Distributed System and Cloud Computing Lab

Subject Code: MCAL32

A Practical Journal Submitted in Fulfilment of the Degree

of

MASTER

In

COMPUTER APPLICATION

Year 2024-2025

By

Mr. Agrawal Yash Gopal

(Application Id: - 53715)

Semester- III (CBCS)

Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

University of Mumbai

PCP Centre

[Vidyavardhini's College of Technology – Vasai Road, Palghar 401202]

Institute of Distance and Open Learning,

Vidya Nagari, Kalina, Santacruz (E) -400098

CERTIFICATE

This to certify that, **Mr. Agrawal Yash Gopal** appearing **Master in Computer Application (Semester III - CBCS) Application ID: 53715** has satisfactorily completed the prescribed practical of **MCAL32- Distributed System and Cloud Computing Lab** as laid down by the University of Mumbai for the academic year 2024-25.

Teacher in charge | Examiners | Coordinator IDOL, MCA
University of Mumbai

Date:- 09/01/2025

Place: - Vasai

Index

Sr. No.	Practical	Signature
1.	Write a program to develop multi-client server application where multiple clients chat with each other concurrently.	
2.	To implement a server calculator using RPC concept.	
3.	Demonstrate a sample RMI Java Application.	
4.	Create RMI Database Application.	
5.	Write a program to demonstrate mutual exclusion using Producer Consumer Problem in Java.	
6.	Implementation of Storage as a Service using Azure.	
7.	Implementation of Identity Access Management (IAM) using AWS Cloud.	
8.	To implement SaaS using some Java online terminal to run program.	
9.	Write a code to take username as input and dynamically changing the text content of Web Page.	
10.	To demonstrate Web Application Development using NetBeans.	

Practical 1

AIM: Write a program to develop multi-client server application where multiple clients chat with each other concurrently.

SOURCE CODE:

MultithreadedSocketServer.java

```
import java.net.*;
import java.io.*;

public class MultithreadedSocketServer
{
   public static void main(String[] args) throws Exception
   {
      try
      {
         ServerSocket server = new ServerSocket(8888);
         int counter = 0;
         System.out.println("Server Started");

         while (true)
         {
            counter++;
            Socket serverClient = server.accept();  // Server accepts the client connection
            System.out.println(" >> " + "Client No:" + counter + " started!");

            ServerClientThread sct = new ServerClientThread(serverClient, counter); // Send the request to a
separate thread
            sct.start();
         }
      }
      catch (Exception e)
      {
         System.out.println(e);
      }
   }
}
```

ServerClientThread.java

```
import java.net.*;
import java.io.*;

class ServerClientThread extends Thread
{
   Socket serverClient;
   int clientNo;
   int square;

   ServerClientThread(Socket inSocket, int counter)
```

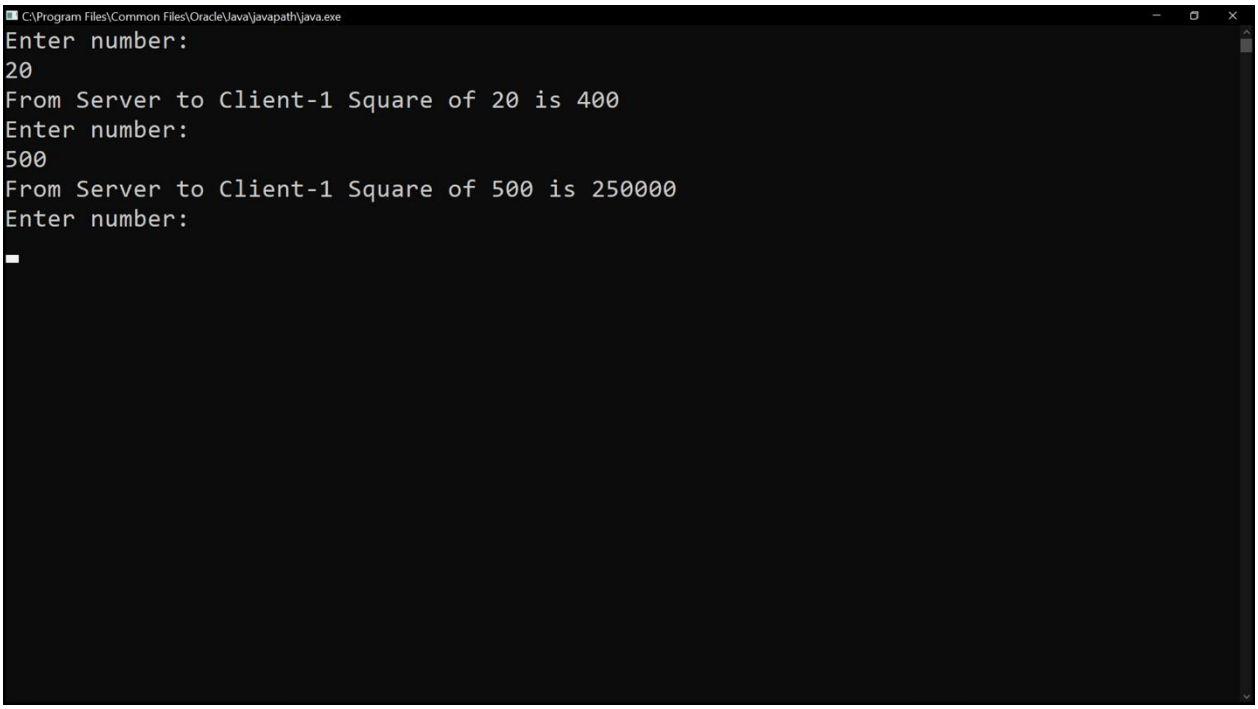
```
    {
        serverClient = inSocket;
        clientNo = counter;
    }

    public void run()
    {
        try
        {
            DataInputStream inStream = new DataInputStream(serverClient.getInputStream());
            DataOutputStream outStream = new DataOutputStream(serverClient.getOutputStream());

            String clientMessage = "", serverMessage = "";

            while (!clientMessage.equals("bye"))
            {
                clientMessage = inStream.readUTF();
                System.out.println("From Client-" + clientNo + ": Number is: " + clientMessage);
                square = Integer.parseInt(clientMessage) * Integer.parseInt(clientMessage);

                serverMessage = "From Server to Client-" + clientNo + " Square of " + clientMessage + " is " + square;
                outStream.writeUTF(serverMessage);
                outStream.flush();
            }

            inStream.close();
            outStream.close();
            serverClient.close();
        }
        catch (Exception ex)
        {
            System.out.println(ex);
        }
        finally
        {
            System.out.println("Client -" + clientNo + " exit!! ");
        }
    }
}
```

