identity and Access Management (IAM) and why is it important?
2. What is the difference between an IAM user, group, and role?
3. How do IAM policies work and how are they attached to users or roles?
4. How can you enforce multi-factor authentication (MFA) for IAM users?
5. What are IAM best practices to ensure secure account access control?

3. **Creating Scaling Policies For Amazon Ec2 Auto Scaling**

1. What is Amazon EC2 Auto Scaling and why is it used?
2. What is the difference between a simple scaling policy and a dynamic scaling policy?
3. How does a target tracking scaling policy work in EC2 Auto Scaling?
4. What types of CloudWatch alarms can trigger an Auto Scaling policy?
5. How do scaling policies help optimize cost and performance for an application?

4. **Creating A Highly Available Web Application**

1. What does high availability mean in the context of a web application?
2. How can using multiple Availability Zones (AZs) help achieve high availability?
3. What role does a load balancer play in a highly available web application?
4. How can Auto Scaling improve the availability and reliability of your web application?
5. What are some ways to handle failures in one region or availability zone to keep the application running?

15. Creating a scalable and highly available environment for the café

1. What is the difference between scalability and high availability in a cloud environment?
2. How does Auto Scaling help the café's web application handle varying customer traffic?
3. What components did you use to ensure high availability for the café's application?
4. How does a Load Balancer distribute traffic in your café's environment?
5. What steps would you take to handle a failure in one Availability Zone to keep the café's services running smoothly?

16. Streaming dynamic content using amazon cloudfront

1. What is Amazon CloudFront and how does it work?
2. How is streaming dynamic content different from serving static content with CloudFront?
3. What is an origin in CloudFront, and what origin types can you use for dynamic content?
4. How does CloudFront help improve the performance and availability of dynamic content delivery?
5. How can you secure dynamic content when using CloudFront?

17. **Breaking a Monolithic Node.Js Application into Microservices**

1. What is the difference between a monolithic architecture and a microservices architecture?
2. What are the main advantages of breaking a monolithic application into microservices?
3. How do microservices communicate with each other in your Node.js application?
4. What challenges can arise when deploying and managing multiple microservices?
5. How did you handle data storage and databases when splitting the monolithic application into microservices?

18. **Implementing a Serverless Architecture On Aws**

1. What does **serverless architecture** mean and how is it different from traditional server-based architecture?
2. Which AWS services are commonly used to build a serverless application?
3. What is AWS Lambda and how does it work in a serverless setup?
4. How do you trigger a Lambda function and pass data to it?
5. What are the advantages and limitations of using a serverless architecture?

19. **Implementing a Serverless Architecture for The Café**

1. Why would the café benefit from using a serverless architecture instead of traditional servers?
2. Which AWS serverless services did you use to build the café's application?
3. How does AWS Lambda handle incoming requests for the café's website or app?
4. How did you store and manage the café's data in your serverless setup?
5. How does using a serverless architecture help the café handle sudden spikes in customer traffic?

20. **creating an aws lambda function and explore using aws Lambda with amazon s3**

1. What is AWS Lambda and how does it work?
2. What are the main steps to create and deploy a Lambda function?
3. How can you trigger a Lambda function using Amazon S3 events?
4. Give an example use case of using Lambda with S3 in your lab experiment.
5. What are the advantages of using Lambda functions with S3 compared to traditional server-based processing?

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

LIST OF EXPERIMENTS

S.NO	NAME OF THE EXERCISE
1	CREATING A STATIC WEBSITE FOR THE CAFÉ
2	CONFIGURE AN S3 BUCKET TO AUTOMATICALLY ENCRYPT ALL UPLOADED OBJECTS
3	SET UP A CROSS-REGION REPLICATION CONFIGURATION FOR AN S3BUCKET
4	CREATING A DYNAMIC WEBSITE FOR THE CAFÉ
5	CREATING AN AMAZON RDS DATABASE
6	MIGRATING A DATABASE TO AMAZON RDS
7	CREATE A WEB APPLICATION THAT STORES DATA IN A MANAGED DATABASE USING EC2 INSTANCES AND AMAZON RDS
8	CREATING A VIRTUAL PRIVATE CLOUD
9	CREATING A VPC NETWORKING ENVIRONMENT FOR THE CAFÉ
10	CREATING A VPC PEERING CONNECTION
11	CONFIGURE A VPC WITH SUBNETS, AN INTERNET GATEWAY, ROUTE TABLES, AND A SECURITY GROUP, AND CONNECT AN ON-PREMISES NETWORK TO THE VPC
12	CONTROLLING ACCOUNT ACCESS BY USING IAM
13	CREATING SCALING POLICIES FOR AMAZON EC2 AUTO SCALING

14	CREATING A HIGHLY AVAILABLE WEB APPLICATION
15	CREATING A SCALABLE AND HIGHLY AVAILABLE ENVIRONMENT FOR THE CAFÉ
16	STREAMING DYNAMIC CONTENT USING AMAZON CLOUDFRONT
17	BREAKING A MONOLITHIC NODE.JS APPLICATION INTO MICROSERVICES
18	IMPLEMENTING A SERVERLESS ARCHITECTURE ON AWS
19	IMPLEMENTING A SERVERLESS ARCHITECTURE FOR THE CAFÉ
20	CREATING AN AWS LAMBDA FUNCTION AND EXPLORE USING AWS LAMBDA WITH AMAZON S3

Ex.1: **CREATING A STATIC WEBSITE FOR THE CAFÉ**

AIM:

To write a program to implement the creation of Static website for the Café.

ALGORITHM:

Task 1: Creating a bucket in Amazon S3

1. Create Bucket:
    a. Go to AWS S3 service.
    b. Create a bucket with a unique name like "website-123".
    c. Enable public access and acknowledge it.
2. Add Tag:
    a. Go to bucket Properties.
    b. Add a tag like "Department: Marketing".
3. Configure for Website:
    a. In bucket Properties, enable static website hosting.
    b. Set index document to "index.html" and error document to "error.html".
4. Get Website Endpoint:
    a. Copy the Bucket website endpoint link.

Task 2: Uploading content to your bucket

1. Download Files:
    a. Right-click and download these files: index.html, script.js, style.css.
    b. Keep the file names the same, including the extensions.
2. Upload to S3 Bucket:
    a. Go to the Amazon S3 console and select your bucket.
    b. Click on the "Objects" tab.
    c. Choose "Upload" and then "Add files".
    d. Select the three files you downloaded.
    e. If asked, confirm overwriting existing files.
    f. Click "Upload".
3. Close:
    a. Once uploaded, click "Close".

Task 3: Enabling access to the objects

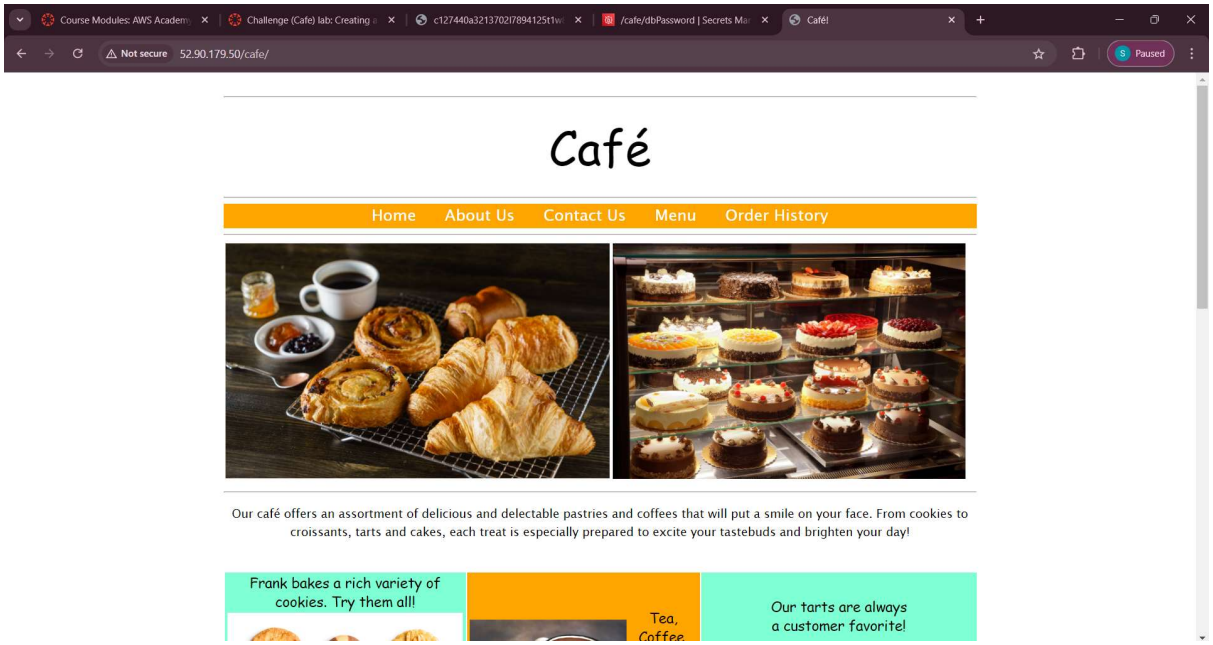
1. Confirm Objects are Private:
    a. Refresh the browser tab showing the 403 Forbidden message.
    b. If closed, go to Properties > Static website hosting > Endpoint link.
2. Make Objects Public:
    a. Go back to the Amazon S3 console tab.
    b. Select all three objects.
    c. From the Actions menu, choose "Make public via ACL".
    d. Confirm the list of objects and choose "Make public".
3. Check Website Access:
    a. Return to the browser tab with the 403 Forbidden message.
    b. Refresh the webpage.

Task 4: Updating the website

1. Edit HTML File:

    a. Open index.html in a text editor.
    b. Replace "Served from Amazon S3" with "Created by Your Name".
    c. Save the file.
2. Upload Edited File:
    a. Go to Amazon S3 console.
    b. Upload the edited index.html.
    c. Select the file, choose "Make public via ACL" from the Actions menu.

OUTPUT:



RESULT:

Thus the Program to implement the static website for the café was successfully executed.

Ex.2: CONFIGURE AN S3 BUCKET TO AUTOMATICALLY ENCRYPT ALL UPLOADED OBJECTS

AIM:

To configure an S3 bucket in AWS to automatically encrypt all uploaded objects, ensuring data security and compliance with encryption standards.

PROCEDURE:

Step 1: Create or Select an Existing S3 Bucket

1. Navigate to the Amazon S3 Console:
    - Sign in to your AWS Management Console.
    - Search for "S3" in the services search bar and select "Amazon S3" from the results.
2. Create a New Bucket (Optional):
    - If you don't already have a bucket, click on "Create bucket."
    - Enter a unique bucket name, choose the desired AWS region, and configure the other settings.
    - Click "Create bucket."

Step 2: Configure Default Encryption for the Bucket

1. Access the Bucket:
    - Go to the "Buckets" list in the S3 console.
    - Click on the name of the bucket you want to configure for encryption.
2. Go to the "Properties" Tab:
    - In the bucket settings, select the "Properties" tab.
3. Enable Default Encryption:
    - Scroll down to the "Default encryption" section.
    - Click on "Edit."
4. Choose Encryption Method:
    - Select the encryption type you want to apply. There are two main options:
        - SSE-S3 (Amazon S3-Managed Keys): AWS manages the encryption keys for you.
        - SSE-KMS (AWS Key Management Service-Managed Keys): You manage your encryption keys using AWS KMS, providing more control and auditing capabilities.
        - Optionally, SSE-C (Customer-Provided Keys) can also be used, but it requires you to manage and provide the encryption keys.
5. Select SSE-S3 or SSE-KMS:
    - For SSE-S3, select "AES-256" as the encryption algorithm. This option uses Amazon S3-managed keys.
    - For SSE-KMS, choose one of your existing KMS keys or create a new one by selecting "Create a KMS key" in the KMS console.
6. Save the Encryption Settings:
    - After selecting your preferred encryption method, click "Save changes."

Step 3: Verify the Encryption Settings

1. Upload an Object to the S3 Bucket:
    - Go to the "Objects" tab for the bucket and upload a test file.
    - Click "Upload" and add a file from your local system to verify the encryption settings.
2. Check the Encryption Status of the Uploaded Object:
    - Once the file is uploaded, go to the "Objects" tab and select the uploaded object.
    - Scroll down to the "Properties" section for the object.
    - You should see an Encryption section, showing either "AES-256" (for SSE-S3) or the KMS key ID (for SSE-KMS), confirming that the object was encrypted during the upload.

Step 4: Enable Bucket Policy for Encryption Compliance

1. Navigate to the "Permissions" Tab:
    - Go back to the S3 bucket and select the "Permissions" tab.
2. Add a Bucket Policy to Enforce Encryption:
    - Scroll down to the "Bucket policy" section and click "Edit."
    - Add the following policy to enforce that all objects uploaded to the bucket must be encrypted:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RequireEncryptedUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::your-bucket-name/*",
            "Condition": {
                "StringNotEquals": {
                   "s3:x-amz-server-side-encryption": "AES256"
                }
            }
        }
    ]
}
```

    - Replace your-bucket-name with your actual bucket name.
    - This policy ensures that any attempt to upload an object without encryption will be denied.
3. Save the Bucket Policy:
    - Click "Save changes."

Step 5: Test and Validate Encryption Enforcement

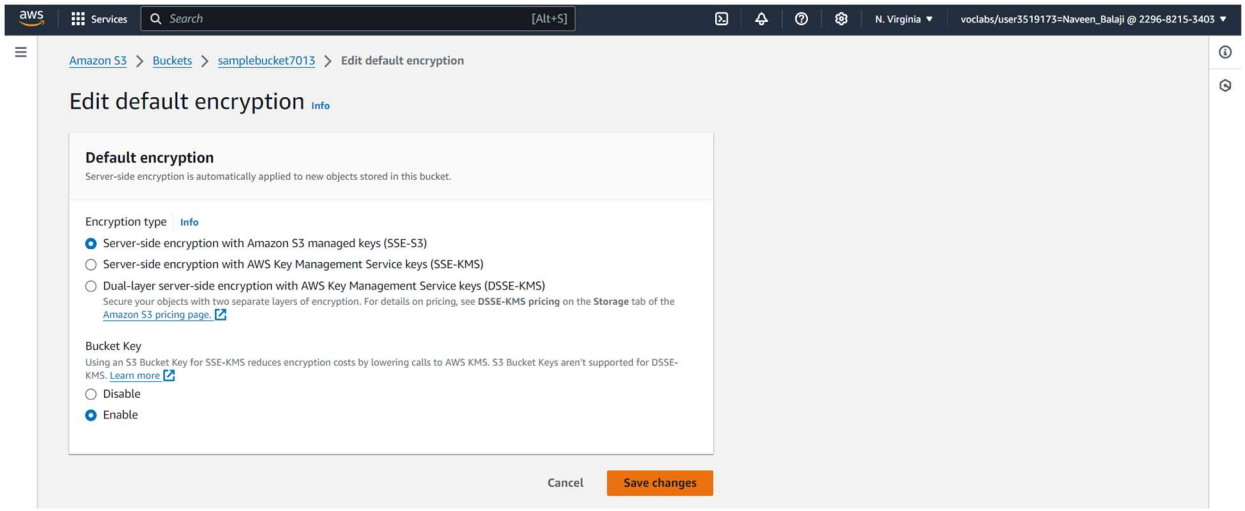
1. Test Object Upload:
    - Attempt to upload an unencrypted object to the bucket using the S3 console or

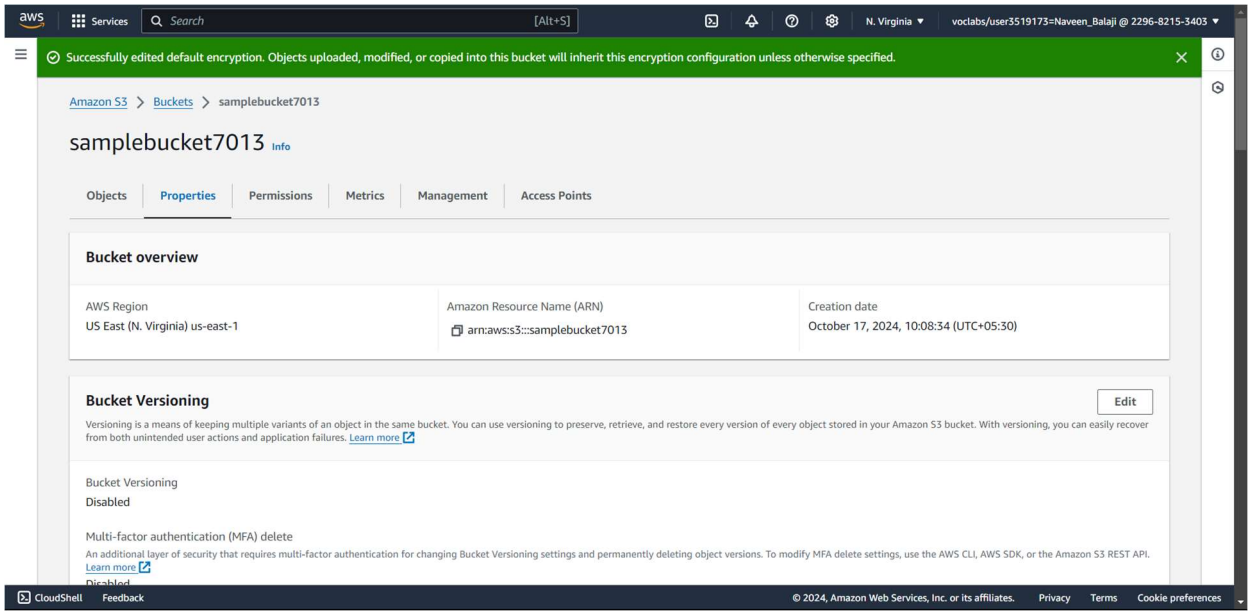
API.

- The upload should fail due to the bucket policy enforcing encryption.

2. Check Encryption Details for Valid Uploads:
   - Upload an encrypted object (via the console, CLI, or SDK) and verify that it is stored in the bucket with the proper encryption enabled.

OUTPUT:





RESULT:

The S3 bucket has been successfully configured to automatically encrypt all uploaded objects using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS Key Management Service-managed keys (SSE-KMS).

Ex.3: SET UP A CROSS-REGION REPLICATION CONFIGURATION FOR AN S3 BUCKET

AIM:

To set up cross-region replication (CRR) for an S3 bucket to automatically replicate objects from a source bucket in one AWS region to a destination bucket in a different region, ensuring data redundancy, disaster recovery, and low-latency access.

PROCEDURE:

Step 1: Create the Source and Destination Buckets

1. Create a Source Bucket:
    - Sign in to the AWS Management Console.
    - Go to "S3" and click "Create bucket."
    - Provide a unique name for the bucket (e.g., source-bucket-123).
    - Select your preferred AWS region (e.g., us-east-1).
    - Enable versioning by clicking on "Enable" in the versioning section.
    - Click "Create bucket."
2. Create a Destination Bucket:
    - Repeat the process to create a destination bucket in a different region (e.g., destination-bucket-456 in us-west-2).
    - Enable versioning for the destination bucket as well.

Step 2: Configure IAM Role for Replication

1. Create an IAM Role:
    - Go to the IAM console and click "Roles."
    - Click "Create role" and choose "S3" as the trusted entity type.
    - Choose "S3" as the service and then select "S3 Replication" use case.
    - Attach the predefined AmazonS3FullAccess policy or create a custom policy with permissions for S3 replication.
    - Complete the role creation process by providing a name for the role (e.g., S3ReplicationRole).

Step 3: Configure Cross-Region Replication (CRR)

1. Go to the Source Bucket:
    - In the S3 console, go to the source bucket (source-bucket-123).
    - Click on the "Management" tab, and under "Replication rules," click "Create replication rule."
2. Define Replication Rule:
    - Give the rule a name (e.g., CrossRegionReplicationRule).
    - Choose "Entire bucket" or specify a prefix if you want to replicate only certain objects.
3. Choose Destination Bucket:
    - Under the "Destination" section, choose the destination bucket (destination-bucket-456).
    - Select the region of the destination bucket (e.g., us-west-2).
4. Choose the IAM Role:

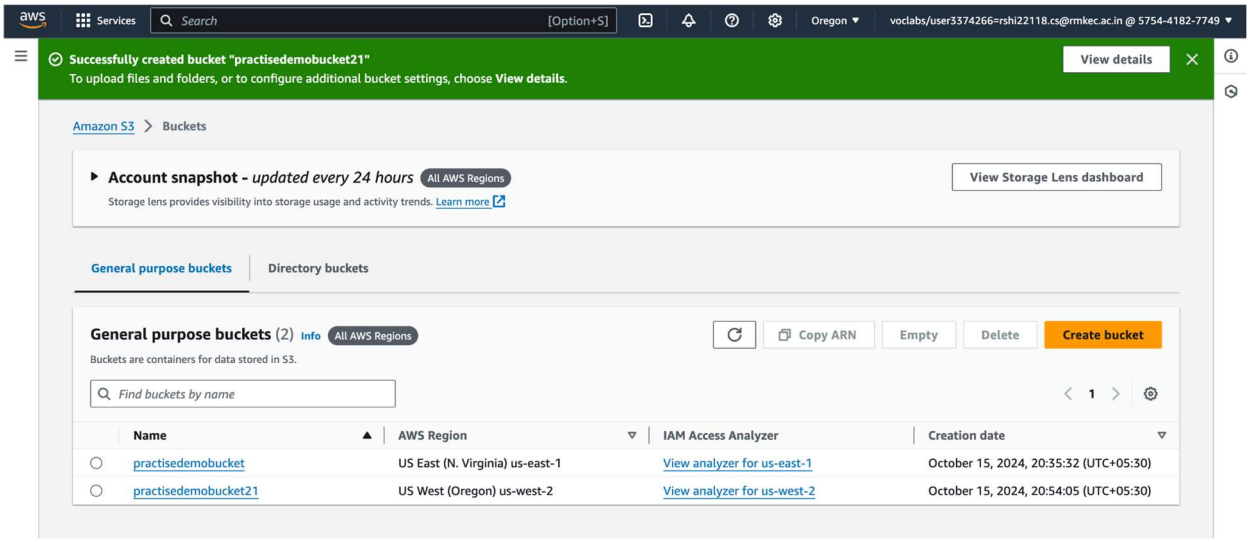
- Select the IAM role you created (S3ReplicationRole) to handle the replication process.

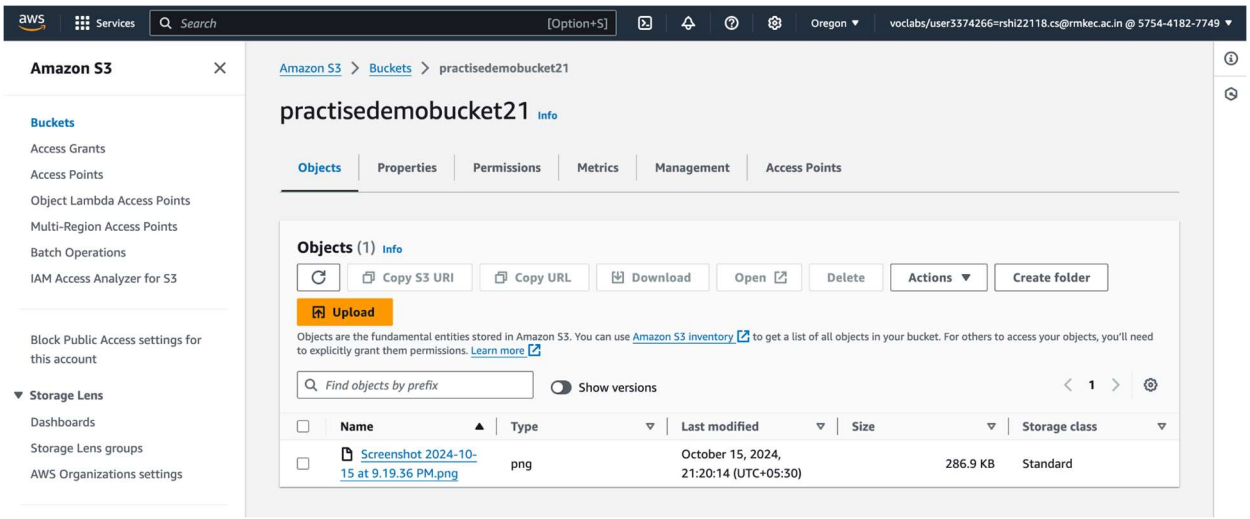
5. Save the Replication Rule:
   - Review the replication rule settings.
   - Click "Save" to apply the rule.

Step 4: Test Cross-Region Replication

1. Upload an Object to the Source Bucket:
   - Upload a test file (e.g., testfile.txt) to the source bucket (source-bucket-123).
2. Verify Replication:
   - Go to the destination bucket (destination-bucket-456), and check the "Objects" tab to confirm that the uploaded file has been automatically replicated.

OUTPUT:





RESULT:

Cross-region replication has been successfully configured between the source and destination S3 buckets. Any object uploaded to the source bucket is automatically replicated to the destination bucket in a different region.

Ex.4: CREATING A DYNAMIC WEBSITE FOR THE CAFÉ

AIM:

To create a dynamic website for a café using AWS services.

PROCEDURE:

Step 1: Set Up an EC2 Instance

1. Launch an EC2 Instance:
    - Go to the AWS Management Console.
    - Navigate to EC2 and click "Launch Instance."
    - Choose an Amazon Linux 2 AMI (or any other preferred OS).
    - Select an instance type (e.g., t2.micro for free tier eligibility).
    - Configure network settings to allow HTTP (port 80) access and assign a key pair for SSH access.
    - Launch the instance.
2. Connect to the EC2 Instance:
    - Once the instance is running, click "Connect" in the EC2 dashboard and connect via SSH or use Cloud9 for web-based access.

Step 2: Install LAMP Stack (Linux, Apache, MySQL, PHP)

1. Update the EC2 Instance:
    - In the terminal, run:
      sudo yum update -y
2. Install Apache Web Server:
    - Install Apache by running:
      sudo yum install httpd -y
    - Start the Apache service:
      sudo systemctl start httpd
      sudo systemctl enable httpd
3. Install MySQL (MariaDB):
    - Install MariaDB:
      sudo yum install mariadb-server -y
    - Start the MariaDB service:
      sudo systemctl start mariadb
      sudo systemctl enable mariadb
    - Secure the MySQL installation:
      sudo mysql_secure_installation
4. Install PHP:
    - Install PHP to support dynamic content:
      sudo yum install php php-mysql -y
    - Restart Apache to apply the PHP configuration:
      sudo systemctl restart httpd

Step 3: Configure the Dynamic Website

1. Create a Test PHP File:
    - Navigate to the Apache web root directory:

cd /var/www/html

- Create a simple PHP file:
  sudo nano index.php
- Add the following code to display a message:
  <?php
  echo "Welcome to the Café Dynamic Website!";
  ?>
- Save and exit the file.

2. Allow HTTP Traffic:
   - Ensure the EC2 instance's security group allows inbound traffic on port 80 (HTTP).
   - If not already configured, modify the security group to allow HTTP access.

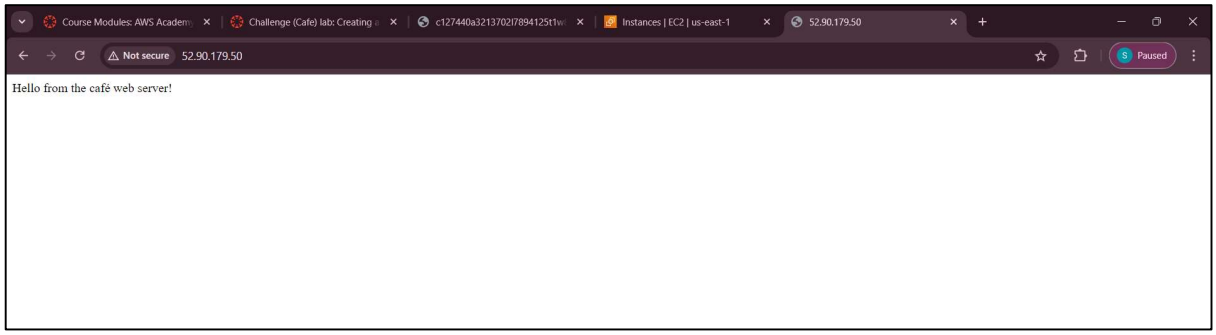
Step 4: Access the Dynamic Website

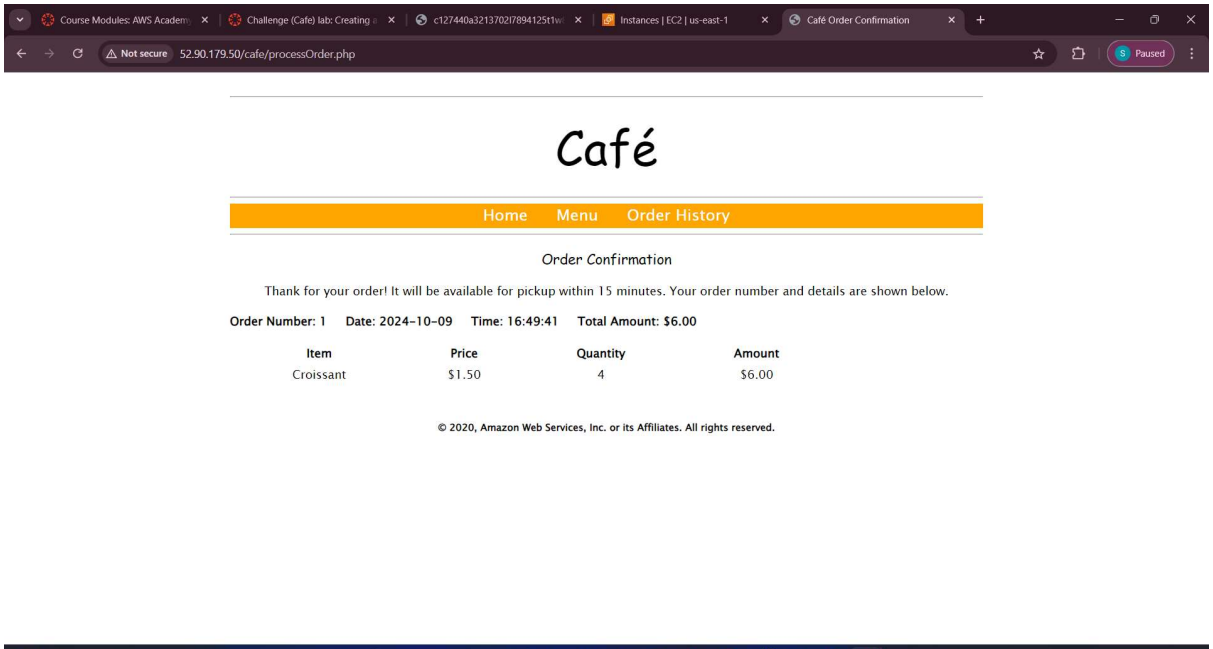
1. Get the EC2 Instance Public IP:
   - In the EC2 dashboard, find the public IP of your instance.
2. Visit the Website:
   - Open a browser and enter the public IP of your instance followed by /index.php:
     http://your-ec2-public-ip/index.php

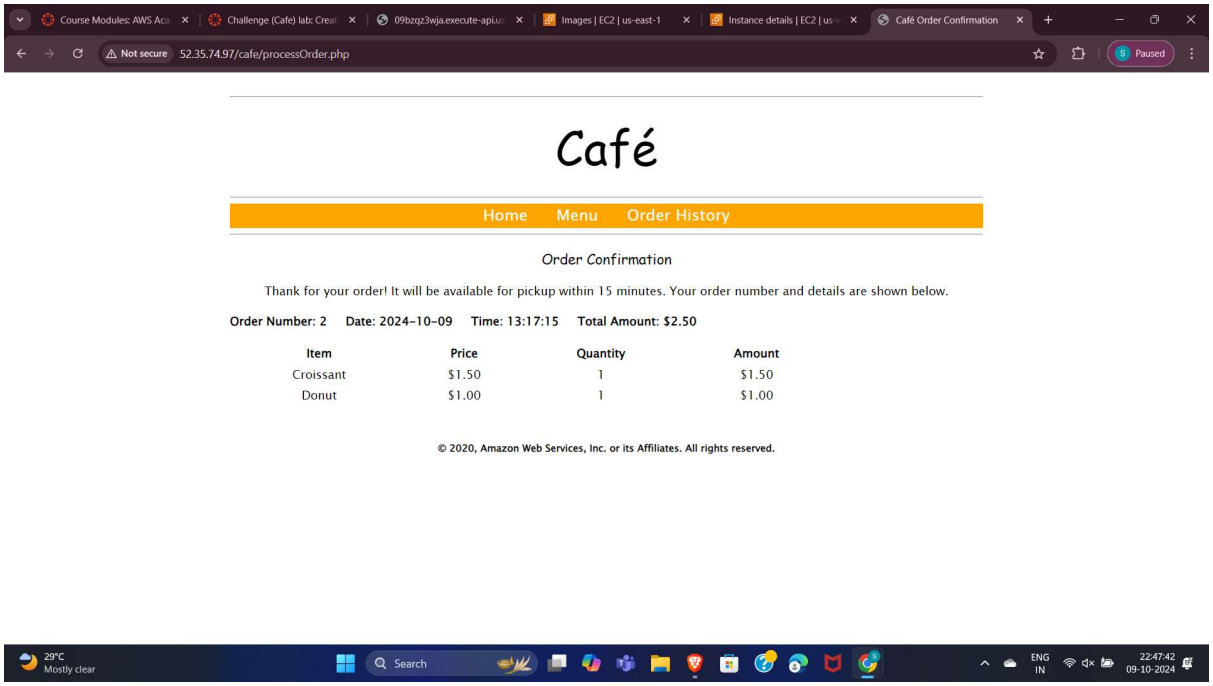
Step 5: Set Up Database (Optional)

1. Create a MySQL Database:
   - Connect to MariaDB:
     sudo mysql -u root -p
   - Create a database and user:
     CREATE DATABASE cafe_db;
     CREATE USER 'cafe_user'@'localhost' IDENTIFIED BY 'password';
     GRANT ALL PRIVILEGES ON cafe_db.* TO 'cafe_user'@'localhost';
     FLUSH PRIVILEGES;
     EXIT;
2. Update Website Configuration:
   - Update your PHP or configuration files (if using a CMS or web application) to connect to the database.

OUTPUT:







RESULT:

A dynamic website for the café has been successfully created using AWS EC2 with a LAMP stack. The website is now accessible from the internet and ready for dynamic content management.

Ex.5: **CREATING AN AMAZON RDS DATABASE**

AIM:

To create an Amazon RDS (Relational Database Service) database instance in AWS for managing a scalable, secure, and efficient relational database in the cloud.

PROCEDURE:

Step 1: Launch Amazon RDS Instance

1. Go to the AWS Management Console:
    - Navigate to the RDS service by typing “RDS” in the search bar.
2. Create a Database:
    - Click on "Create database" in the RDS dashboard.
3. Select Database Creation Method:
    - Choose "Standard create" for more control over settings.
4. Choose Database Engine:
    - Select the database engine (e.g., MySQL, PostgreSQL, MariaDB, SQL Server, etc.).
    - Select MySQL.
5. Specify DB Instance Details:
    - DB instance class: Choose a free-tier eligible instance like db.t3.micro.
    - Storage type: Select General Purpose (SSD).
    - Allocated storage: Set to 20 GB (or less for free-tier).
6. Set Database Settings:
    - DB instance identifier: Give your database a unique name like cafe-db.
    - Master username: Set a username like admin.
    - Master password: Set a secure password and confirm it.
7. Configure Connectivity:
    - VPC: Choose a VPC (default or custom).
    - Subnet group: Use the default subnet group.
    - Public access: Set to Yes if you need to connect from outside AWS; otherwise, choose No for internal access only.
    - VPC security group: Select an existing security group or create a new one to allow access on port 3306 (for MySQL).
8. Additional Configuration:
    - Enable automatic backups and set a backup retention period if needed.
    - Leave other settings as default for a basic setup.
9. Create Database:
    - Review all settings and click "Create database".

Step 2: Connect to the Amazon RDS Database

1. Find Database Endpoint:
    - After creation, go to the Databases section in RDS and find your database.
    - Copy the endpoint (this will be used to connect to the database).
2. Connect to Database:
    - Use a MySQL client like MySQL Workbench, CLI, or PHPMyAdmin.

- Use the endpoint, username, and password to connect.

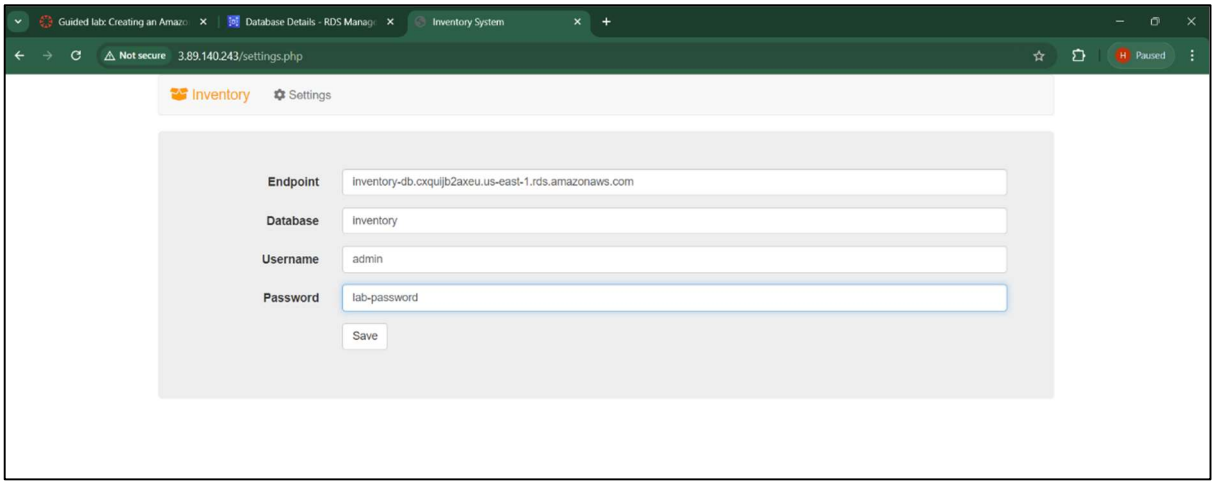
Example connection command using the CLI:

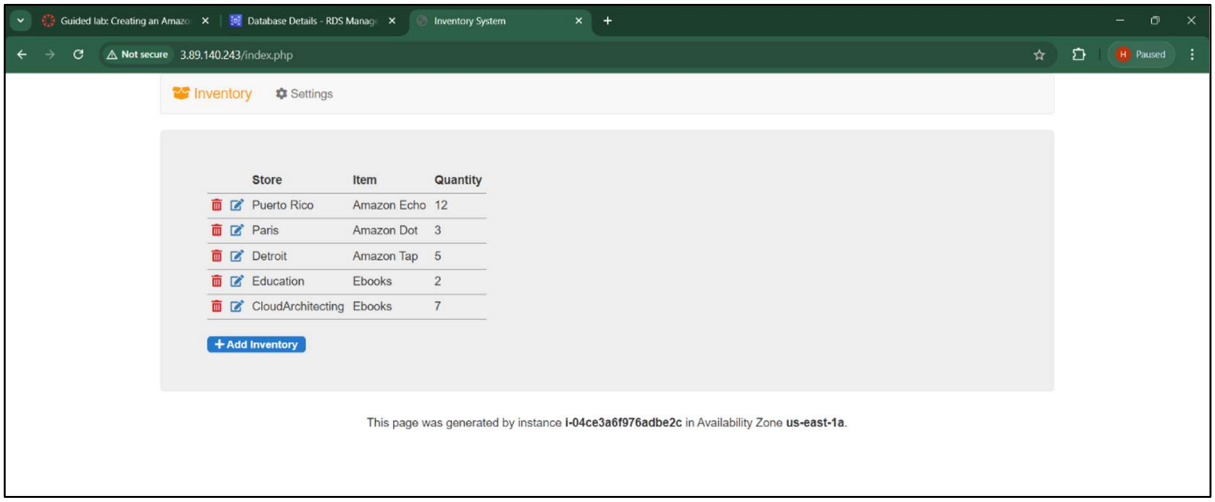
mysql -h your-db-endpoint.rds.amazonaws.com -u admin -p

Step 3: Configure Database Access

1. Modify Security Groups:
   - Ensure your EC2 instance (or your own IP) is added to the security group to allow inbound MySQL connections on port 3306.
2. Set Up Database Tables (Optional):
   - After connecting to the RDS instance, you can start creating tables, inserting data, and running queries as needed.

OUTPUT:





RESULT:

An Amazon RDS database has been successfully created and is available for use. The database can be connected to using a MySQL client or integrated with a web application hosted on AWS or other services.

Ex.6: MIGRATING A DATABASE TO AMAZON RDS

AIM:

To migrate an existing on-premise or cloud-hosted database to Amazon RDS (Relational Database Service) for better scalability, reliability, and management in the AWS cloud environment.

PROCEDURE:

Step 1: Set Up the Target Amazon RDS Database

1. **Create an RDS Instance:**
   - In the AWS Management Console, navigate to **RDS** and click **"Create database"**.
   - Choose your database engine (e.g., **MySQL**, **PostgreSQL**, **SQL Server**, etc.).
   - Select the instance type (e.g., **db.t3.micro** for a small instance).
   - Configure the instance (choose a username, password, and allocate storage).
   - Enable **public access** and ensure the security group allows access on the appropriate database port (e.g., port **3306** for MySQL).
   - Click **Create database** and wait for the instance to be ready.

Step 2: Export the Source Database

1. **Export Database Dump:**
   - Use database-specific tools to export your on-premise or cloud-hosted database. For example, if using MySQL, you can use mysqldump to create a dump file:

     mysqldump -u username -p database_name > database_dump.sql
   - This creates a file (database_dump.sql) containing all the schema and data from your database.

Step 3: Transfer the Database Dump to the RDS Instance

1. **Connect to the RDS Database:**
   - In the AWS Management Console, go to the **RDS** section and find your RDS instance.
   - Copy the **endpoint** of your RDS instance.
2. **Upload Database Dump:**
   - Use the MySQL command line (or your database's equivalent tool) to upload the dump to RDS:

     mysql -h your-rds-endpoint.rds.amazonaws.com -u admin -p database_name < database_dump.sql
   - You will be prompted to enter the password for the RDS database.

Step 4: Verify the Migration

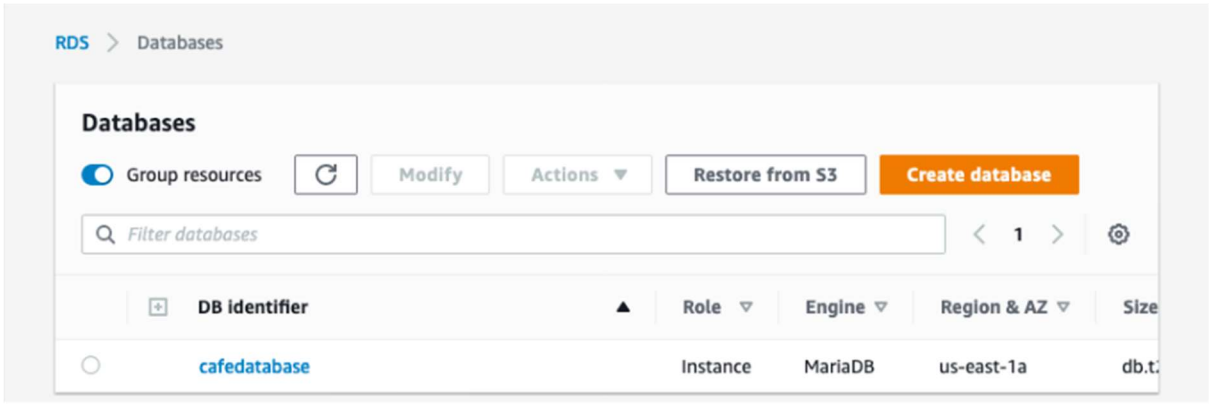
1. **Check Database Integrity:**
   - Once the database is imported, connect to the RDS instance using a MySQL client, such as **MySQL Workbench** or the command line.
   - Run queries to ensure all data has been migrated correctly:

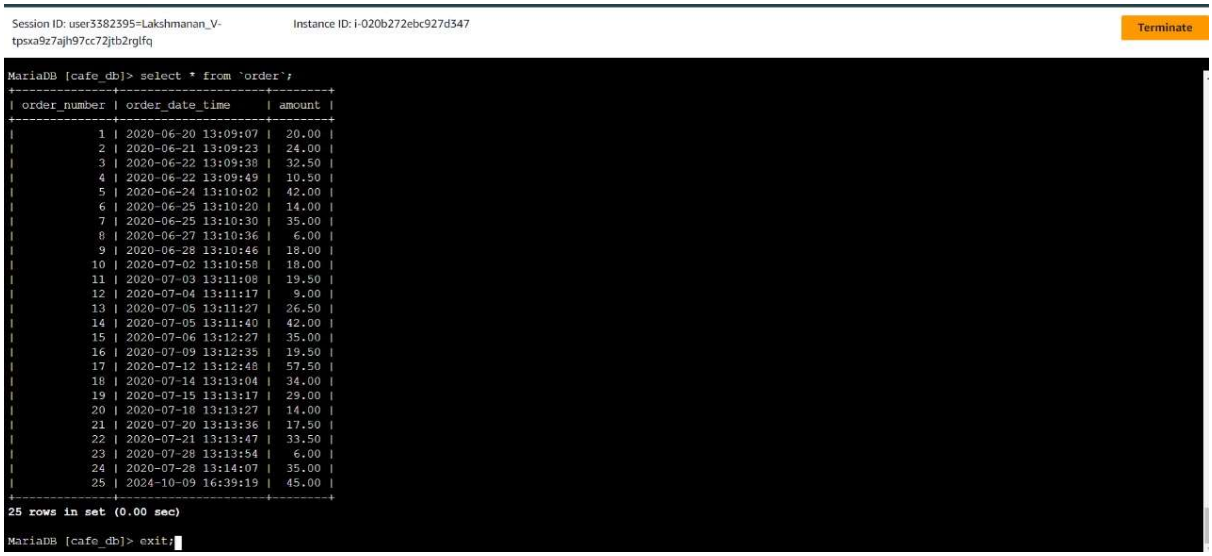
     mysql -h your-rds-endpoint.rds.amazonaws.com -u admin -p

     USE database_name;

     SHOW TABLES;
2. **Test Application Connectivity:**
   - If you are migrating a database for a web application, update the application's connection string to point to the new RDS instance.
   - Verify that the application works properly with the new RDS database.

OUTPUT:





RESULT:

The existing database has been successfully migrated to Amazon RDS. The database is now hosted on a scalable, managed platform, providing high availability and automated backups while reducing the need for manual database management.

Ex.7: CREATE A WEB APPLICATION THAT STORES DATA IN A MANAGED DATABASE USING EC2 INSTANCES AND AMAZON RDS

AIM:

To create a web application that stores user data in a managed database using Amazon EC2 instances for hosting the application and Amazon RDS for database management.

PROCEDURE:

Step 1: Set Up the Amazon RDS Database

1. Create an RDS Instance:
    - Go to the AWS Management Console and select RDS.
    - Click on "Create database".
    - Choose the database engine (e.g., MySQL, PostgreSQL, etc.).
    - Select Standard create for configuration.
    - Choose an instance class (e.g., db.t3.micro).
    - Configure settings like DB instance identifier, master username, and password.
    - Enable public access and set the security group to allow inbound connections on the appropriate port (e.g., 3306 for MySQL).
    - Click Create database and wait for the instance to become available.
2. Locate the RDS Endpoint:
    - In the left navigation pane, click on Databases to view your RDS instances.
    - Select your RDS instance.
    - In the Connectivity & security tab, find the Endpoint listed (e.g., your-db-instance.123456789012.us-east-1.rds.amazonaws.com).

Step 2: Launch an EC2 Instance

1. Create an EC2 Instance:
    - Navigate to the EC2 service in the AWS Management Console.
    - Click "Launch Instance".
    - Choose an Amazon Machine Image (AMI) (e.g., Amazon Linux 2 or Ubuntu).
    - Select an instance type (e.g., t2.micro for free-tier).
    - Configure instance settings, including security group to allow HTTP (port 80) and SSH (port 22) access.
    - Click Launch and select an existing key pair or create a new one for SSH access.
2. Locate the Key Pair:
    - In the left navigation pane, find and click on Key Pairs under the Network & Security section.
    - Ensure you have the private key file (.pem) downloaded and stored securely, as it cannot be retrieved again after creation.

Step 3: Set Up the Web Application

1. **Connect to EC2 Instance:**
    - Use an SSH client to connect to your EC2 instance:
      ssh -i "your-key-pair.pem" ec2-user@your-ec2-public-ip
2. **Install Required Packages:**
    - Update the package manager and install a web server (e.g., **Apache** or **Nginx**) and PHP or Node.js as required for your application:

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
```