TCPClient.java

```
import java.net.*;
import java.io.*;

public class TCPClient
{
    public static void main(String[] args) throws Exception
    {
        try
        {
            Socket socket = new Socket("127.0.0.1", 8888);
```

```
        DataInputStream inStream = new DataInputStream(socket.getInputStream());
        DataOutputStream outStream = new DataOutputStream(socket.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String clientMessage = "", serverMessage = "";

        while (!clientMessage.equals("bye"))
        {
           System.out.println("Enter number: ");
           clientMessage = br.readLine();
           outStream.writeUTF(clientMessage);
           outStream.flush();

           serverMessage = inStream.readUTF();
           System.out.println(serverMessage);
        }

        outStream.close();
        inStream.close();
        socket.close();
     }
     catch (Exception e)
     {
        System.out.println(e);
     }
  }
}
```

OUTPUT:



Practical 2

AIM: To implement a server calculator using RPC concept.

SOURCE CODE:

RPCClient.java

```
import java.io.*;
import java.net.*;

class RPCClient {
   RPCClient() {
      try {
         InetAddress ia = InetAddress.getLocalHost();
         DatagramSocket ds = new DatagramSocket();
         DatagramSocket ds1 = new DatagramSocket(1300);

         System.out.println("\nRPC Client\n");
         System.out.println("Enter method name and parameters like: add 3 4");

         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
         while (true) {
            String str = br.readLine();
            byte[] b = str.getBytes();
            DatagramPacket dp = new DatagramPacket(b, b.length, ia, 1200);
            ds.send(dp);

            dp = new DatagramPacket(b, b.length);
            ds1.receive(dp);
            String s = new String(dp.getData(), 0, dp.getLength());
            System.out.println("\nResult = " + s + "\n");
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      new RPCClient();
   }
}
```