3. **Deploy Application Code:**
    - Upload your web application code (HTML, CSS, JavaScript, etc.) to the EC2 instance. You can use SCP, SFTP, or any other method to transfer files.
    - Place the code in the web server's root directory (e.g., /var/www/html for Apache).

- **Upload Your Web Application Code:**
    - Create the following files on your local machine:
        - index.html
        - style.css
        - script.js
    - Sample code:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <title>Welcome to Our Café</title>
</head>
<body>
  <header>
    <h1>Welcome to Our Café</h1>
  </header>
  <main>
    <h2>Our Specialties</h2>
    <p>Enjoy the finest coffee and delicious pastries!</p>
    <button id="orderButton">Order Now</button>
  </main>
  <footer>
    <p>&copy; 2024 Café Name. All rights reserved.</p>
  </footer>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f8f8f8;
```

```
}

header {
   background-color: #4CAF50;
   color: white;
   text-align: center;
   padding: 1em 0;
}

main {
   padding: 20px;
   text-align: center;
}

footer {
   text-align: center;
   padding: 10px 0;
   background-color: #4CAF50;
   color: white;
   position: fixed;
   width: 100%;
   bottom: 0;
}
```

script.js

```
document.getElementById('orderButton').addEventListener('click', function() {
   alert('Thank you for your order!');
});
```

- **Upload the Files to the EC2 Instance:**
  - Open a terminal on your local machine and use SCP to upload:

```
scp -i "your-key-pair.pem" index.html style.css script.js ec2-user@your-ec2-public-ip:/home/ec2-user/
```

- **Move Files to the Web Server's Root Directory:**
  - Connect to the EC2 instance via SSH:

```
ssh -i "your-key-pair.pem" ec2-user@your-ec2-public-ip
```

  - Move the files to the web server's root directory:

```
sudo mv /home/ec2-user/index.html /var/www/html/
sudo mv /home/ec2-user/style.css /var/www/html/
sudo mv /home/ec2-user/script.js /var/www/html/
```

  - Set appropriate permissions:

```
sudo chown apache:apache /var/www/html/index.html
sudo chown apache:apache /var/www/html/style.css
sudo chown apache:apache /var/www/html/script.js
```

4. **Configure Database Connection:**

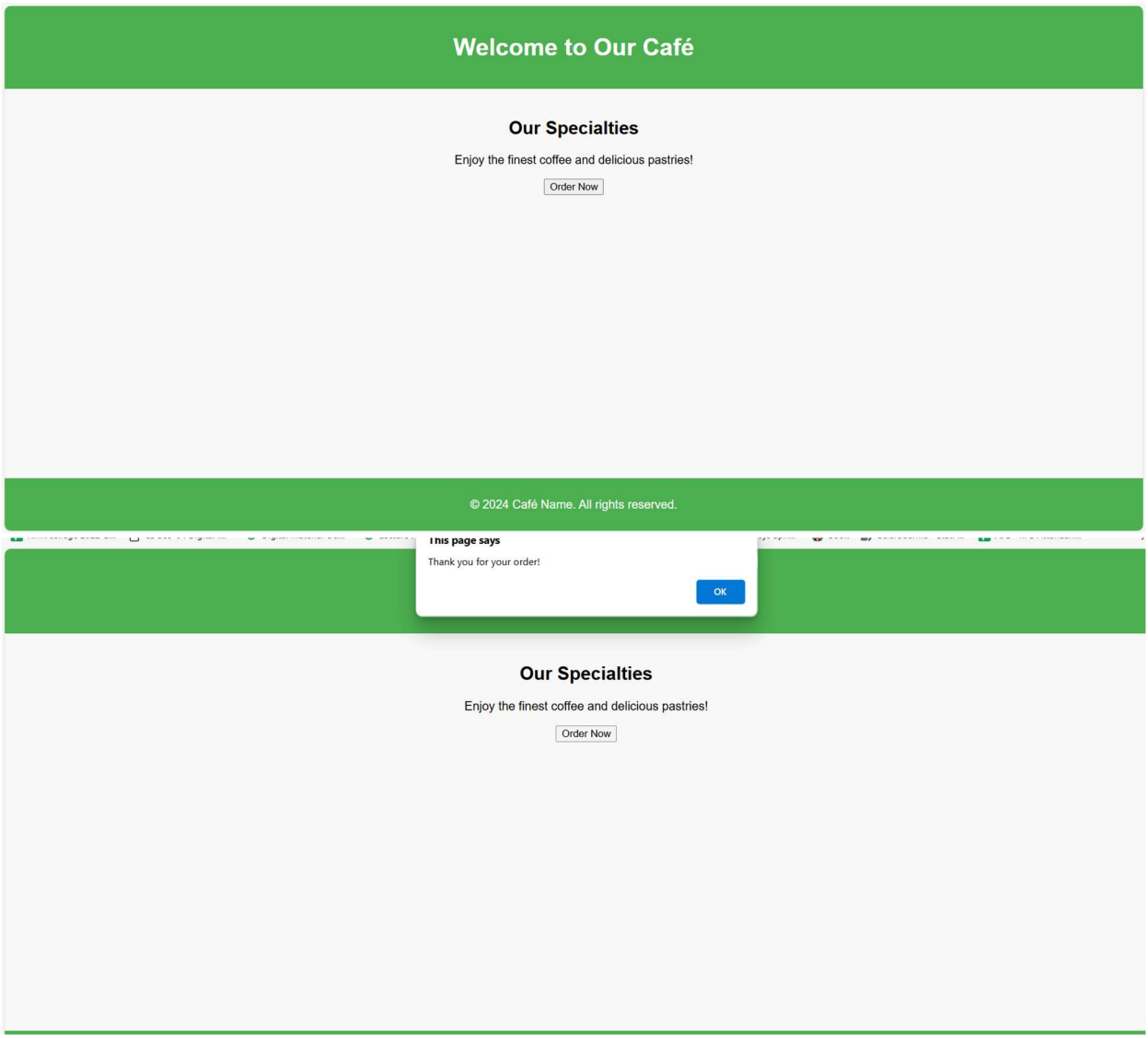
- In your application code, configure the connection string to connect to the RDS database using its endpoint, username, and password. For example, in PHP:

  $conn = new mysqli("your-rds-endpoint.rds.amazonaws.com", "admin", "yourpassword", "database_name");

Step 4: Test the Web Application

1. **Access the Web Application:**
   - Open a web browser and enter the public IP address of your EC2 instance.
   - Ensure that the web application loads correctly.
2. **Test Database Operations:**
   - Use the web application to perform database operations, such as inserting and retrieving data, to verify that it interacts with the RDS database successfully.

OUTPUT:



RESULT:

A web application is created on an EC2 instance, storing data in an Amazon RDS database. It is accessible online, providing a reliable cloud-based data management solution.

Ex.8: **CREATING A VIRTUAL PRIVATE CLOUD**

AIM:

To create a Virtual Private Cloud (VPC) in AWS to isolate resources and securely manage networking.

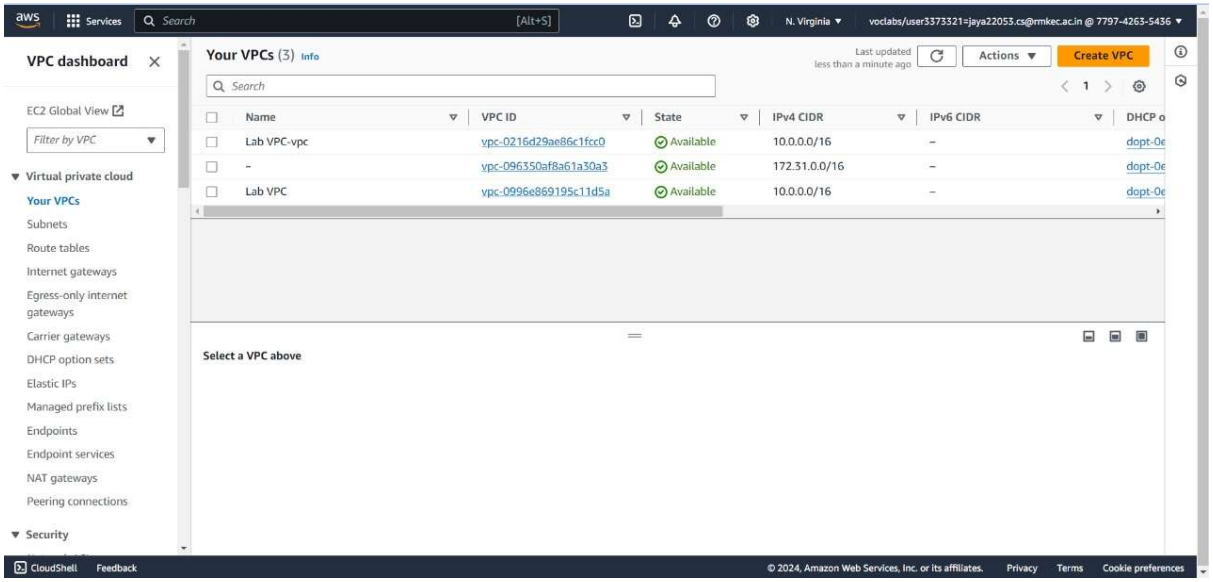
PROCEDURE:

1. **Access the VPC Dashboard:**
    - Go to the AWS Management Console.
    - In the search box, type "VPC" and select it from the options.
2. **Create a New VPC:**
    - Click on "Your VPCs" in the left menu.
    - Click the "Create VPC" button.
    - In the "Create VPC" dialog:
        - **Name tag:** Enter a name for your VPC (e.g., "MyVPC").
        - **IPv4 CIDR block:** Specify an IP range (e.g., 10.0.0.0/16).
        - **IPv6 CIDR block:** Leave this as "No IPv6 CIDR block" unless IPv6 is required.
        - **Tenancy:** Choose "Default" for shared hardware.
    - Click "Create VPC."
3. **Create Subnets:**
    - Select "Subnets" from the left menu.
    - Click the "Create subnet" button.
    - In the "Create Subnet" dialog:
        - **Name tag:** Enter a name for the subnet (e.g., "MySubnet").
        - **VPC:** Select the VPC you just created.
        - **Availability Zone:** Choose an availability zone (e.g., "us-east-1a").
        - **IPv4 CIDR block:** Specify a CIDR block (e.g., 10.0.1.0/24).
    - Click "Create subnet."
    - Repeat this step to create additional subnets if needed.
4. **Create an Internet Gateway:**
    - Click on "Internet Gateways" in the left menu.
    - Click the "Create internet gateway" button.
    - Enter a name for the internet gateway (e.g., "MyInternetGateway") and click "Create."
    - Select the newly created internet gateway and click on "Actions," then choose "Attach to VPC."
    - Select the VPC you created earlier and click "Attach."
5. **Configure Route Tables:**
    - Click on "Route Tables" in the left menu.
    - Select the route table associated with your VPC.
    - Click on the "Routes" tab, then click "Edit routes."
    - Click "Add route":
        - **Destination:** Enter 0.0.0.0/0 (for all IP addresses).
        - **Target:** Choose the internet gateway you created.

- Click "Save routes."

6. **Configure Security Groups:**
   - Click on "Security Groups" in the left menu.
   - Click the "Create security group" button.
   - Enter a name and description for the security group (e.g., "MySecurityGroup").
   - Select the VPC you created.
   - Define inbound and outbound rules to control access (e.g., allow HTTP (port 80) and SSH (port 22) traffic).
7. **Launch an EC2 Instance in Your VPC:**
   - Go to the EC2 dashboard and click on "Launch Instance."
   - Choose an Amazon Machine Image (AMI) and instance type.
   - In the "Configure Instance" section, select your VPC and subnet.
   - Assign a public IP if needed, and choose the security group you created.
   - Complete the remaining steps and launch the instance.

OUTPUT:



RESULT:

A Virtual Private Cloud (VPC) is created in AWS with isolated subnets, an internet gateway for external connectivity, and security groups to manage access.

Ex.9: CREATING A VPC NETWORKING ENVIRONMENT FOR THE CAFÉ

AIM:

To create a Virtual Private Cloud (VPC) networking environment in AWS to host the café's applications and securely manage network traffic.

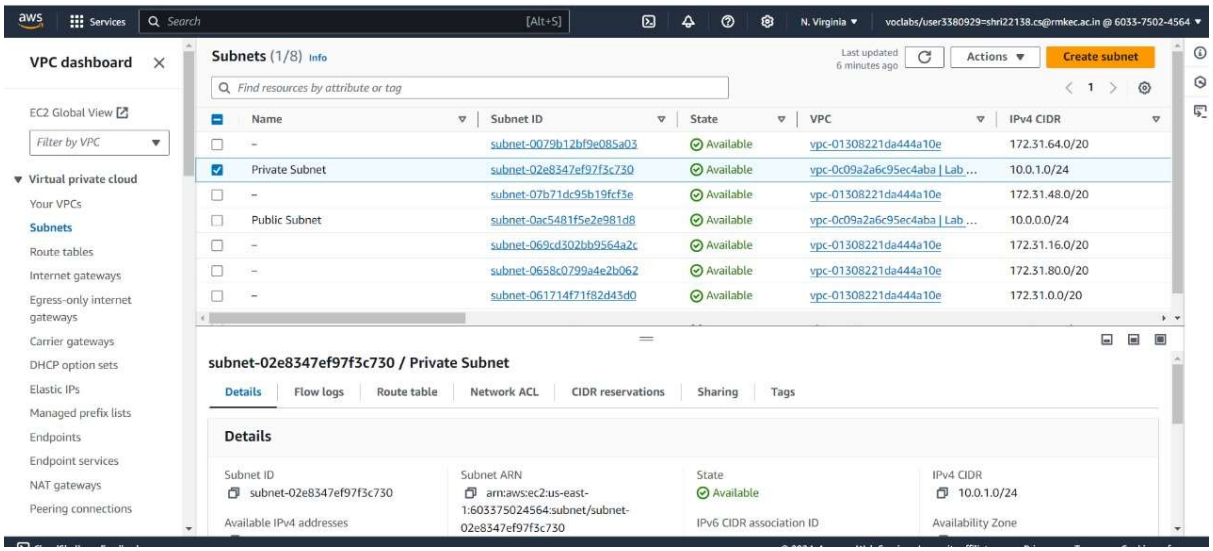
PROCEDURE:

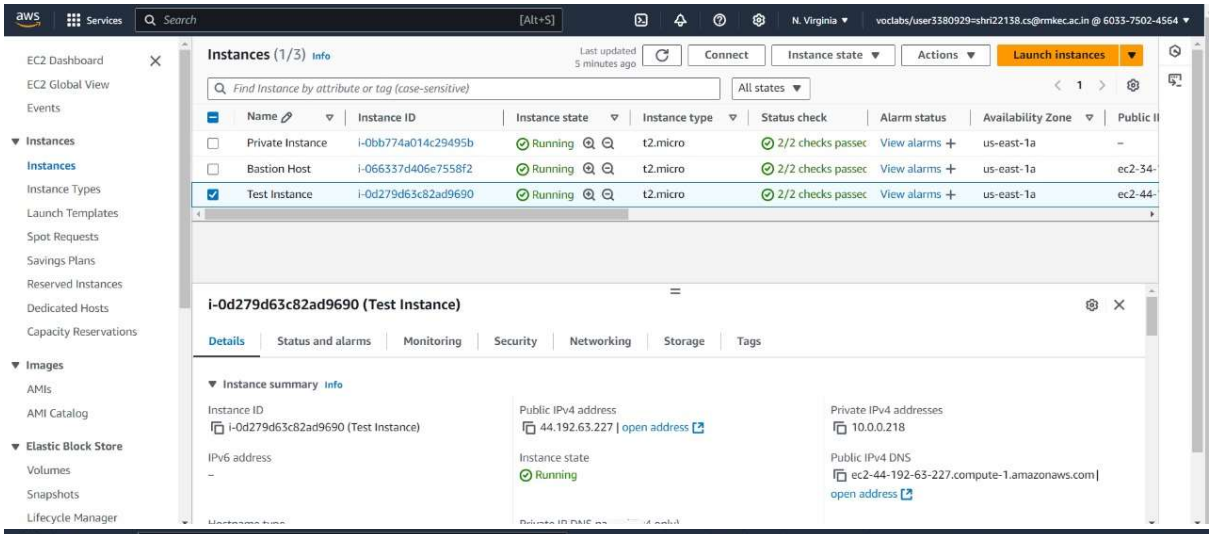
1. **Access the VPC Dashboard:**
    - Log in to the AWS Management Console.
    - In the search bar, type "VPC" and select it from the dropdown menu.
2. **Create a New VPC:**
    - Click on "Your VPCs" in the left-hand navigation pane.
    - Click the "Create VPC" button.
    - Fill out the details:
        - **Name tag:** Enter a name for your VPC (e.g., "CafeVPC").
        - **IPv4 CIDR block:** Specify an IP range (e.g., 10.0.0.0/16).
        - **Tenancy:** Choose "Default."
    - Click "Create VPC."
3. **Create Subnets:**
    - In the left navigation pane, click on "Subnets."
    - Click the "Create subnet" button.
    - Provide the details for the first subnet:
        - **Name tag:** Enter a name (e.g., "CafePublicSubnet").
        - **VPC:** Select the VPC created in the previous step.
        - **Availability Zone:** Choose an availability zone (e.g., "us-east-1a").
        - **IPv4 CIDR block:** Specify a CIDR block (e.g., 10.0.1.0/24).
    - Click "Create subnet."
    - Repeat the process to create a second subnet for private resources:
        - **Name tag:** Enter a name (e.g., "CafePrivateSubnet").
        - **IPv4 CIDR block:** Specify a CIDR block (e.g., 10.0.2.0/24).
4. **Create an Internet Gateway:**
    - Click on "Internet Gateways" in the left menu.
    - Click the "Create internet gateway" button.
    - Name the internet gateway (e.g., "CafeInternetGateway") and click "Create."
    - Select the internet gateway and click "Actions," then "Attach to VPC."
    - Select the VPC you created and click "Attach."
5. **Configure Route Tables:**
    - Click on "Route Tables" in the left navigation pane.
    - Select the route table associated with your public subnet.
    - Click the "Routes" tab, then "Edit routes."
    - Click "Add route":
        - **Destination:** Enter 0.0.0.0/0.
        - **Target:** Select the internet gateway you created.
    - Click "Save routes."
6. **Create Security Groups:**

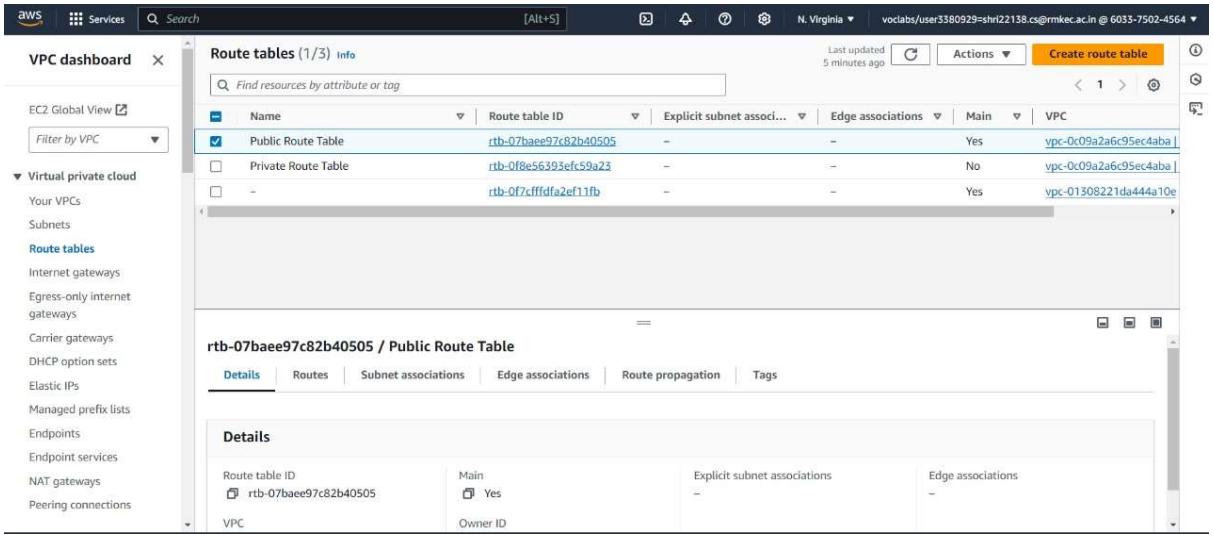
- In the left navigation pane, click on "Security Groups."
- Click "Create security group."
- Enter a name (e.g., "CafeSecurityGroup") and description.
- Choose the VPC you created.
- Add inbound rules to allow HTTP (port 80) and HTTPS (port 443) traffic from anywhere. Also, allow SSH (port 22) from your IP address for secure access.

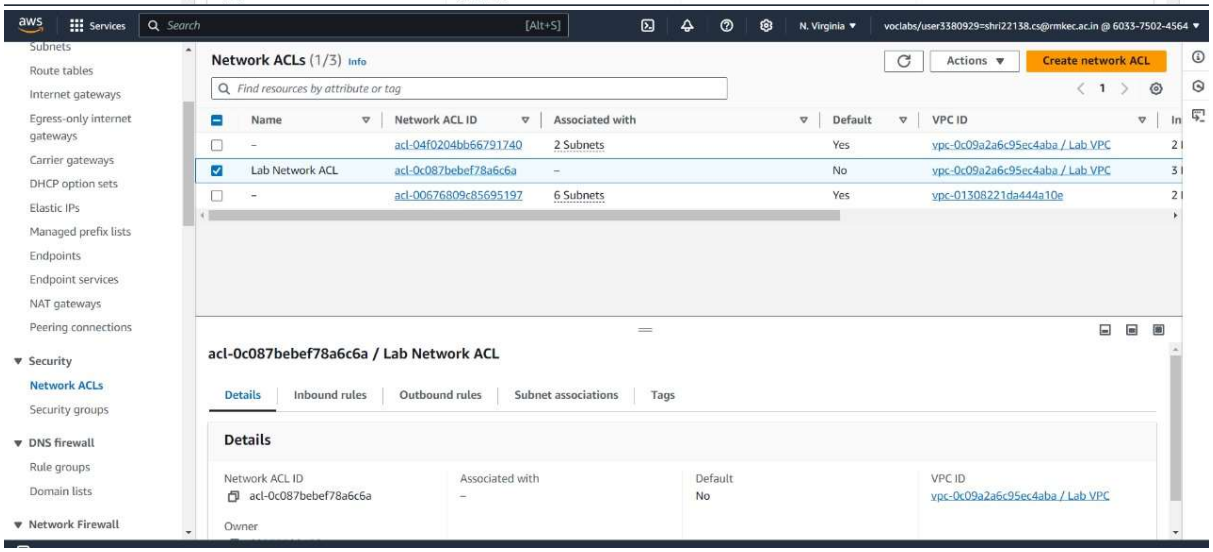
7. **Launch EC2 Instances in Your VPC:**
   - Go to the EC2 dashboard and click "Launch Instance."
   - Choose an Amazon Machine Image (AMI) suitable for your café application (e.g., Ubuntu Server).
   - Select an instance type (e.g., t2.micro).
   - In the "Configure Instance" section, choose your VPC and public subnet.
   - Enable "Auto-assign Public IP" to ensure your instance can be accessed from the internet.
   - Choose the security group created earlier.
   - Complete the remaining steps and launch the instance.
8. **Set Up RDS for Database Management (Optional):**
   - If a database is required, go to the RDS dashboard and click "Create database."
   - Choose the database engine (e.g., MySQL).
   - In the "DB Instance" settings, select the VPC you created.
   - Place the RDS instance in the private subnet to enhance security.
   - Configure security groups to allow access from your EC2 instance.

OUTPUT:









RESULT:

A VPC networking environment for the café has been established, featuring public and private subnets, an internet gateway, and security groups for traffic management.

Ex.10: **CREATING A VPC PEERING CONNECTION**

AIM:

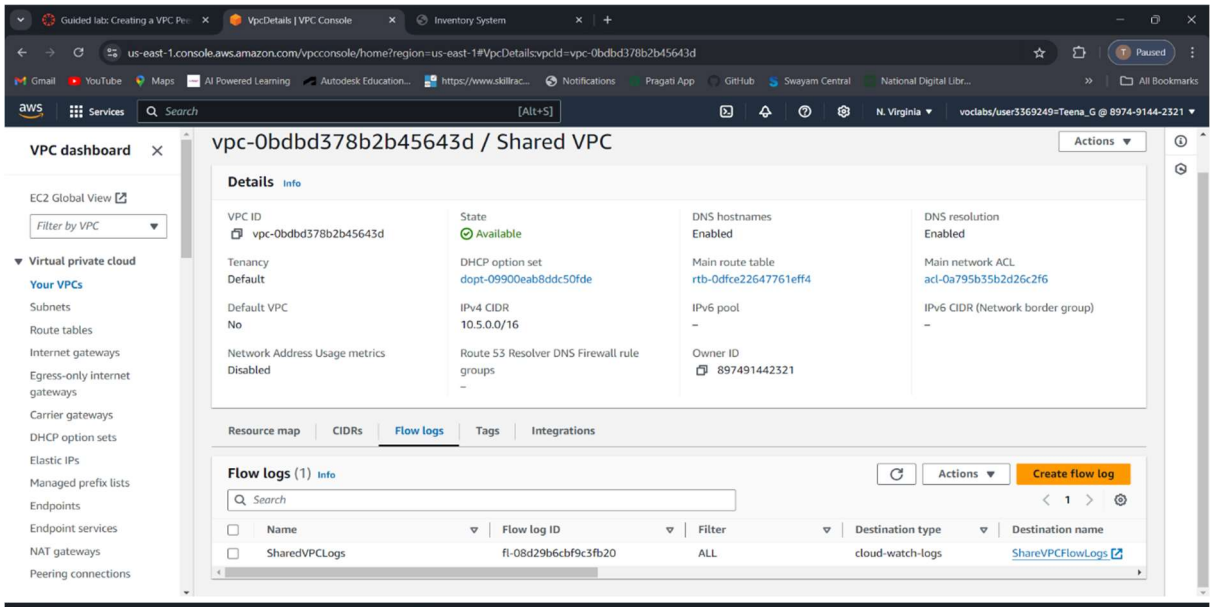
To establish a VPC peering connection between two VPCs to enable secure communication between them.

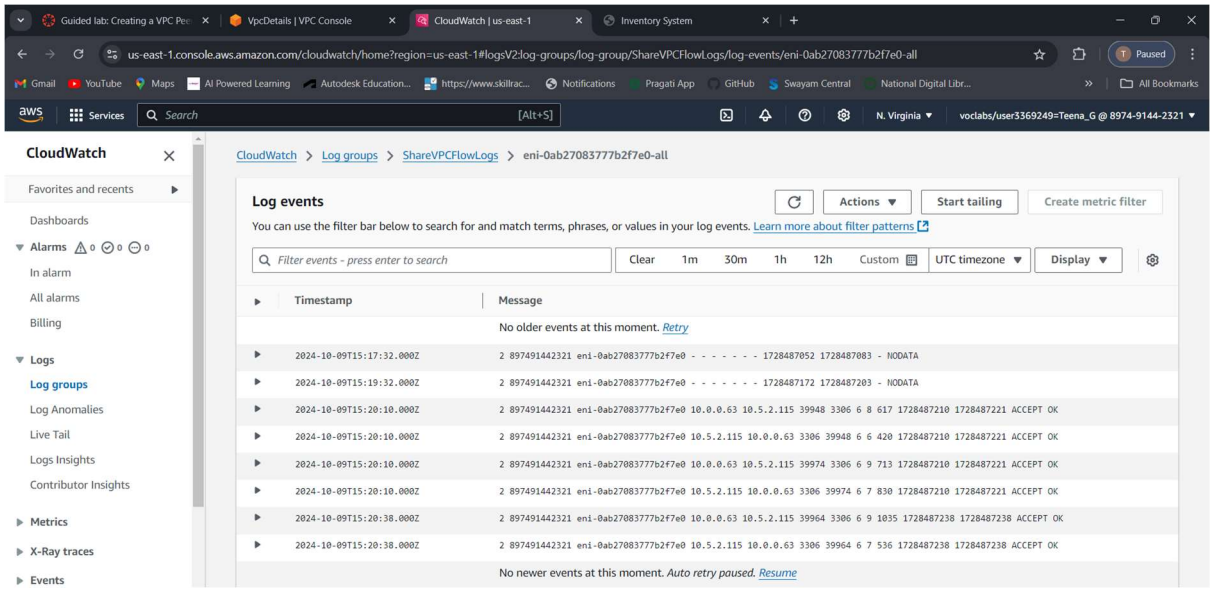
PROCEDURE:

1. **Access the VPC Dashboard:**
   - Log in to the AWS Management Console.
   - In the search bar, type "VPC" and select it from the dropdown menu.
2. **Identify VPCs for Peering:**
   - Click on "Your VPCs" in the left-hand navigation pane.
   - Note the VPC IDs of the VPCs you want to peer (e.g., "VPC A" and "VPC B").
3. **Create the VPC Peering Connection:**
   - Click on "Peering Connections" in the left menu.
   - Click the "Create Peering Connection" button.
   - Fill out the details:
     - **Peering connection name:** Enter a name (e.g., "CafeVPCPeering").
     - **VPC ID (Requester):** Select the VPC that is requesting the peering (e.g., VPC A).
     - **VPC ID (Accepter):** Select the VPC to be peered (e.g., VPC B).
   - Click "Create Peering Connection."
4. **Accept the Peering Connection:**
   - In the Peering Connections section, find the newly created peering connection.
   - Select it and click on "Actions," then choose "Accept Request."
   - Confirm the acceptance of the peering connection.
5. **Update Route Tables:**
   - Go to "Route Tables" in the left menu.
   - Select the route table associated with the first VPC (e.g., VPC A).
   - Click the "Routes" tab and then "Edit routes."
   - Click "Add route":
     - **Destination:** Enter the CIDR block of the second VPC (e.g., 10.0.2.0/24 for VPC B).
     - **Target:** Select the VPC peering connection.
   - Click "Save routes."
   - Repeat this step for the second VPC, adding a route for the first VPC's CIDR block.
6. **Update Security Groups:**
   - In the AWS Management Console, navigate to the "EC2" dashboard.
   - In the left menu, click on "Security Groups" under "Network & Security."
   - Select the security group associated with the first VPC (e.g., VPC A).
   - Click on the "Inbound rules" tab and then "Edit inbound rules."
   - Click "Add rule":
     - **Type:** Select the protocol (e.g., All traffic, HTTP, or custom).
     - **Source:** Enter the CIDR range of the peered VPC (e.g., 10.0.2.0/24 for VPC B).

- Click "Save rules."
- Repeat this step for the second VPC, adding a rule to allow traffic from the first VPC's CIDR block.

OUTPUT:





RESULT:

A VPC peering connection has been successfully established between the two VPCs, allowing secure communication between them.

Ex.11: CONFIGURE A VPC WITH SUBNETS, AN INTERNET GATEWAY, ROUTE TABLES, AND A SECURITY GROUP, AND CONNECT AN ON-PREMISES NETWORK TO THE VPC

AIM:

To create a Virtual Private Cloud (VPC) with necessary configurations and establish a VPN connection to an on-premises network.

PROCEDURE:

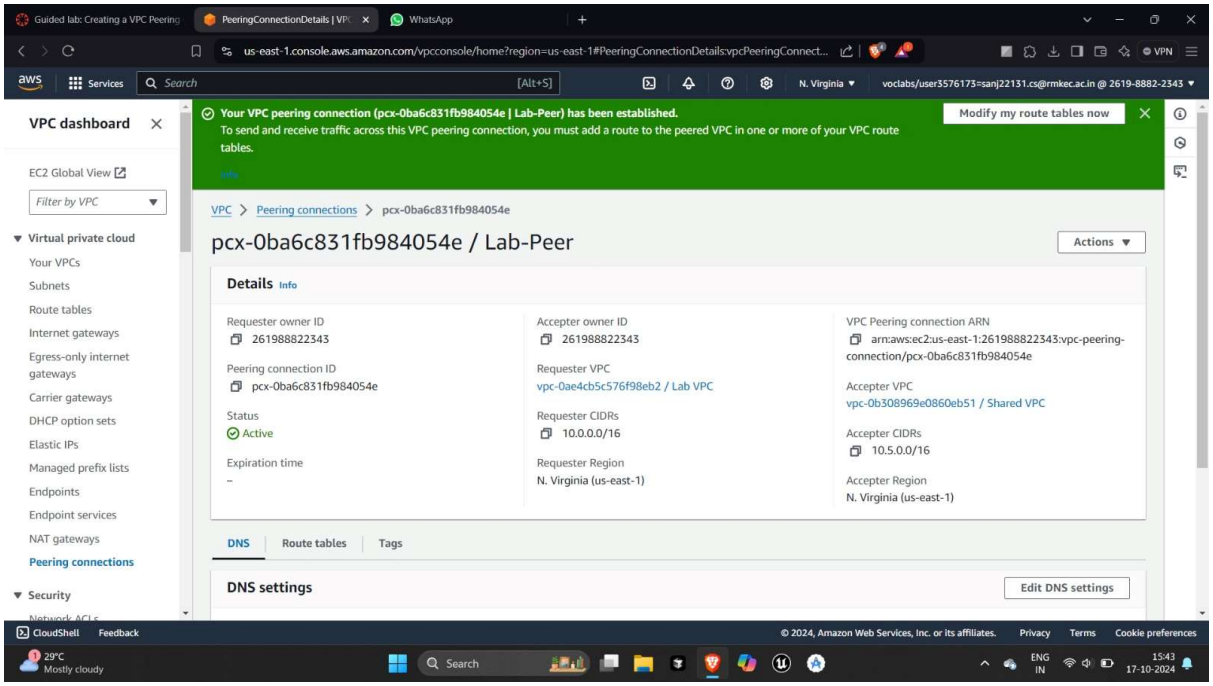
1. **Create a VPC:**
    - Go to the VPC Dashboard in the AWS Management Console.
    - Click on Your VPCs in the left-hand menu.
    - Click on Create VPC.
    - Enter the following details:
        - Name tag: Enter a name for your VPC (e.g., "CafeVPC").
        - IPv4 CIDR block: Enter a CIDR block (e.g., 10.0.0.0/16).
        - IPv6 CIDR block: (Optional) Choose an IPv6 CIDR block if needed.
        - Tenancy: Choose "Default."
    - Click Create VPC.
2. **Create Subnets:**
    - In the VPC dashboard, click on Subnets.
    - Click on Create Subnet.
    - Enter the following details for each subnet:
        - Name tag: Give a name (e.g., "PublicSubnet").
        - VPC: Select the VPC you created.
        - Availability Zone: Choose an availability zone.
        - IPv4 CIDR block: Enter a subnet CIDR block (e.g., 10.0.1.0/24).
    - Repeat to create a private subnet (e.g., "PrivateSubnet" with 10.0.2.0/24).
3. **Create an Internet Gateway:**