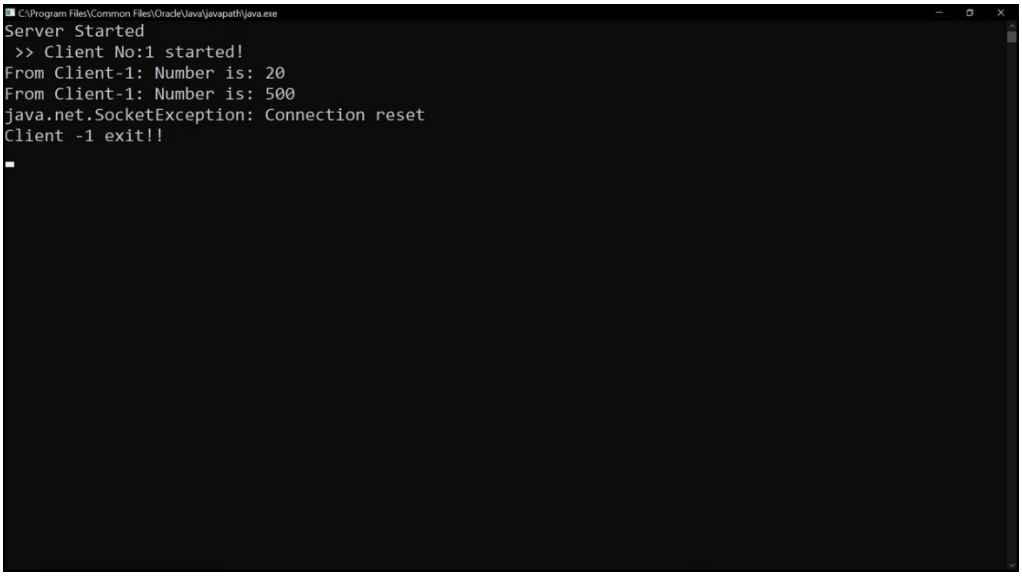
RPCServer.java

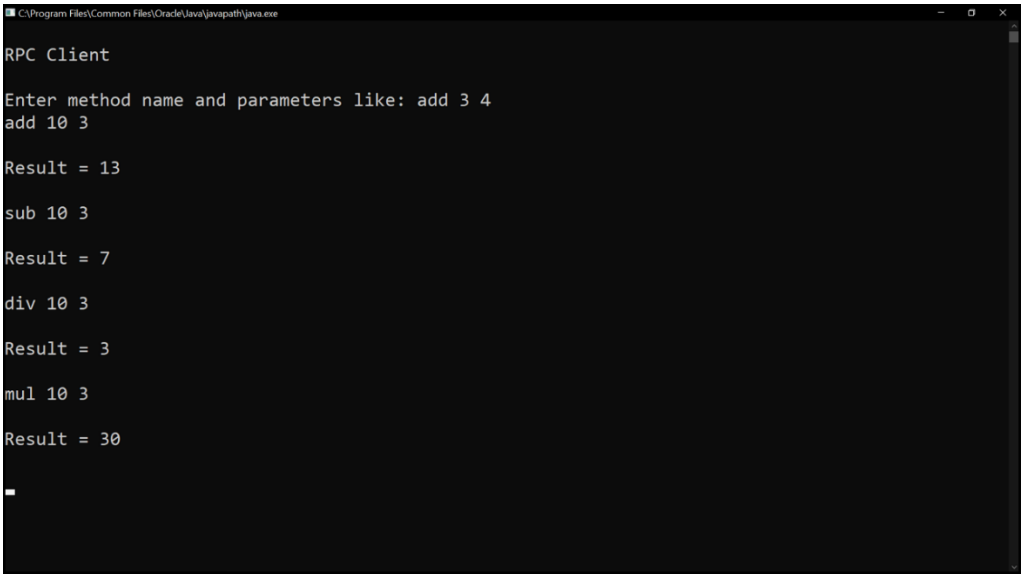
```
import java.net.*;
import java.util.StringTokenizer;

class RPCServer {
   DatagramSocket ds;
   DatagramPacket dp;
   String str, methodName, result;
   int val1, val2;
```

```
RPCServer() {
   try {
      ds = new DatagramSocket(1200);
      byte[] b = new byte[4096];

      while (true) {
         dp = new DatagramPacket(b, b.length);
         ds.receive(dp);
         str = new String(dp.getData(), 0, dp.getLength());

         if (str.equalsIgnoreCase("q")) {
            System.exit(1);
         } else {
            StringTokenizer st = new StringTokenizer(str, " ");
            int i = 0;
            while (st.hasMoreTokens()) {
               String token = st.nextToken();
               methodName = token;
               val1 = Integer.parseInt(st.nextToken());
               val2 = Integer.parseInt(st.nextToken());
            }
         }

         InetAddress ia = InetAddress.getLocalHost();

         // Perform the operation based on the method name
         if (methodName.equalsIgnoreCase("add")) {
            result = "" + add(val1, val2);
         } else if (methodName.equalsIgnoreCase("sub")) {
            result = "" + sub(val1, val2);
         } else if (methodName.equalsIgnoreCase("mul")) {
            result = "" + mul(val1, val2);
         } else if (methodName.equalsIgnoreCase("div")) {
            result = "" + div(val1, val2);
         }

         byte[] b1 = result.getBytes();
         DatagramSocket ds1 = new DatagramSocket();
         DatagramPacket dp1 = new DatagramPacket(b1, b1.length, InetAddress.getLocalHost(), 1300);
         System.out.println("result: " + result + "\n");
         ds1.send(dp1);
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
}

public int add(int val1, int val2) {
   return val1 + val2;
}
```

```
    public int sub(int val1, int val2) {
        return val1 - val2;
    }

    public int mul(int val1, int val2) {
        return val1 * val2;
    }

    public int div(int val1, int val2) {
        return val1 / val2;
    }

    public static void main(String[] args) {
        new RPCServer();
    }
}
```

OUTPUT:





Practical 3

AIM: Demonstrate a sample RMI Java Application.

SOURCE CODE:

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
   public static void main(String[] args) {
      try {
         // Connect to the RMI registry and look up the HelloService
         Registry registry = LocateRegistry.getRegistry("localhost", 5000);
         Hello hello = (Hello) registry.lookup("HelloService");