    - In the VPC dashboard, click on Internet Gateways.
    - Click on Create Internet Gateway.
    - Enter a name tag (e.g., "CafeInternetGateway").
    - Click Create Internet Gateway.
    - Select the Internet Gateway and click Actions -> Attach to VPC. Select your VPC and click Attach.
4. **Configure Route Tables:**
    - In the VPC dashboard, click on Route Tables.
    - Select the route table associated with your public subnet.
    - Click on the Routes tab and then Edit routes.
    - Click Add route and enter the following:
        - Destination: 0.0.0.0/0
        - Target: Select your Internet Gateway.
    - Click Save routes.
5. **Create a Security Group:**

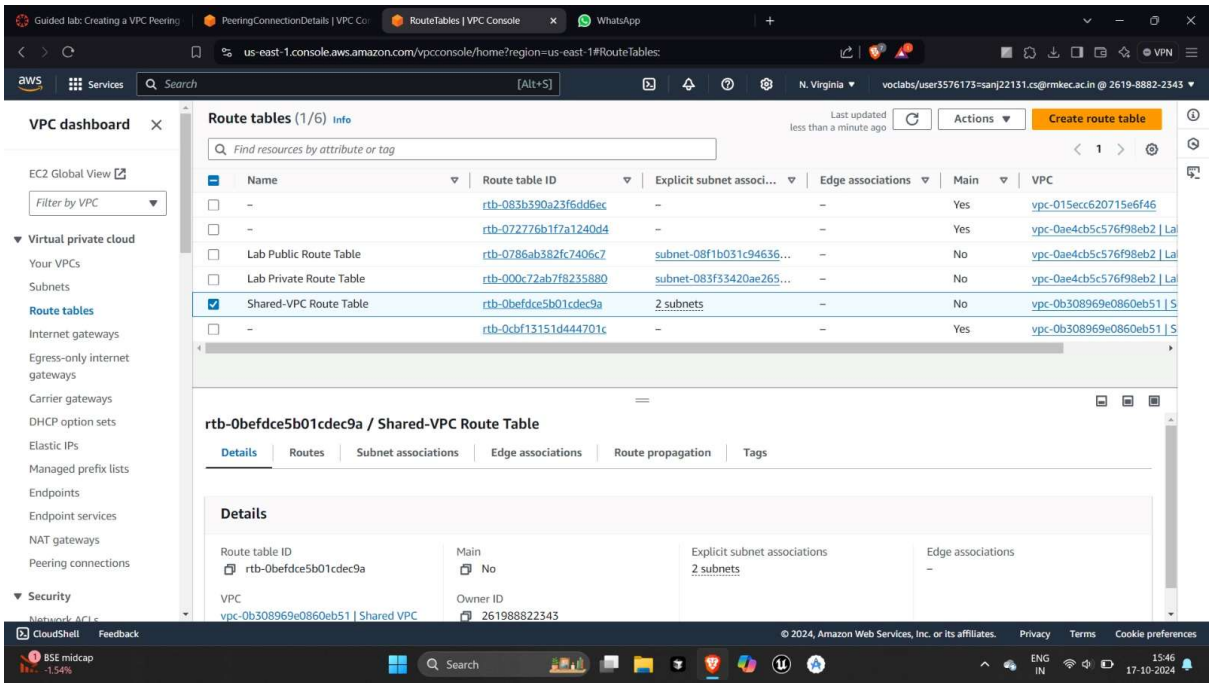
- In the VPC dashboard, click on Security Groups.
- Click on Create Security Group.
- Enter a name (e.g., "CafeSecurityGroup") and description.
- Select your VPC.
- Click Create.
- After creation, select the security group and click on the Inbound rules tab, then Edit inbound rules to add rules (e.g., allow HTTP/HTTPS, SSH).

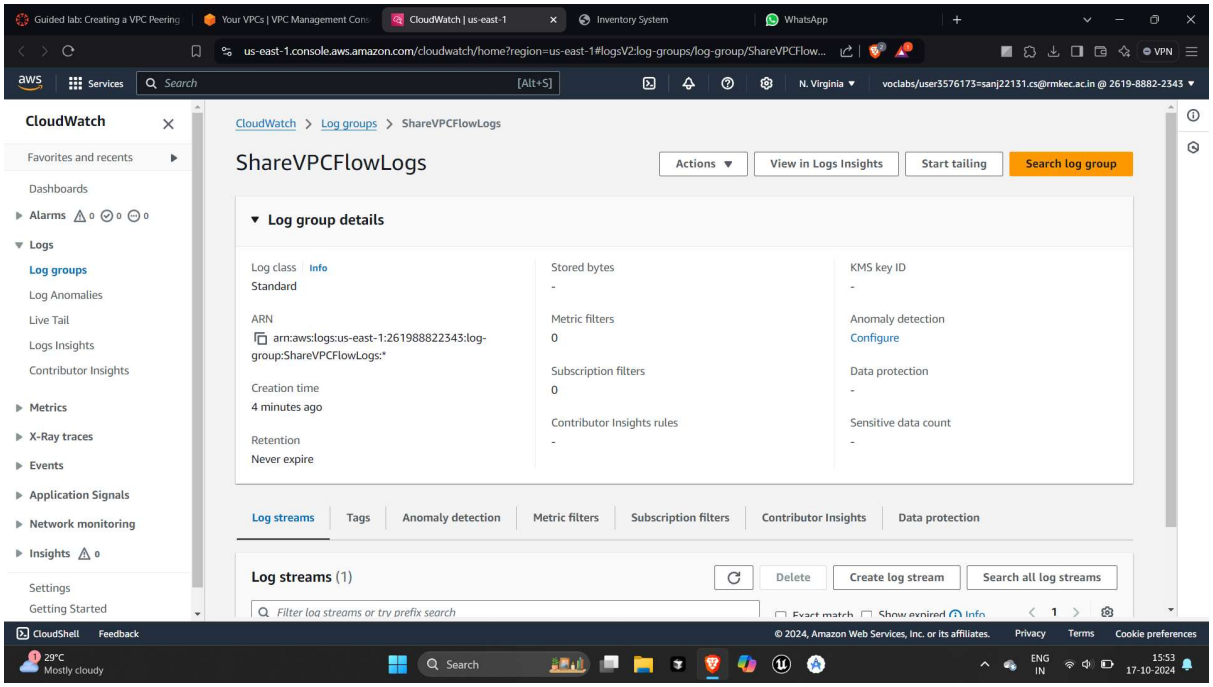
6. **Create a Virtual Private Gateway:**
   - Click on Virtual Private Gateways in the left-hand menu.
   - Click on Create Virtual Private Gateway.
   - Provide a name tag for your VPN gateway (e.g., "CafeVPNGateway").
   - Click Create Virtual Private Gateway and then attach it to your VPC.
7. **Create a Customer Gateway:**
   - In the VPC dashboard, click on Customer Gateways.
   - Click on Create Customer Gateway.
   - Provide the following details:
     - Name tag: (e.g., "OnPremisesGateway").
     - Routing: Select "Static" or "Dynamic."
     - IP Address: Enter the public IP of your on-premises router.
   - Click Create Customer Gateway.
8. **Create a VPN Connection:**
   - Click on VPN Connections in the VPC dashboard.
   - Click on Create VPN Connection.
   - Enter details for the VPN connection:
     - Name tag: (e.g., "CafeVPNConnection").
     - Target Gateway Type: Select "Virtual Private Gateway."
     - Virtual Private Gateway: Select the virtual private gateway you created.
     - Customer Gateway: Select the customer gateway.
     - Routing Options: Choose "Static" or "Dynamic" and provide any required prefixes.
   - Click Create VPN Connection.
9. **Download the VPN Configuration:**
   - After the VPN connection is created, select it from the list.
   - Click Download Configuration.
   - Choose your Vendor and Platform and click Download to save the configuration file for your on-premises device.
10. **Configure Your On-Premises Device:**
    - Open the downloaded configuration file.
    - Follow the instructions to configure your on-premises router/firewall:
      - Add VPN tunnel configurations.
      - Set up IPsec settings.
      - Define routing rules and security policies.
    - Access your router's web interface or CLI to apply the settings.
11. **Test the VPN Connection:**
    - After configuring your device, initiate the VPN connection.

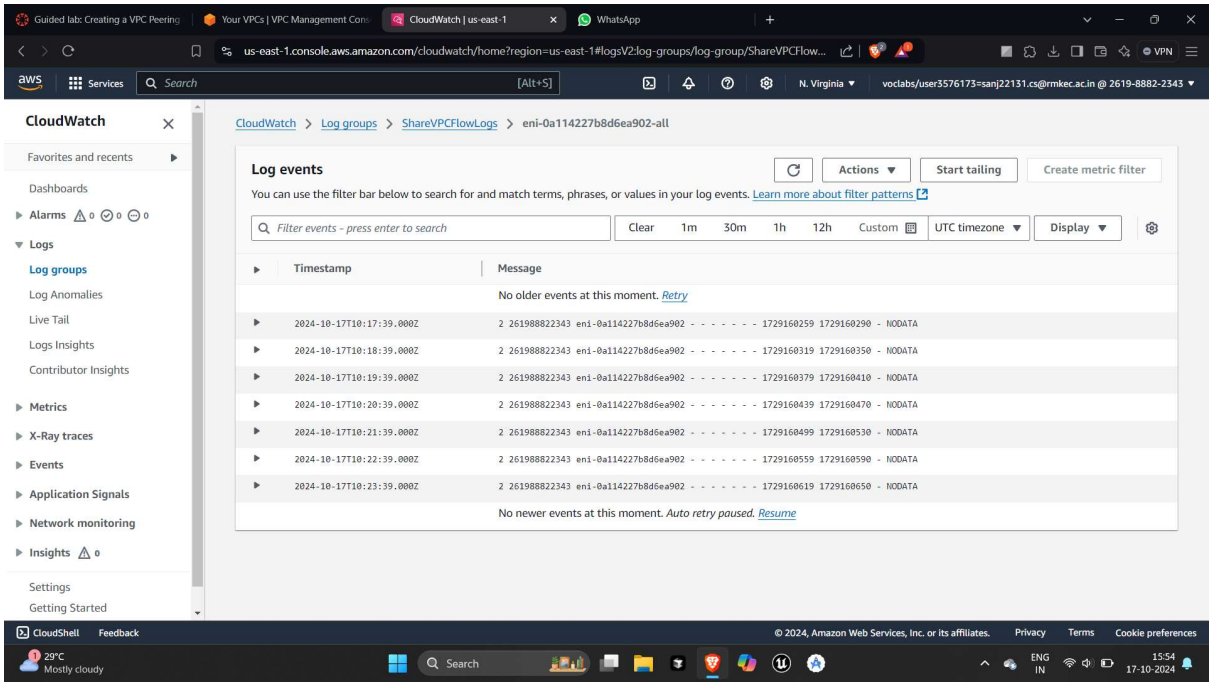
- Check the status of your VPN connection in the AWS Management Console under VPN Connections.
- Verify connectivity between your on-premises network and the VPC.

OUTPUT:









RESULT:

A VPC has been successfully configured with subnets, an Internet gateway, route tables, and a security group.

Ex.12: CONTROLLING ACCOUNT ACCESS BY USING IAM

AIM:

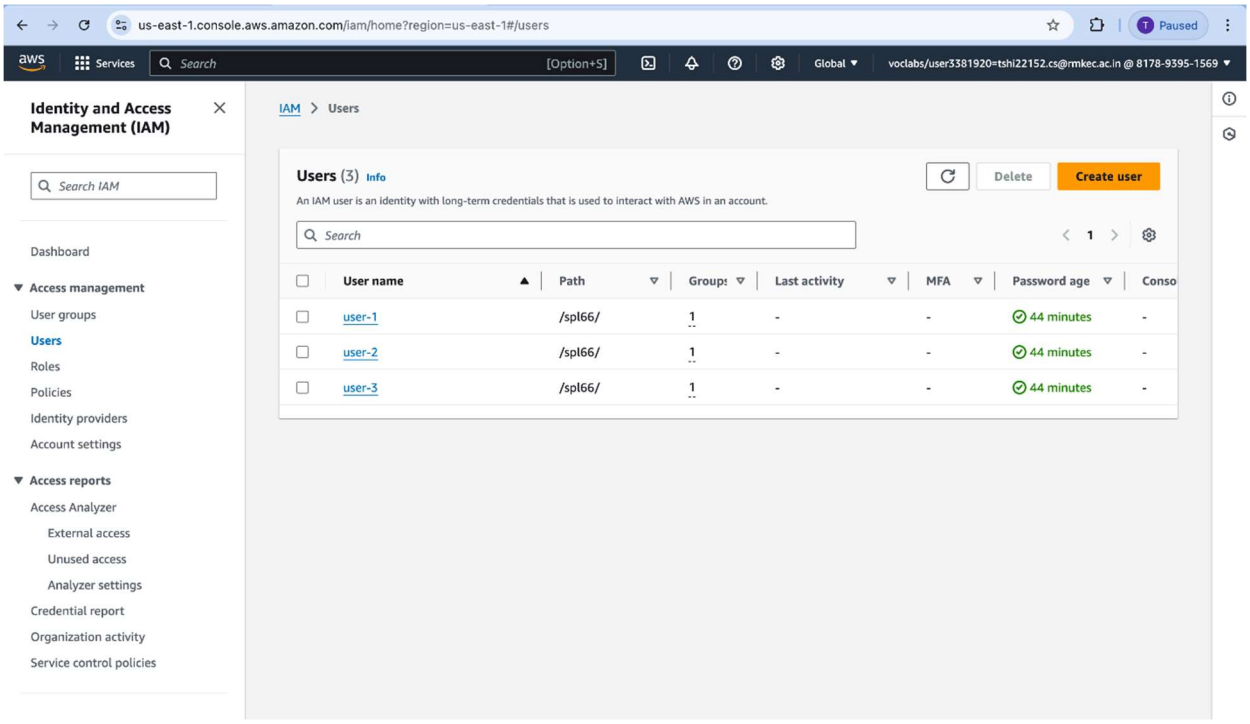
To configure Identity and Access Management (IAM) in AWS to control access to AWS resources.

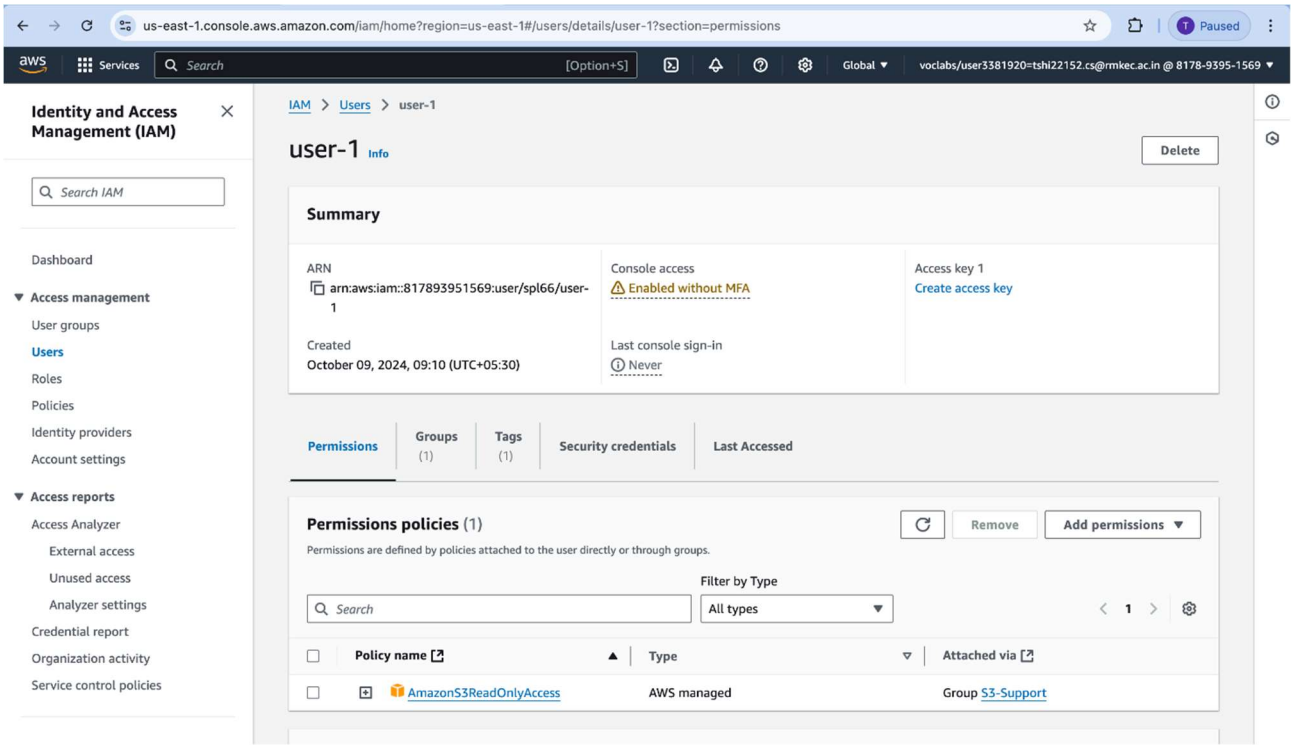
PROCEDURE:

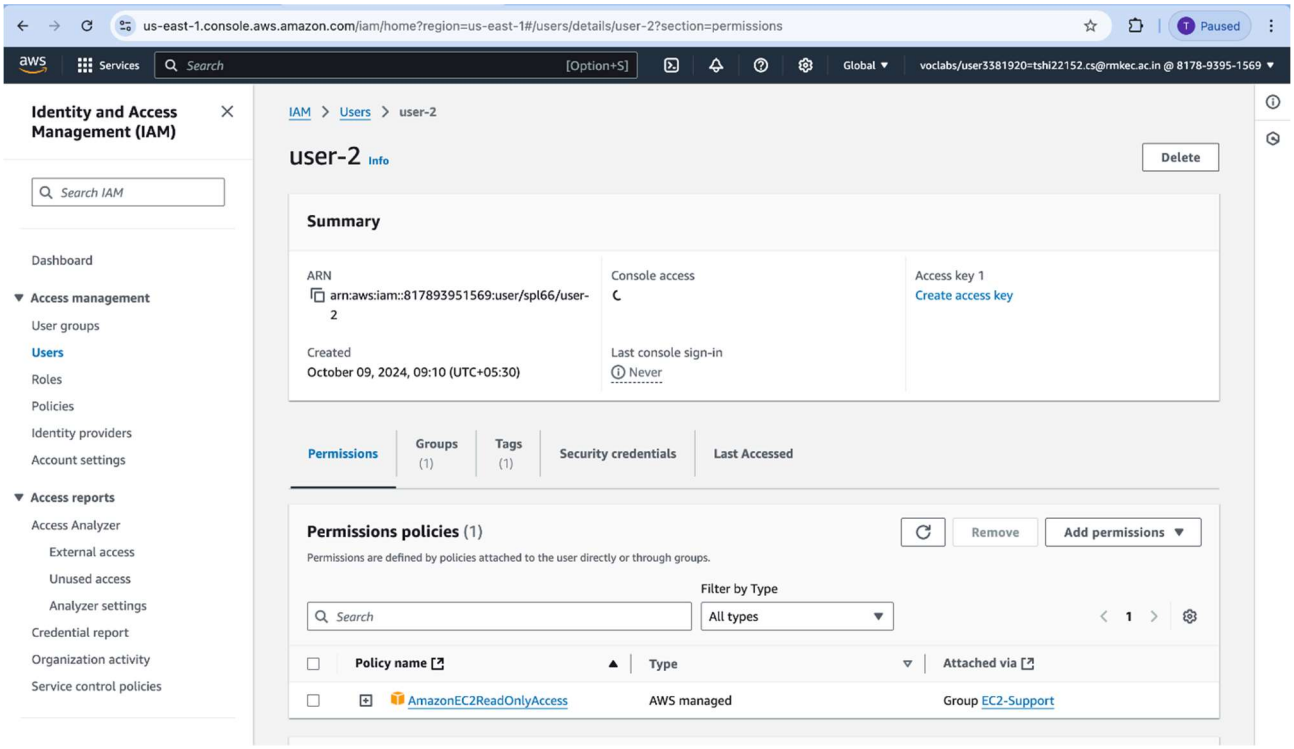
1. Log in to the AWS Management Console:
    - Open a web browser and go to the AWS Management Console.
    - Sign in with your AWS account credentials.
2. Access the IAM Dashboard:
    - In the AWS Management Console, search for IAM (Identity and Access Management) and select it.
3. Create IAM User 1: Basic User (Read-Only Access)
    - In the IAM dashboard, click on Users in the left-hand menu.
    - Click on Add user.
    - Enter a User name (e.g., "BasicUser").
    - Select AWS Management Console access and set a custom password.
    - Click Next: Permissions.
    - Select Attach existing policies directly and search for the ReadOnlyAccess policy.
    - Click on Next: Tags (optional) and then Next: Review.
    - Review the details and click Create user.
    - Take note of the login credentials.
4. Create IAM User 2: Developer User (Write Access to S3)
    - Click on Add user again.
    - Enter a User name (e.g., "DevUser").
    - Select AWS Management Console access and set a custom password.
    - Click Next: Permissions.
    - Select Attach existing policies directly and search for the AmazonS3FullAccess policy.
    - Click on Next: Tags (optional) and then Next: Review.
    - Review the details and click Create user.
    - Note the login credentials.
5. Create IAM User 3: Admin User (Full Access)
    - Click on Add user again.
    - Enter a User name (e.g., "AdminUser").
    - Select AWS Management Console access and set a custom password.
    - Click Next: Permissions.
    - Select Attach existing policies directly and search for the AdministratorAccess policy.
    - Click on Next: Tags (optional) and then Next: Review.
    - Review the details and click Create user.
    - Note the login credentials.
6. Verify User Permissions:

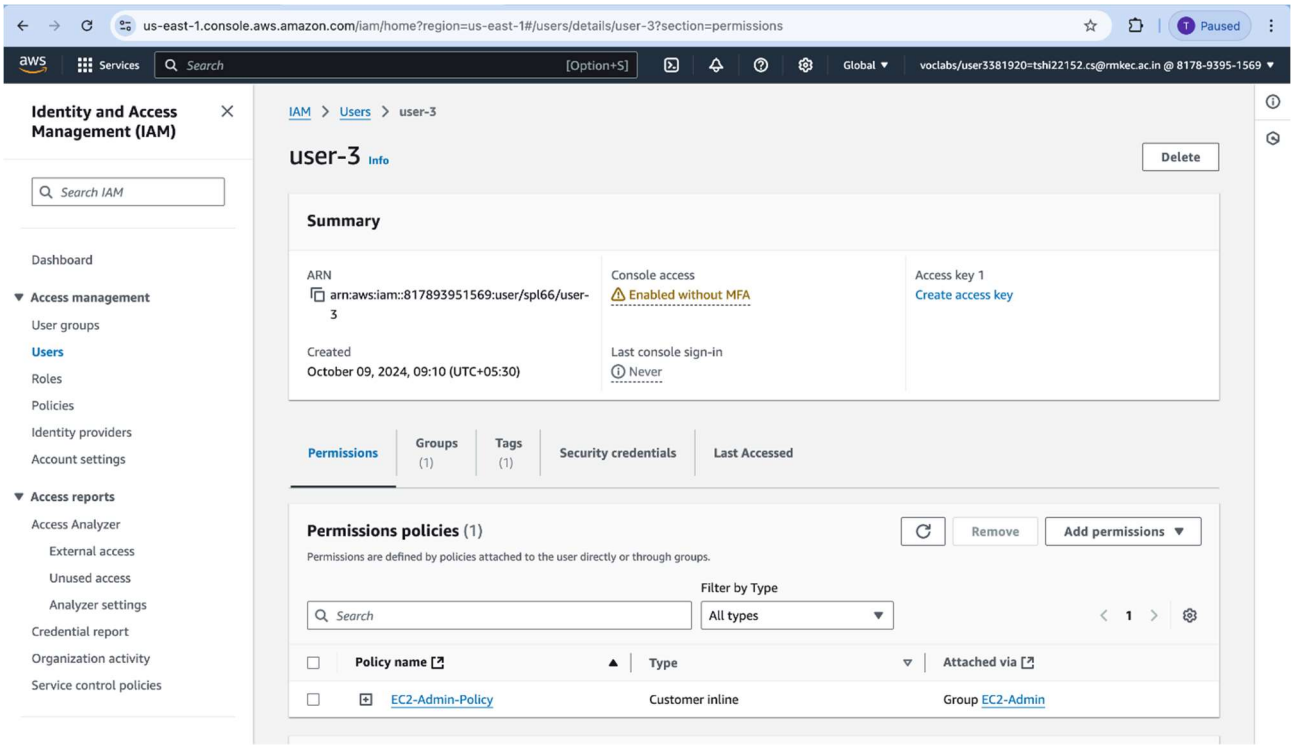
- Log in as BasicUser:
  - Use the BasicUser credentials to log in to the AWS Management Console.
  - Navigate to the S3 service.
  - Attempt to create a new S3 bucket. The user should receive an error message indicating insufficient permissions since this user has read-only access.
- Log in as DevUser:
  - Use the DevUser credentials to log in to the AWS Management Console.
  - Navigate to the S3 service.
  - Attempt to create a new S3 bucket. The user should successfully create the bucket, confirming they have write access.
- Log in as AdminUser:
  - Use the AdminUser credentials to log in to the AWS Management Console.
  - Navigate to any service (e.g., EC2, RDS, S3).
  - Attempt to create a new EC2 instance or delete an S3 bucket. The user should successfully perform any actions, confirming they have full access.

OUTPUT:




us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/user-1?section=permissions
Paused
Services
Search
[Option+S]
Global
voclabs/user3381920=tshi22152.cs@rmkec.ac.in @ 8178-9395-1569
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
External access
Unused access
Analyzer settings
Credential report
Organization activity
Service control policies
IAM > Users > user-1
user-1 Info
Delete
Summary
ARN
arn:aws:iam::817893951569:user/spl66/user-1
Console access
Enabled without MFA
Access key 1
Create access key
Created
October 09, 2024, 09:10 (UTC+05:30)
Last console sign-in
Never
Permissions
Groups (1)
Tags (1)
Security credentials
Last Accessed
Permissions policies (1)
Permissions are defined by policies attached to the user directly or through groups.
Remove
Add permissions
Filter by Type
Search
All types
1
Policy name
Type
Attached via
AmazonS3ReadOnlyAccess
AWS managed
Group S3-Support


us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/user-2?section=permissions
Paused
Services
Search
[Option+S]
Global
voclabs/user3381920=tshi22152.cs@rmkec.ac.in @ 8178-9395-1569
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
External access
Unused access
Analyzer settings
Credential report
Organization activity
Service control policies
IAM > Users > user-2
user-2 Info
Delete
Summary
ARN
arn:aws:iam::817893951569:user/spl66/user-2
Console access
Access key 1
Create access key
Created
October 09, 2024, 09:10 (UTC+05:30)
Last console sign-in
Never
Permissions
Groups (1)
Tags (1)
Security credentials
Last Accessed
Permissions policies (1)
Permissions are defined by policies attached to the user directly or through groups.
Remove
Add permissions
Filter by Type
Search
All types
1
Policy name
Type
Attached via
AmazonEC2ReadOnlyAccess
AWS managed
Group EC2-Support



RESULT:

Three IAM users were successfully created with different permissions.

Ex.13: CREATING SCALING POLICIES FOR AMAZON EC2 AUTO SCALING

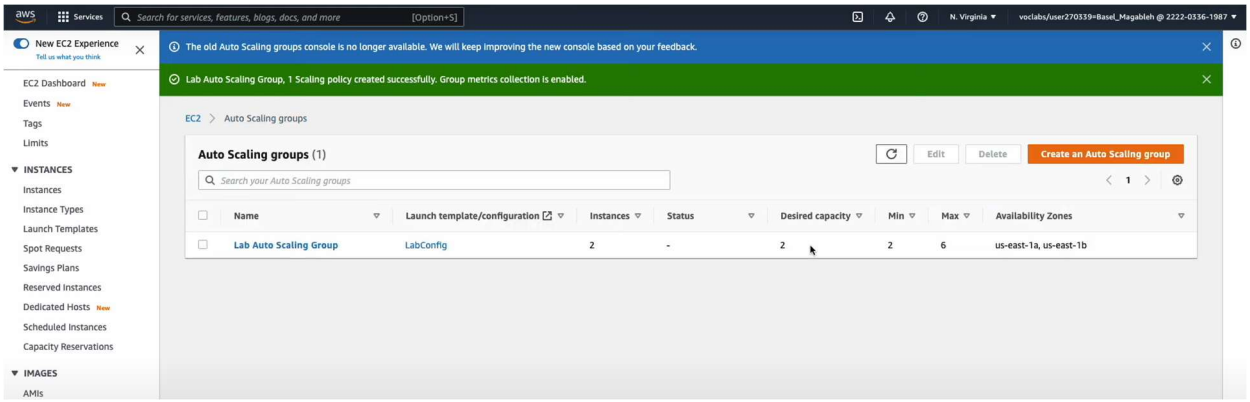
AIM:

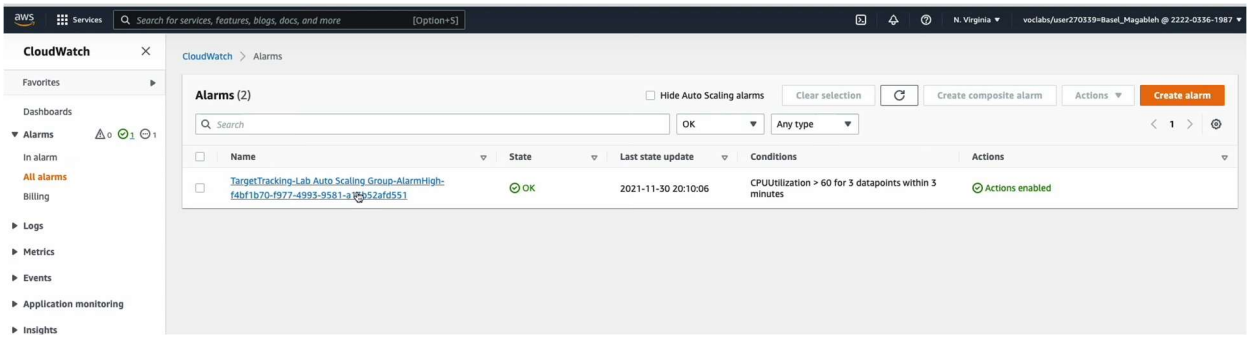
To create and configure scaling policies for Amazon EC2 Auto Scaling to automatically adjust the number of EC2 instances based on demand.

PROCEDURE:

1. Log in to the AWS Management Console:
    - Open a web browser and go to the AWS Management Console.
    - Sign in with your AWS account credentials.
2. Access the EC2 Dashboard:
    - In the AWS Management Console, search for EC2 and select it.
3. Create an Auto Scaling Group:
    - In the EC2 dashboard, click on Auto Scaling Groups in the left-hand menu.
    - Click on Create Auto Scaling group.
    - Provide a name for the Auto Scaling group (e.g., "MyAutoScalingGroup").
    - Choose an existing Launch Template or Launch Configuration or create a new one to define the EC2 instance settings.
    - Click Next.
4. Configure Auto Scaling Group Settings:
    - Select the VPC and Subnets where the instances will run.
    - Set the Desired capacity, Minimum size, and Maximum size for the Auto Scaling group.
    - Click Next.
5. Configure Scaling Policies:
    - Under Set scaling policies, choose either:
        - Target tracking scaling policy: Automatically adjusts the capacity to maintain a specific metric.
            - For example, set the target to maintain an average CPU utilization of 50%.
        - Step scaling policy: Define how many instances to add or remove based on CloudWatch alarms.
    - Click Create after configuring the scaling policy settings.
6. Add Notifications (Optional):
    - Under Configure notifications, you can add SNS topics to receive notifications about scaling events.
    - Click Next.
7. Review and Create:
    - Review the configuration for the Auto Scaling group and the scaling policies.
    - Click Create Auto Scaling group.
8. Monitor Auto Scaling Activity:
    - In the Auto Scaling Groups dashboard, you can monitor the performance and scaling activities of your Auto Scaling group.
    - Check the Activity History tab to view scaling events.

OUTPUT:





RESULT:

An Auto Scaling group was successfully created with configured scaling policies.

Ex.14: CREATING A HIGHLY AVAILABLE WEB APPLICATION

AIM:

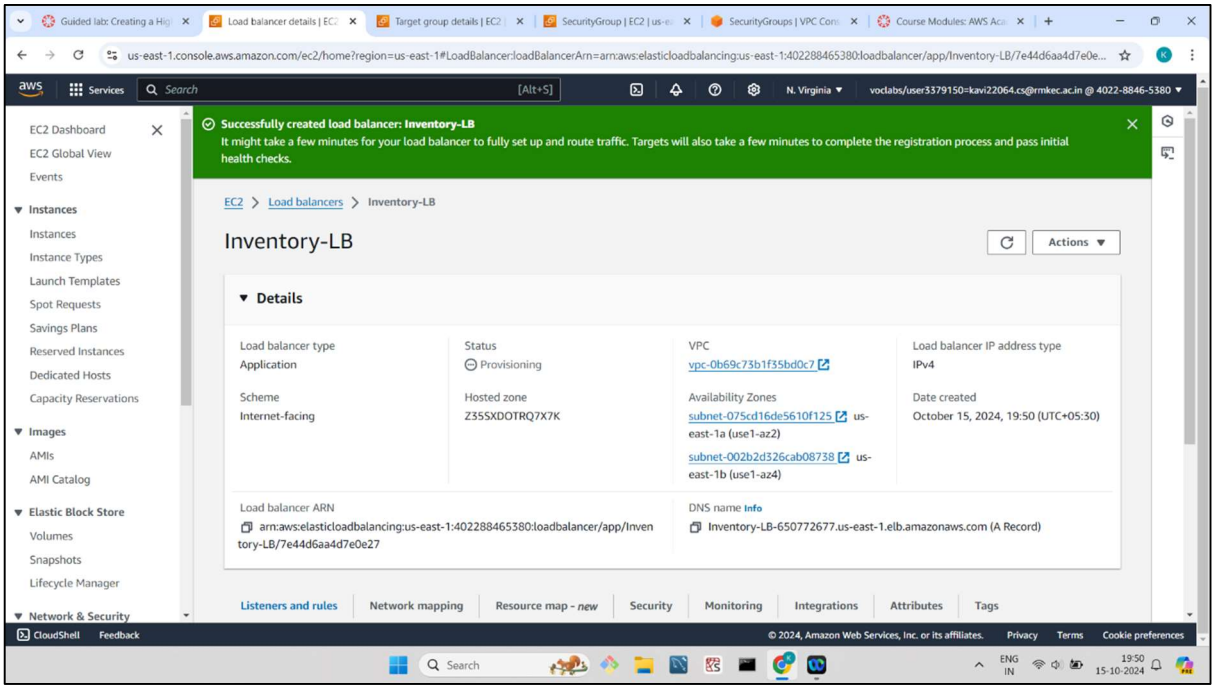
To create a highly available web application using Amazon Web Services (AWS) that ensures minimal downtime and provides seamless access to users.

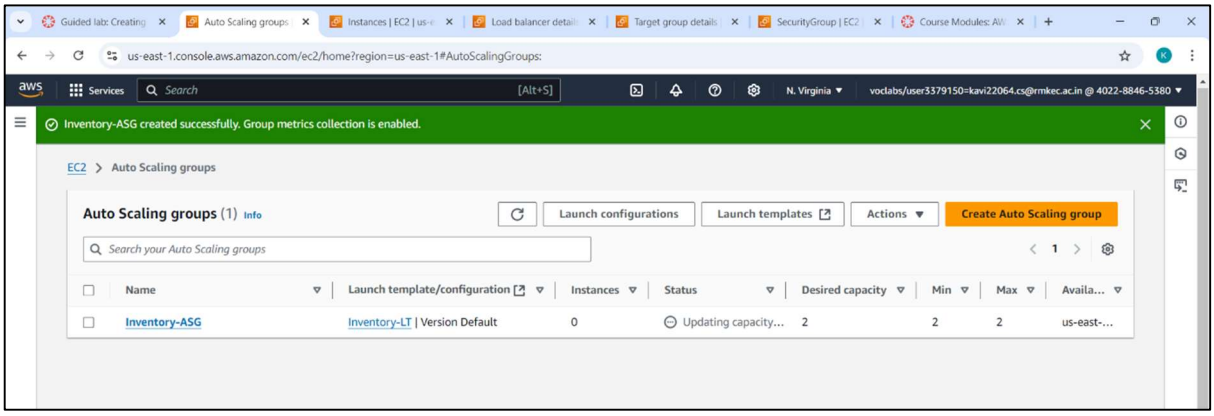
PROCEDURE:

1. **Set Up a VPC:**
    - Log in to AWS Management Console:
        - Navigate to the AWS Management Console and sign in with your credentials.
    - Create a VPC:
        - In the services menu, select VPC.
        - Click on Create VPC.
        - Choose a name and specify a CIDR block (e.g., 10.0.0.0/16).
        - Click Create.
    - Configure Subnets:
        - In the VPC dashboard, select Subnets and click on Create Subnet.
        - Create at least two subnets in different Availability Zones (AZs) (e.g., 10.0.1.0/24 for AZ1 and 10.0.2.0/24 for AZ2).
        - Ensure one of these subnets is designated as public by modifying the route table to include an Internet Gateway.
2. **Launch EC2 Instances:**
    - Create EC2 Instances:
        - Go to the EC2 dashboard and click on Launch Instance.
        - Select an Amazon Machine Image (AMI) (e.g., Amazon Linux) and an instance type (e.g., t2.micro).
        - Choose the VPC and the public subnet you created earlier.
        - Configure instance details and ensure to launch instances in both subnets for redundancy.
        - Repeat to launch additional instances as needed (e.g., launch three EC2 instances in total across the subnets).
3. **Set Up an Elastic Load Balancer (ELB):**
    - Create ELB:
        - In the EC2 dashboard, select Load Balancers and click on Create Load Balancer.
        - Choose Application Load Balancer (recommended for web applications).
    - Configure Load Balancer:
        - Name your load balancer and select the scheme (internet-facing).
        - Choose the VPC and select the subnets from different AZs.
    - Configure Listeners and Security Settings:
        - Set up listeners (e.g., HTTP on port 80 and/or HTTPS on port 443).
        - For HTTPS, request an SSL certificate from AWS Certificate Manager (ACM) or upload your own.
    - Configure Target Group:

        - Create a target group, selecting the protocol (HTTP) and the port (80).
        - Register your EC2 instances in this target group.
    - Health Checks:
        - Set up health checks to monitor instance availability (e.g., use the /health endpoint).
    - Review and Create:
        - Review the settings and click Create to launch the load balancer.
4. **Configure Auto Scaling:**
    - Create Auto Scaling Group:
        - In the EC2 dashboard, go to Auto Scaling Groups and click on Create Auto Scaling Group.
        - Select the launch template or configuration that matches your EC2 instances.
    - Define Scaling Policies:
        - Set the desired capacity and configure scaling policies (e.g., add instances if CPU utilization exceeds 70%).
    - Review and Create:
        - Review your settings and click Create Auto Scaling Group.
5. **Set Up Route 53:**
    - Create Hosted Zone:
        - Go to the Route 53 console and select Hosted Zones.
        - Click Create Hosted Zone and enter your domain name (e.g., yourcafe.com).
    - Configure DNS Records:
        - Create an A record that points to your Elastic Load Balancer's DNS name to direct traffic to your application.
6. **Monitor and Test:**
    - CloudWatch Monitoring:
        - Use AWS CloudWatch to monitor metrics such as CPU utilization and traffic to your application.
    - Test Application:
        - Simulate traffic and manually stop EC2 instances to verify that the ELB reroutes traffic to healthy instances.
    - Health Checks:
        - Ensure health checks are properly configured and that the application is accessible.

OUTPUT:





RESULT:

A highly available web application was successfully deployed on AWS, utilizing multiple EC2 instances behind an Elastic Load Balancer, ensuring minimal downtime and seamless user access.

Ex.15: CREATING A SCALABLE AND HIGHLY AVAILABLE ENVIRONMENT FOR THE CAFÉ

AIM:

To create a scalable and highly available environment for a café's web application using Amazon Web Services (AWS), ensuring minimal downtime and seamless user experience.

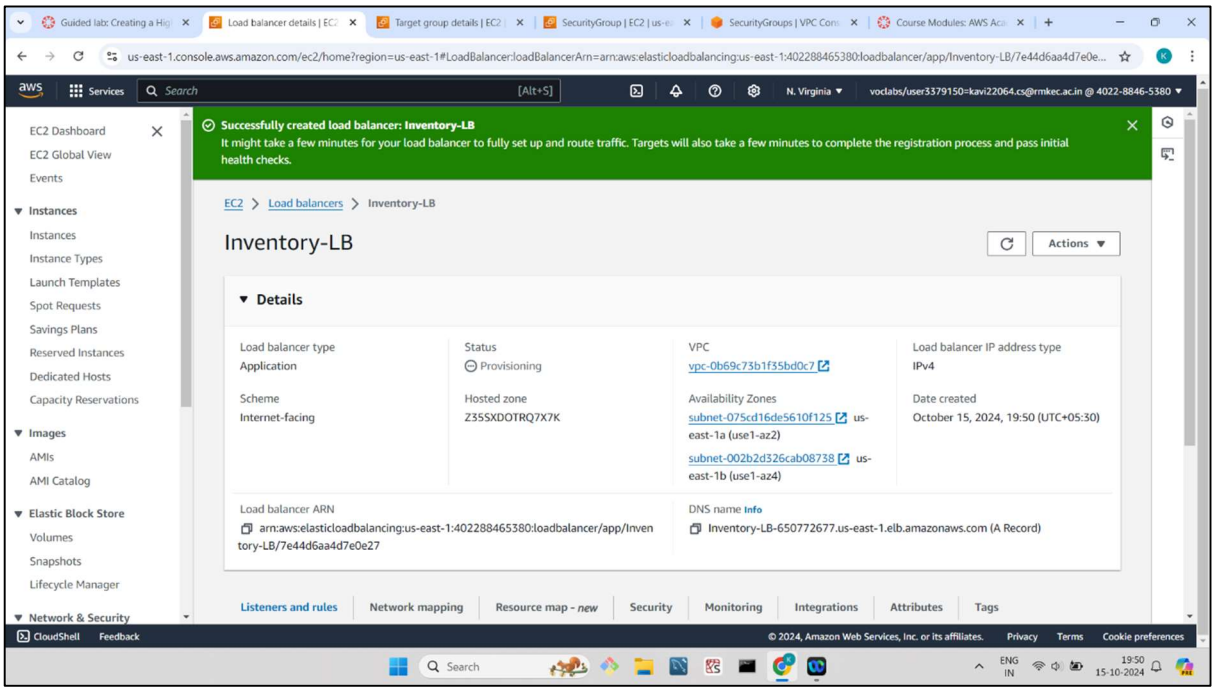
PROCEDURE:

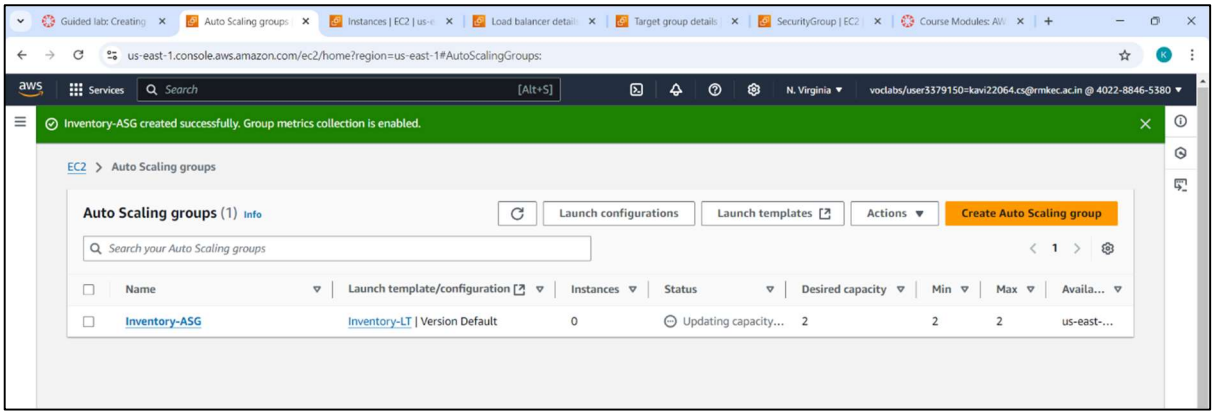
1. **Set Up a VPC:**
    - Log in to AWS Management Console:
        - Navigate to the AWS Management Console and sign in with your credentials.
    - Create a VPC:
        - In the services menu, select VPC.
        - Click on Create VPC.
        - Choose a name and specify a CIDR block (e.g., 10.0.0.0/16).
        - Click Create.
    - Configure Subnets:
        - In the VPC dashboard, select Subnets and click on Create Subnet.
        - Create at least two subnets in different Availability Zones (AZs) (e.g., 10.0.1.0/24 for AZ1 and 10.0.2.0/24 for AZ2).
        - Designate one of these subnets as public by modifying the route table to include an Internet Gateway.
2. **Launch EC2 Instances:**
    - Create EC2 Instances:
        - Go to the EC2 dashboard and click on Launch Instance.
        - Select an Amazon Machine Image (AMI) (e.g., Amazon Linux) and an instance type (e.g., t2.micro).
        - Choose the VPC and the public subnet you created earlier.
        - Configure instance details and ensure to launch instances in both subnets for redundancy.
        - Repeat to launch multiple EC2 instances (e.g., three EC2 instances total).
3. **Set Up an Elastic Load Balancer (ELB):**
    - Create ELB:
        - In the EC2 dashboard, select Load Balancers and click on Create Load Balancer.
        - Choose Application Load Balancer (recommended for web applications).
    - Configure Load Balancer:
        - Name your load balancer and select the scheme (internet-facing).
        - Choose the VPC and select the subnets from different AZs.
    - Configure Listeners and Security Settings:
        - Set up listeners (e.g., HTTP on port 80 and/or HTTPS on port 443).
        - For HTTPS, request an SSL certificate from AWS Certificate Manager (ACM) or upload your own.
    - Configure Target Group:

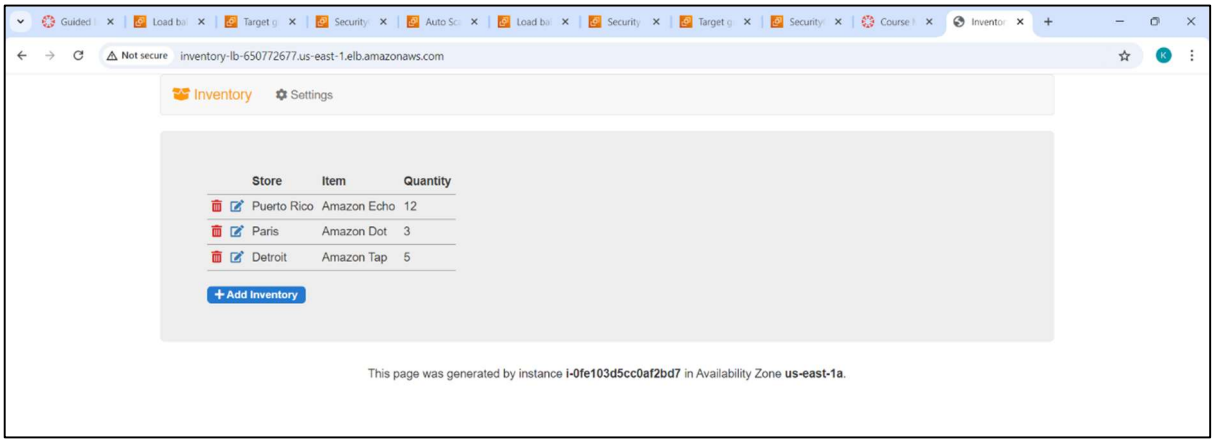
        - Create a target group, selecting the protocol (HTTP) and the port (80).
        - Register your EC2 instances in this target group.
    - Health Checks:
        - Set up health checks to monitor instance availability (e.g., use the /health endpoint).
    - Review and Create:
        - Review the settings and click Create to launch the load balancer.
4. **Create an RDS Instance:**
    - Launch RDS Instance:
        - Navigate to the RDS dashboard and select Create Database.
        - Choose a database engine (e.g., MySQL) and select the Multi-AZ deployment option for high availability.
    - Configure RDS:
        - Set the instance specifications (size, storage) and ensure it's in the same VPC.
        - Define database settings (DB name, username, and password).
        - Click Create Database to launch the instance.
5. **Configure Auto Scaling:**
    - Create Auto Scaling Group:
        - In the EC2 dashboard, go to Auto Scaling Groups and click on Create Auto Scaling group.
        - Select the launch template or configuration that matches your EC2 instances.
    - Define Scaling Policies:
        - Set the desired capacity and configure scaling policies (e.g., add instances if CPU utilization exceeds 70%).
    - Review and Create:
        - Review your settings and click Create Auto Scaling Group.
6. **Set Up Route 53:**
    - Create Hosted Zone:
        - Go to the Route 53 console and select Hosted Zones.
        - Click Create Hosted Zone and enter your domain name (e.g., yourcafe.com).
    - Configure DNS Records:
        - Create an A record that points to your Elastic Load Balancer's DNS name to direct traffic to your application.
7. **Monitor and Test:**
    - CloudWatch Monitoring:
        - Use AWS CloudWatch to monitor metrics such as CPU utilization, database connections, and traffic to your application.
    - Test Application:
        - Simulate traffic and manually stop EC2 instances to verify that the ELB reroutes traffic to healthy instances.
    - Health Checks:

- Ensure health checks are properly configured and that the application is accessible.

OUTPUT:







RESULT:

A scalable and highly available environment for the café's web application was successfully deployed on AWS, utilizing multiple EC2 instances behind an Elastic Load Balancer and an Amazon RDS instance for database management.

Ex.16: STREAMING DYNAMIC CONTENT USING AMAZON CLOUDFRONT

AIM:

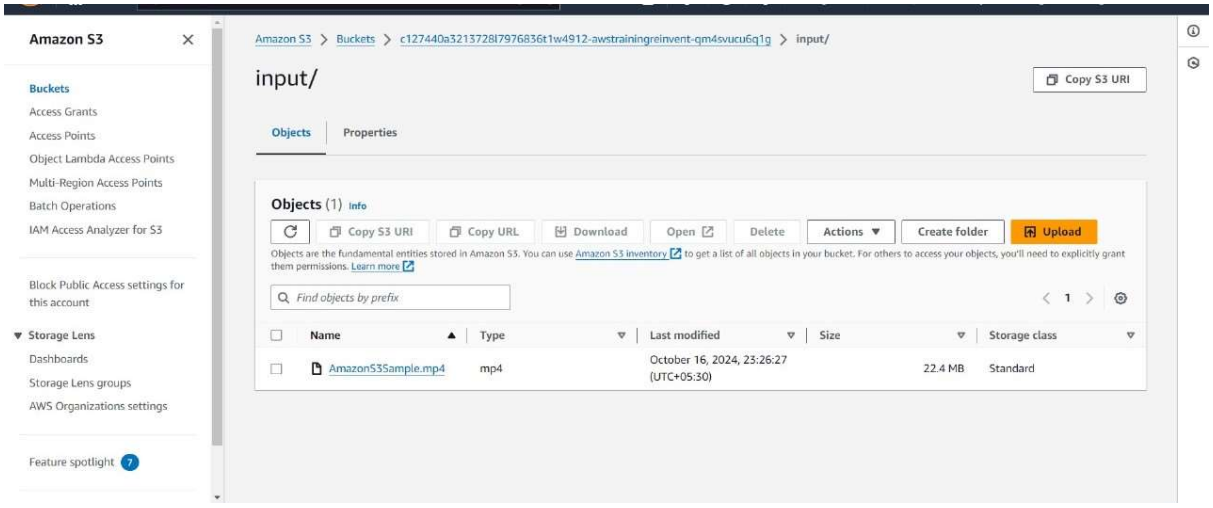
To stream dynamic content using Amazon CloudFront, providing a fast and reliable content delivery network (CDN) for web applications.

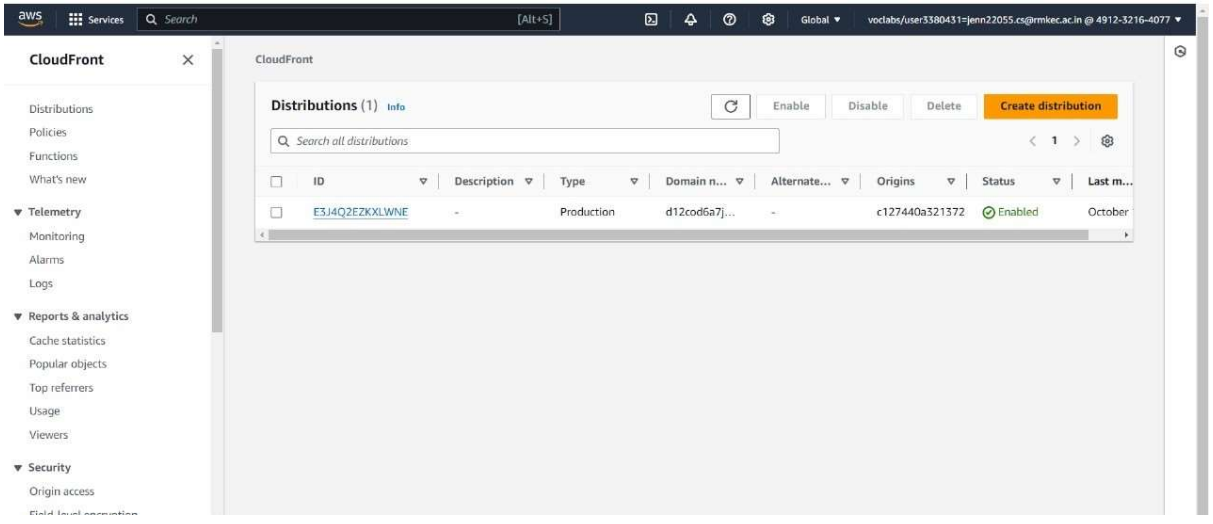
PROCEDURE:

1. **Create an S3 Bucket:**
    - Log in to AWS Management Console:
        - Navigate to the AWS Management Console and sign in with your credentials.
    - Create an S3 Bucket:
        - In the services menu, select S3.
        - Click on Create bucket.
        - Enter a unique bucket name (e.g., my-cafe-content) and select a region.
        - Configure settings as needed and click Create bucket.
    - Upload Content (Optional):
        - Select the bucket and click on Upload to add your dynamic content (e.g., HTML files, images, etc.).
2. **Create a CloudFront Distribution:**
    - Go to CloudFront:
        - In the AWS Management Console, search for and select CloudFront.
    - Create Distribution:
        - Click on Create Distribution.
        - Choose Web for the delivery method.
    - Configure Origin Settings:
        - For Origin Domain Name, select your S3 bucket (or enter the URL of your dynamic content origin, such as an EC2 instance or an application load balancer).
        - Set Origin Path if necessary.
        - Choose the Origin ID (this is automatically generated).
    - Configure Default Cache Behavior:
        - Set the Viewer Protocol Policy (e.g., Redirect HTTP to HTTPS).
        - Configure Allowed HTTP Methods (GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE).
        - Enable Cache Based on Selected Request Headers and choose the headers you want to include.
    - Configure Distribution Settings:
        - Set Price Class based on your expected traffic (e.g., Use Only U.S., Canada and Europe).
        - Optionally, set Custom Error Responses and Logging options.
    - Create Distribution:
        - Review the settings and click Create Distribution.
        - Wait for the distribution to deploy (this may take several minutes).
3. **Access Your Content:**

- **Test Streaming:**
  - Use the CloudFront domain name (or your custom domain) to access your dynamic content.
  - Check that the content streams correctly and is delivered quickly.
- **Monitor Performance:**
  - Use **AWS CloudWatch** to monitor metrics related to CloudFront, such as requests, errors, and data transfer.

OUTPUT:





RESULT:

Dynamic content was successfully streamed using Amazon CloudFront, leveraging its CDN capabilities to deliver content quickly and reliably to users.

Ex.17: BREAKING A MONOLITHIC NODE.JS APPLICATION INTO MICROSERVICES

AIM:

To break down a monolithic Node.js application into microservices and deploy them on AWS, improving scalability, maintainability, and deployment flexibility.

PROCEDURE:

1. Identify Components of the Monolithic Application:

- Analyze the Current Application:
  - Review the existing Node.js application architecture.
  - Identify distinct functionalities that can be separated into individual services (e.g., user management, product catalog, order processing).

2. Define Microservices:

- Determine Service Boundaries:
  - Create a list of microservices based on the identified components.
  - Define responsibilities and APIs for each service.
    - User Service: Manages user registration and authentication.
    - Product Service: Manages product information and inventory.
    - Order Service: Processes orders and handles payment.

3. Create Individual Microservices:

- Set Up the Project Structure:
  - For each microservice, create a separate directory and initialize it with npm init.
  - Install necessary dependencies (e.g., Express, Mongoose).
- Develop the Microservices:
  - Write code for each service. Example code for the User Service:

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
app.use(express.json());

mongoose.connect('mongodb://<your-mongo-db-url>/userdb', { useNewUrlParser: true, useUnifiedTopology: true });

const UserSchema = new mongoose.Schema({ username: String, password: String });
const User = mongoose.model('User', UserSchema);

app.post('/users', async (req, res) => {
  const user = new User(req.body);
  await user.save();
  res.status(201).send(user);
});

const PORT = process.env.PORT || 3000;
```

```
app.listen(PORT, () => {
  console.log(`User Service running on port ${PORT}`);
});
```