         // Invoke the remote method
         hello.printMessage();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
   void printMessage() throws RemoteException;
}
```

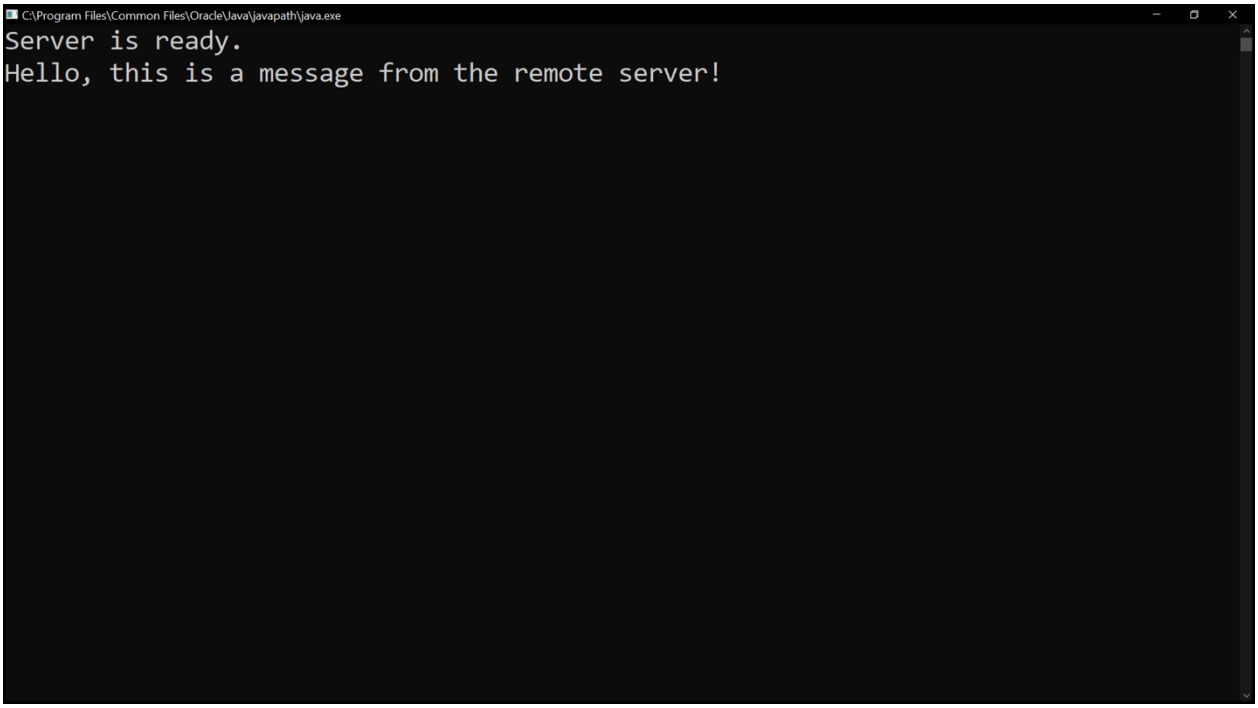
HelloImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {
   // Constructor
   protected HelloImpl() throws RemoteException {
      super();
   }

   // Implementation of the remote method
   @Override
   public void printMessage() throws RemoteException {
      System.out.println("Hello, this is a message from the remote server!");
   }
}
```

Server.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
   public static void main(String[] args) {
      try {
         // Create the remote object and bind it to the RMI registry
         HelloImpl hello = new HelloImpl();
         Registry registry = LocateRegistry.createRegistry(5000);
         registry.rebind("HelloService", hello);

         System.out.println("Server is ready.");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

OUTPUT:



Practical 4

AIM: Create RMI Database Application.

SOURCE CODE:

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;

public class Client {
   public static void main(String[] args) {
      try {
         // Connect to the RMI registry on port 6000
         Registry registry = LocateRegistry.getRegistry("localhost", 6000);
         DatabaseService dbService = (DatabaseService) registry.lookup("DatabaseService");

         // Call the remote method and print the result
         List<String> students = dbService.getStudentList();
         System.out.println("Student List:");
         // Printing a predefined student list
         System.out.println("ID: 1, Name: Alice, Grade: A");
         System.out.println("ID: 2, Name: Bob, Grade: B");
         System.out.println("ID: 3, Name: Charlie, Grade: A");

         // Iterate over the retrieved list and print each student's details
         for (String student : students) {
            System.out.println(student);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

DatabaseService.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface DatabaseService extends Remote {
   List<String> getStudentList() throws RemoteException;
}
```

DatabaseServiceImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class DatabaseServiceImpl extends UnicastRemoteObject implements DatabaseService {
   protected DatabaseServiceImpl() throws RemoteException {
      super();
   }

   @Override
   public List<String> getStudentList() throws RemoteException {
      List<String> students = new ArrayList<>();
      try {
         // Connect to the database
         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/RMIExampleDB",
"root", "password");
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT * FROM students");

         // Fetch student data
         while (rs.next()) {
            students.add("ID: " + rs.getInt("id") + ", Name: " + rs.getString("name") + ", Grade: " +
rs.getString("grade"));
         }

         conn.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return students;
   }
}
```

Server.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
   public static void main(String[] args) {
      try {
         // Create and bind the remote object
         DatabaseServiceImpl dbService = new DatabaseServiceImpl();
         Registry registry = LocateRegistry.createRegistry(6000); // Custom port
         registry.rebind("DatabaseService", dbService);

         System.out.println("Server is ready on port 5000.");
      } catch (Exception e) {
         e.printStackTrace();
      }
   } }
```

OUTPUT:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5131]
(c) Microsoft Corporation. All rights reserved.

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 4>javac -cp .:mysql-connector-java.jar Databa

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 4>start rmiregistry

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 4>start java -cp .;mysql-connector-java.jar S

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 4>java -cp .;mysql-connector-java.jar Client
Student List:
ID: 1, Name: Alice, Grade: A
ID: 2, Name: Bob, Grade: B
ID: 3, Name: Charlie, Grade: A

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 4>
```

Practical 5

AIM: Write a program to demonstrate mutual exclusion using Producer Consumer Problem in Java.