4. Containerize Microservices:

- **Create a Dockerfile:**
  - For each service, create a Dockerfile to containerize the application:

```
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["node", "server.js"]
```

5. Set Up AWS Services for Deployment:

- Use Amazon Elastic Container Service (ECS):
  - Create an ECS Cluster:
    - Log in to the AWS Management Console.
    - Navigate to ECS and create a new cluster (e.g., select "Networking only" for Fargate).
  - Create a Task Definition for Each Microservice:
    - In the ECS console, select "Task Definitions" and click "Create new Task Definition."
    - Choose Fargate as the launch type.
    - Define a container with:
      - Image: The Docker image of your microservice (you can push your images to Amazon Elastic Container Registry (ECR) first).
      - Memory and CPU: Allocate sufficient resources for each service.
      - Port Mapping: Specify which port your application runs on (e.g., port 3000).
- Deploy the Services:
  - Create a Service for Each Microservice:
    - Under the ECS cluster, click "Create" to add a new service.
    - Select the Task Definition and configure the service settings.
    - Set desired task count (e.g., 2 for redundancy) and enable service discovery if required.

6. Implement Load Balancing:

- Set Up an Application Load Balancer (ALB):
  - In the AWS Management Console, go to the EC2 service and select "Load Balancers."
  - Click "Create Load Balancer" and choose "Application Load Balancer."
  - Configure:
    - Name your load balancer and select the scheme (internet-facing).
    - Define listeners (HTTP on port 80 and/or HTTPS on port 443).
    - Choose target groups for each microservice, registering the ECS services as targets.

7. Database Management:

- Set Up Amazon RDS:
    - In the AWS Management Console, navigate to RDS and select "Create Database."
    - Choose a database engine (e.g., MySQL or PostgreSQL).
    - Enable Multi-AZ deployment for high availability.
    - Configure database settings (DB instance class, storage, and credentials).
- Connect Microservices to RDS:
    - Update the microservices code to connect to the RDS instance using the provided endpoint and credentials.

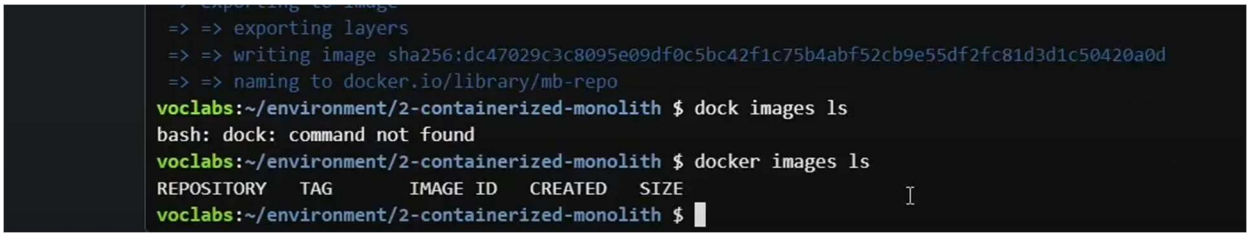
8. Configure IAM Roles and Security Groups:

- Create IAM Roles:
    - Ensure that your ECS tasks have the appropriate IAM role with permissions to access RDS and other AWS resources.
- Configure Security Groups:
    - Set security groups to allow traffic between the microservices and the database.
    - Allow inbound traffic to the load balancer from the internet.