SOURCE CODE:

Consumer.java

```
class Consumer extends Thread {
   private SharedBuffer buffer;

   public Consumer(SharedBuffer buffer) {
      this.buffer = buffer;
   }

   @Override
   public void run() {
      try {
         for (int i = 0; i < 5; i++) {
            buffer.consume();
            Thread.sleep(1500); // Simulating time taken to consume
         }
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

Producer.java

```
class Producer extends Thread {
   private SharedBuffer buffer;

   public Producer(SharedBuffer buffer) {
      this.buffer = buffer;
   }

   @Override
   public void run() {
      try {
         for (int i = 0; i < 5; i++) {
            buffer.produce(i);
            Thread.sleep(1000); // Simulating time taken to produce
         }
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

ProducerConsumerExample.java

```
public class ProducerConsumerExample {
   public static void main(String[] args) {
      SharedBuffer buffer = new SharedBuffer();

      // Create and start producer and consumer threads
      Thread producer = new Producer(buffer);
      Thread consumer = new Consumer(buffer);

      producer.start();
      consumer.start();
   }
}
```

SharedBuffer.java

```
class SharedBuffer {
   private int buffer;
   private boolean isEmpty = true; // Buffer state flag

   // Producer produces the item
   public synchronized void produce(int value) throws InterruptedException {
      while (!isEmpty) {
         wait(); // Wait if buffer is not empty
      }
      buffer = value;  // Producing the item
      System.out.println("Produced: " + value);
      isEmpty = false;  // Buffer is no longer empty
      notifyAll();  // Notify consumers that item is available
   }

   // Consumer consumes the item
   public synchronized void consume() throws InterruptedException {
      while (isEmpty) {
         wait(); // Wait if buffer is empty
      }
      System.out.println("Consumed: " + buffer);
      isEmpty = true;  // Buffer is empty now
      notifyAll();  // Notify producers that buffer is empty
   }
}
```

OUTPUT:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5131]
(c) Microsoft Corporation. All rights reserved.

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 5>javac *.java

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 5>java ProducerConsumerExample
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4

D:\Swagat\Study\MCA SEM 3\Practicals\DS & CC\Practical 5>
```

Practical 6

AIM: Implementation of Storage as a Service using Azure.

Theory:

Amazon S3 (Simple Storage Service) is a scalable, durable, and low-latency cloud storage service offered by AWS (Amazon Web Services). It is designed for storing and retrieving any amount of data, anytime, from anywhere on the web. S3 is commonly used for backup, archiving, content storage, and as a data lake for big data analytics.

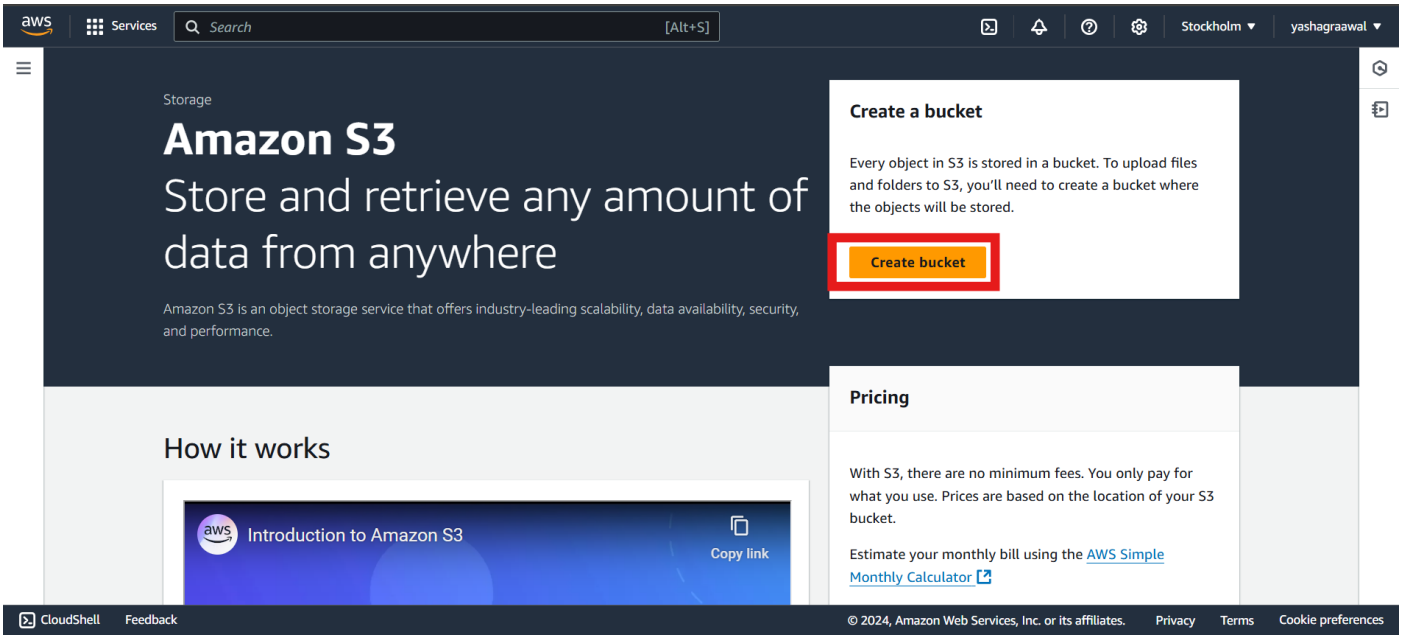
Key features include:

- Scalability: Automatically scales to accommodate growing data without needing to manage infrastructure.