9. Monitor and Test:

- Use AWS CloudWatch:
    - Set up CloudWatch alarms and dashboards to monitor the performance and health of your services.
- Conduct Testing:
    - Simulate traffic to your application and verify that the load balancer distributes requests to healthy instances.
    - Ensure each microservice operates independently and can communicate effectively.

OUTPUT:



RESULT:

The monolithic Node.js application was successfully decomposed into microservices and deployed on AWS using ECS, RDS, and ALB.

Ex.18: IMPLEMENTING A SERVERLESS ARCHITECTURE ON AWS

AIM:

To implement a serverless architecture on Amazon Web Services (AWS) that automatically scales based on demand and reduces the need for server management.

PROCEDURE:

1. Log in to AWS Management Console:
    - Navigate to AWS Management Console and sign in with your credentials.
2. Create a Lambda Function:
    - Access AWS Lambda:
        - In the services menu, search for Lambda and select it.
    - Create Function:
        - Click on Create function.
        - Choose Author from scratch.
        - Enter Function name (e.g., MyServerlessFunction).
        - Select Runtime (e.g., Python 3.x).
        - Choose Create a new role with basic Lambda permissions.
    - Function Code:
        - In the code editor, enter a simple function. Here's an example code snippet for a basic greeting function:

```
def lambda_handler(event, context):
    name = event.get('name', 'World')
    return {
        'statusCode': 200,
        'body': f'Hello, {name}!'
    }
```

    - Create Function:
        - Click Create function at the bottom.
3. Test the Lambda Function:
    - Create a Test Event:
        - Click on Test and configure a test event. Use this example:

```
{
  "name": "Cafe User"
}
```

    - Run the Test:
        - Click on Test again to execute the function. You should see the response:

```
{
  "statusCode": 200,
  "body": "Hello, Cafe User!"
}
```

4. **Set Up API Gateway:**
    - Access API Gateway:
        - In the services menu, search for API Gateway and select it.
    - Create API:

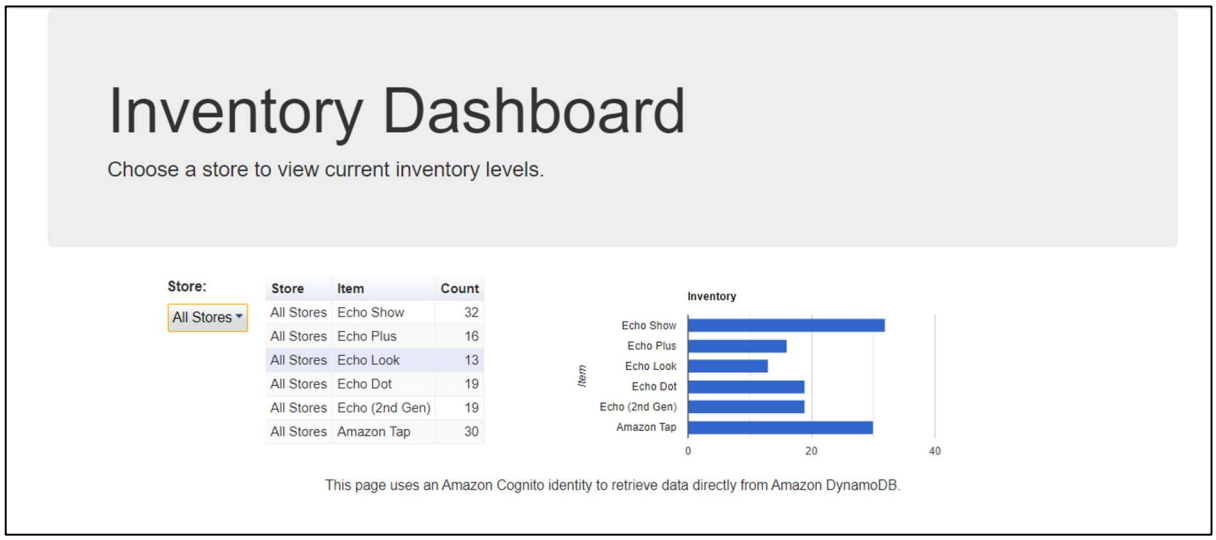
- Click on Create API.
- Choose HTTP API and click Build.

- Configure API:
  - Click Add integration and select Lambda function.
  - Choose the Lambda function you created (e.g., MyServerlessFunction).
  - Click Next, set up routes (e.g., /greet), and choose ANY method.
- Deploy API:
  - Click on Deploy and follow the prompts to create a stage (e.g., prod).
- Copy the API Endpoint:
  - Note the API endpoint URL for testing.

5. **Test the API Gateway:**
   - Use a tool like Postman or a web browser to test the API:
     - Make a request to the endpoint (e.g., https://your-api-id.execute-api.region.amazonaws.com/prod/greet?name=Cafe User).
   - Verify that the response is:

```
{
  "statusCode": 200,
  "body": "Hello, Cafe User!"
}
```

OUTPUT:



RESULT:

A serverless architecture was successfully implemented on AWS using Lambda and API Gateway.

Ex.19: IMPLEMENTING A SERVERLESS ARCHITECTURE FOR THE CAFÉ

AIM:

To create a serverless architecture on AWS specifically for a café application that allows customers to place orders online without managing servers, ensuring scalability and cost-effectiveness.

PROCEDURE:

1. **Log in to AWS Management Console:**
   - Navigate to the AWS Management Console and sign in with your credentials.
2. **Create a Lambda Function:**
   - Access AWS Lambda:
     - In the services menu, search for Lambda and select it.
   - Create Function:
     - Click on Create function.
     - Choose Author from scratch.
     - Enter Function name (e.g., CafeOrderFunction).
     - Select Runtime (e.g., Python 3.x).
     - Choose Create a new role with basic Lambda permissions.
   - Function Code:
     - In the code editor, enter a simple function to process orders. Here's an example code snippet:

```
def lambda_handler(event, context):
    order_details = event.get('order', {})
    item = order_details.get('item', 'Unknown')
    quantity = order_details.get('quantity', 1)
    return {
        'statusCode': 200,
        'body': f'Order received: {quantity} x {item}'
    }
```

   - Create Function:
     - Click Create function at the bottom.
3. **Test the Lambda Function:**
   - Create a Test Event:

```
{
  "order": {
    "item": "Coffee",
    "quantity": 2
  }
}
```

   - Run the Test:
     - Click on Test again to execute the function. You should see the response:

```
{
  "statusCode": 200,
  "body": "Order received: 2 x Coffee"
```

```
}
```

4. **Set Up API Gateway:**
    - Access API Gateway:
        - In the services menu, search for API Gateway and select it.
    - Create API:
        - Click on Create API.
        - Choose HTTP API and click Build.
    - Configure API:
        - Click Add integration and select Lambda function.
        - Choose the Lambda function you created (e.g., CafeOrderFunction).
        - Click Next, set up routes (e.g., /order), and choose ANY method.
    - Deploy API:
        - Click on Deploy and follow the prompts to create a stage (e.g., prod).
    - Copy the API Endpoint:
        - Note the API endpoint URL for testing.
5. **Test the API Gateway:**
    - Use a tool like Postman or a web browser to test the API:
        - Make a request to the endpoint (e.g., https://your-api-id.execute-api.region.amazonaws.com/prod/order).
        - Include a JSON body in the request:

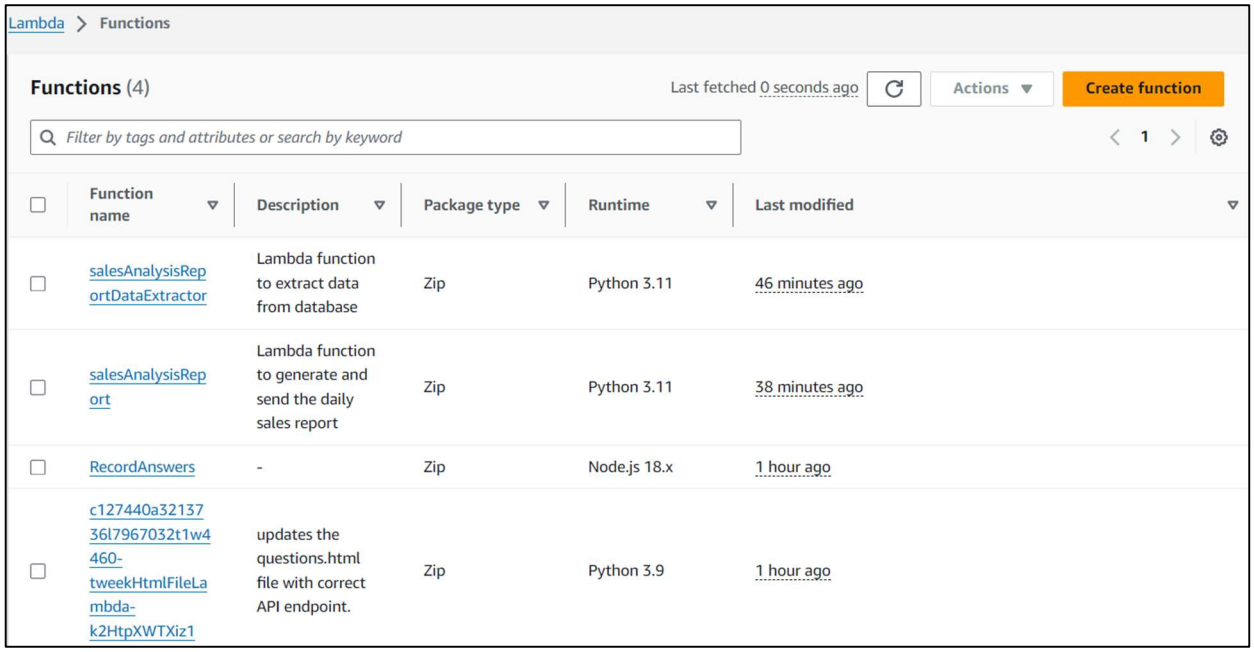
```
{
  "order": {
    "item": "Coffee",
    "quantity": 2
  }
}
```

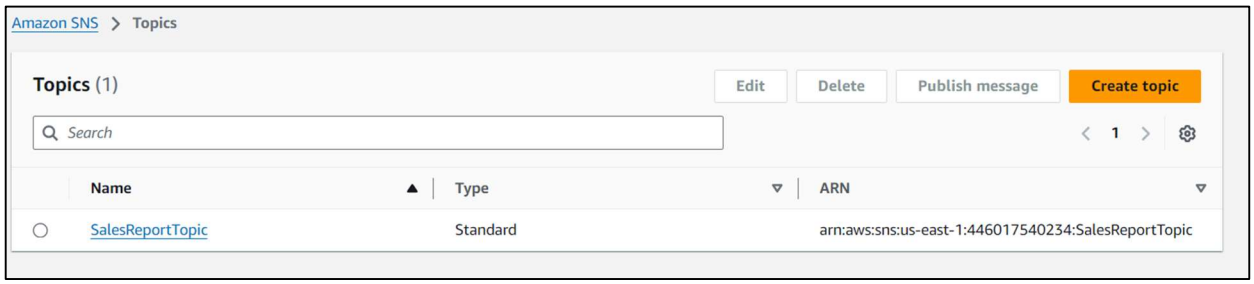
    - Verify that the response is:

```
{
  "statusCode": 200,
  "body": "Order received: 2 x Coffee"
}
```

6. **Monitor and Optimize:**
    - Use AWS CloudWatch to monitor Lambda function execution and API usage.
    - Optimize the function code and API configuration based on performance metrics.

OUTPUT:





Sales Report Topic <no-reply@sns.amazonaws.com>
to me

Sales Analysis Report
Date: 2024-10-16

Product Group: Pastries

Item Name	Quantity
*********	********
Croissant	29
Donut	23
Chocolate Chip Cookie	18
Muffin	6
Strawberry Blueberry Tart	34
Strawberry Tart	33

Product Group: Drinks

Item Name	Quantity
*********	********
Coffee	33
Hot Chocolate	17
Latte	24

RESULT:

A serverless architecture tailored for the café application was successfully implemented on AWS.

Ex.20: CREATING AN AWS LAMBDA FUNCTION AND EXPLORE USING AWS LAMBDA WITH AMAZON S3

AIM:

To create an AWS Lambda function that triggers upon the upload of files to an Amazon S3 bucket, demonstrating how to process the files in a serverless architecture.

PROCEDURE:

1. **Log in to AWS Management Console:**
   - Navigate to the AWS Management Console and sign in with your credentials.
2. **Create an S3 Bucket:**
   - Access Amazon S3:
     - In the services menu, search for S3 and select it.
   - Create Bucket:
     - Click on Create bucket.
     - Enter a unique Bucket name (e.g., my-cafe-uploads).
     - Choose a Region and click Create bucket.
3. **Create a Lambda Function:**
   - Access AWS Lambda:
     - In the services menu, search for Lambda and select it.
   - Create Function:
     - Click on Create function.
     - Choose Author from scratch.
     - Enter Function name (e.g., ProcessS3Upload).
     - Select Runtime (e.g., Python 3.x).
     - Choose Create a new role with basic Lambda permissions.
   - Function Code:
     - In the code editor, enter a simple function to process the uploaded file. Here's an example code snippet:

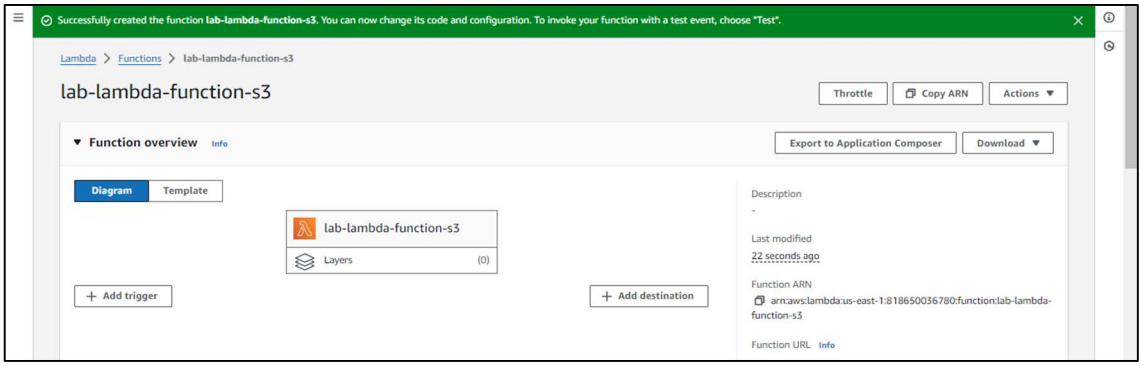
```
import json
import boto3

def lambda_handler(event, context):
    # Get the S3 bucket name and object key from the event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Create an S3 client
    s3 = boto3.client('s3')

    # Retrieve the object
    response = s3.get_object(Bucket=bucket, Key=key)
    file_content = response['Body'].read().decode('utf-8')

    # Process the file content (example: print it)
```

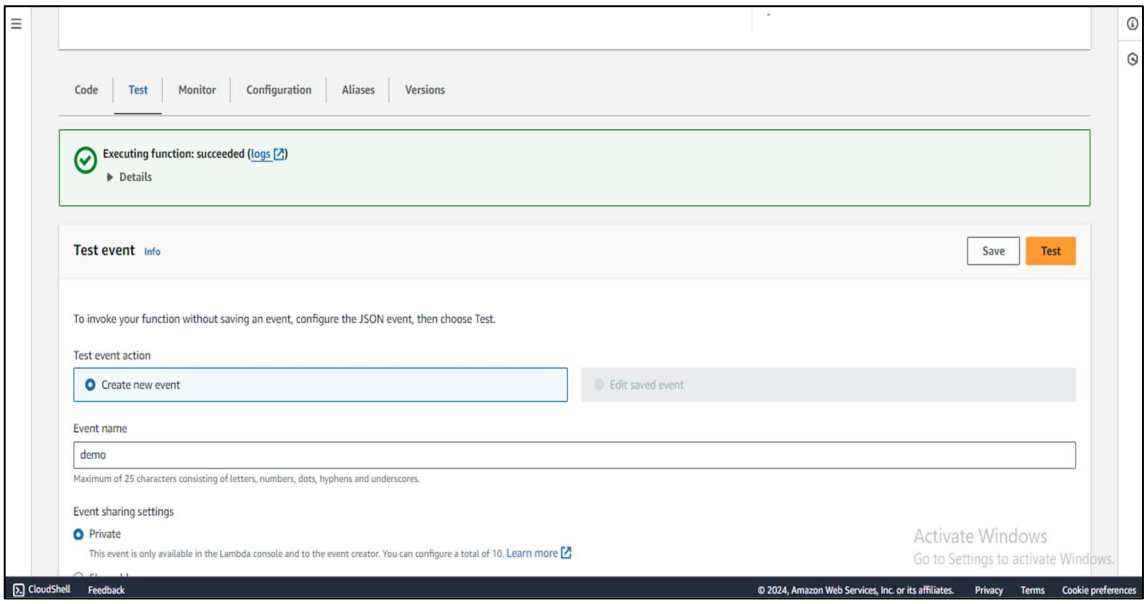
```
        print(f'File content from {key}:')
        print(file_content)

        return {
            'statusCode': 200,
            'body': json.dumps('File processed successfully!')
        }
```

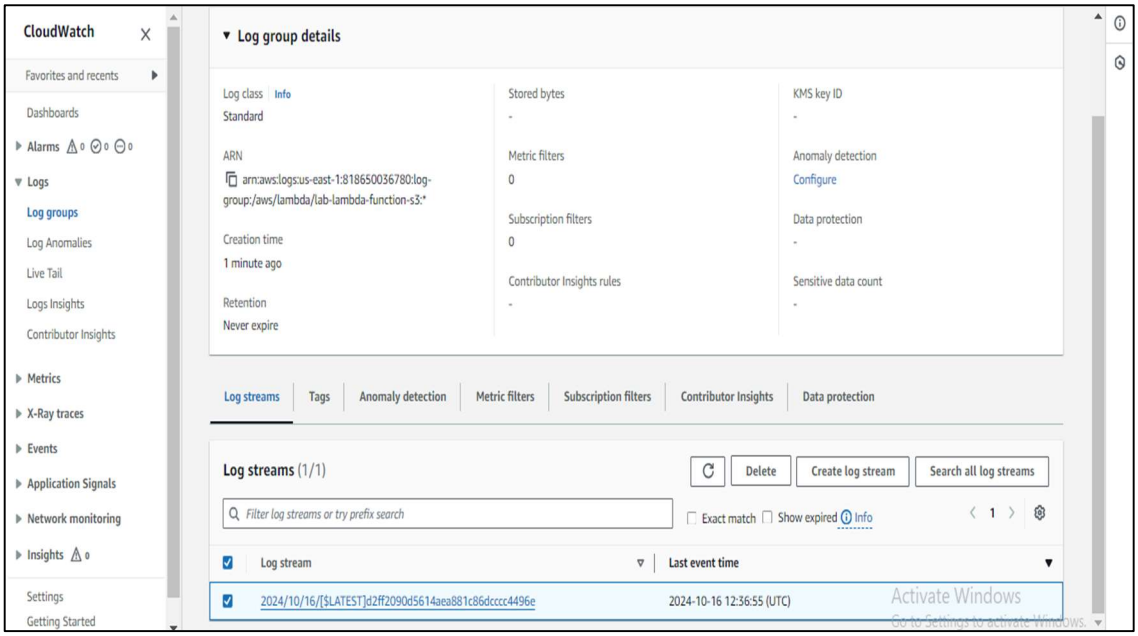
- Create Function:
  - Click Create function at the bottom.

4. **Set Up S3 Trigger:**
   - Configure Trigger:
     - In the Lambda function dashboard, scroll down to Function overview.
     - Click on Add trigger and select S3.
   - Choose Bucket:
     - Select the S3 bucket you created earlier (e.g., my-cafe-uploads).
   - Event Type:
     - Choose All object create events.
   - Enable Trigger:
     - Ensure the trigger is enabled and click Add.
5. **Test the Setup:**
   - Upload a File to S3:
     - Go back to the S3 bucket, click Upload, and select a text file (e.g., order.txt).
   - Check Lambda Execution:
     - After the file upload, go to the Monitoring tab in your Lambda function dashboard.
     - Check the Logs in AWS CloudWatch to verify that the function processed the uploaded file. You should see the file content printed in the logs.
6. **Clean Up Resources:**
   - Once you've finished testing, delete the Lambda function and S3 bucket to avoid incurring charges.

OUTPUT:







RESULT:

An AWS Lambda function was successfully created that triggers on file uploads to an Amazon S3 bucket. The function retrieves and processes the file content, demonstrating the integration of AWS Lambda with S3 in a serverless architecture.