- Durability: S3 stores data across multiple facilities, offering 99.999999999% durability, ensuring data is protected against loss.
- Storage Classes: S3 offers different storage classes, like Standard, Intelligent-Tiering, Glacier, and others, allowing users to optimize costs based on their data access patterns.
- Security: S3 provides robust security features such as encryption (at rest and in transit), access control policies, and identity management.
- Easy Access: Data can be accessed via the web, APIs, or AWS SDKs, and can be easily integrated with other AWS services.

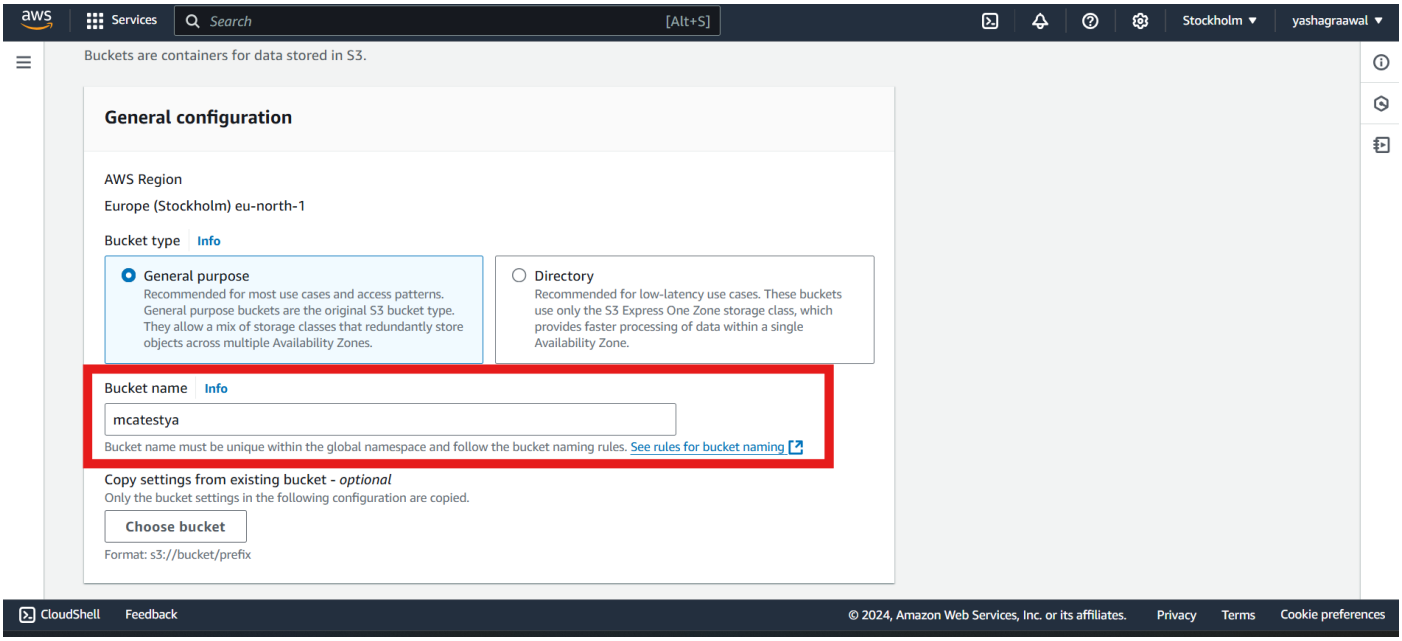
It is widely used for hosting static websites, storing backup data, and archiving large datasets for analytics or disaster recovery.

Steps to Create the Storage service on AWS:

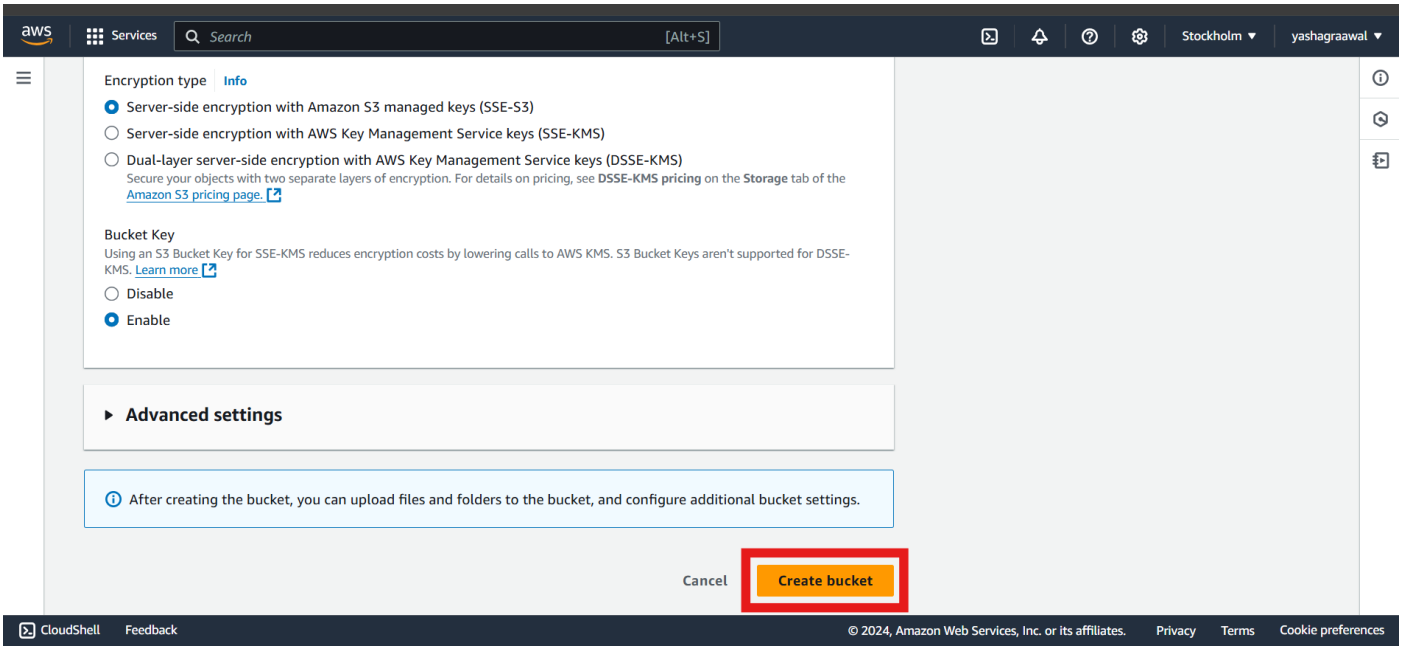
Step 1: Login to your AWS Root account.
Step 2: Naviate to S3 Service.
Step 3: Click on Create Bucket.



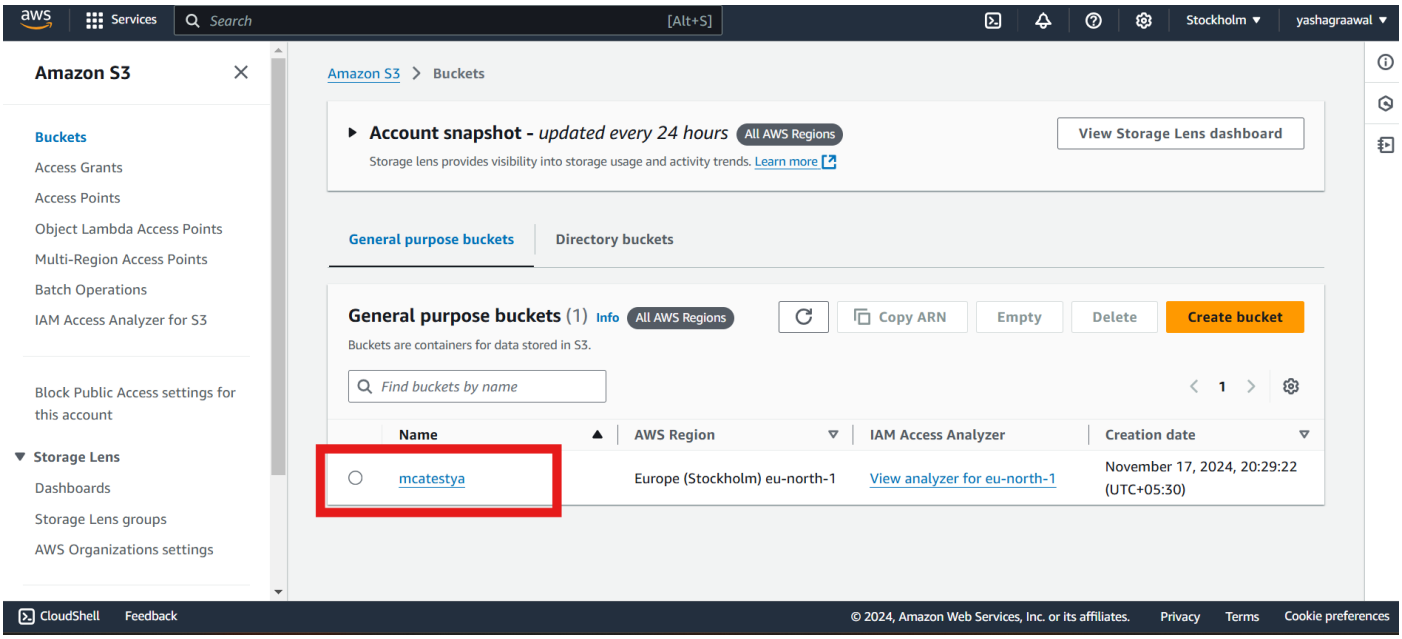
Step 4: Enter a unique Bucket name.



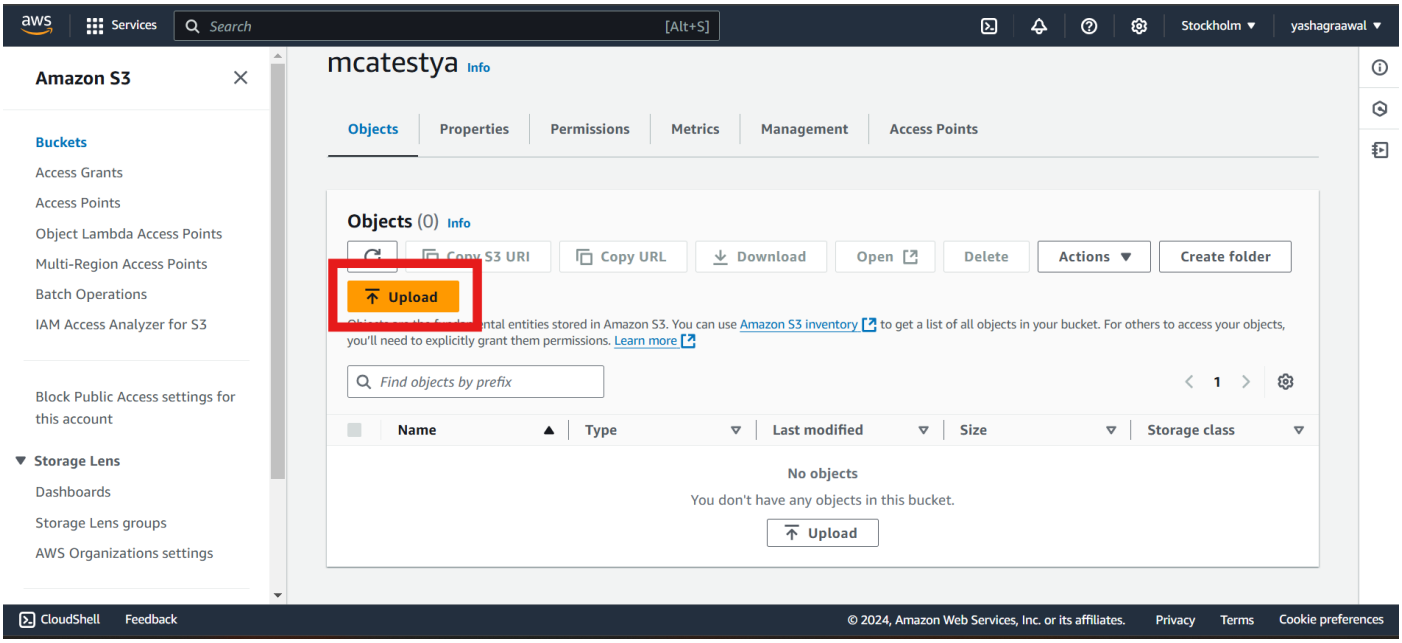
Step 5: Keep default settings or customize as per requirements (e.g., versioning, encryption), and click on Create Bucket.



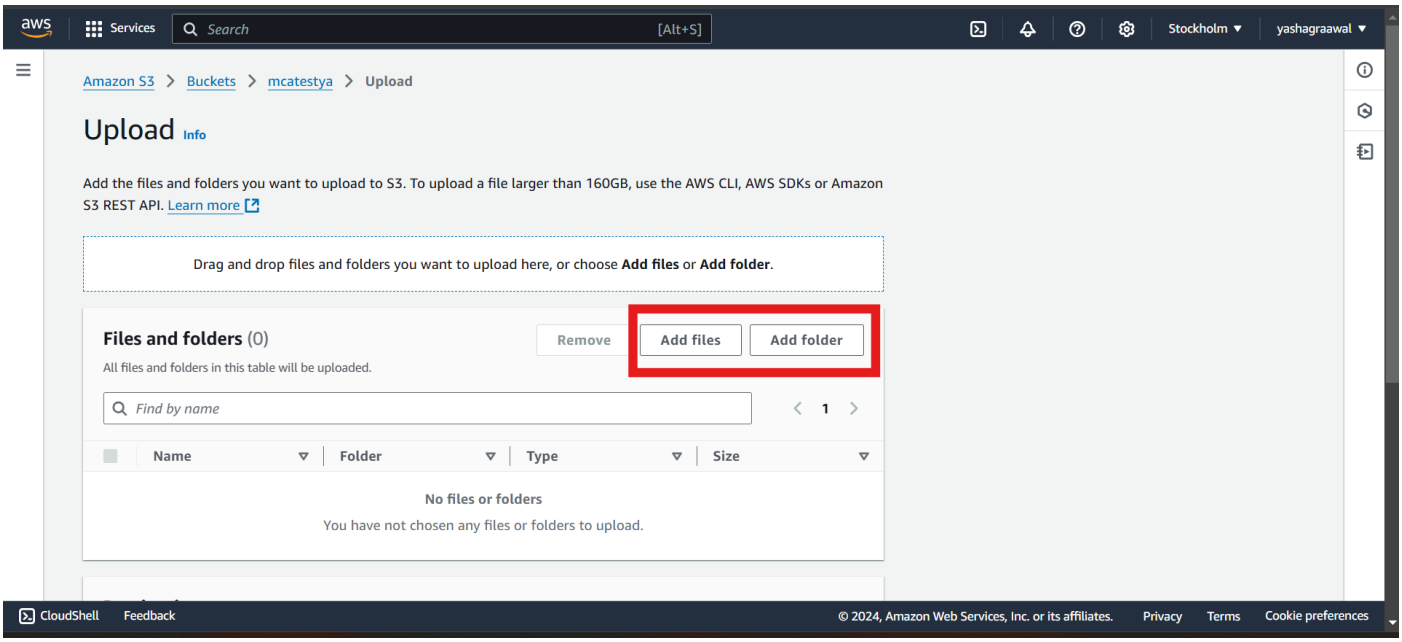
Step 6: Once Bucket is created, it will be visible under Bucket list click on the name of the bucket.

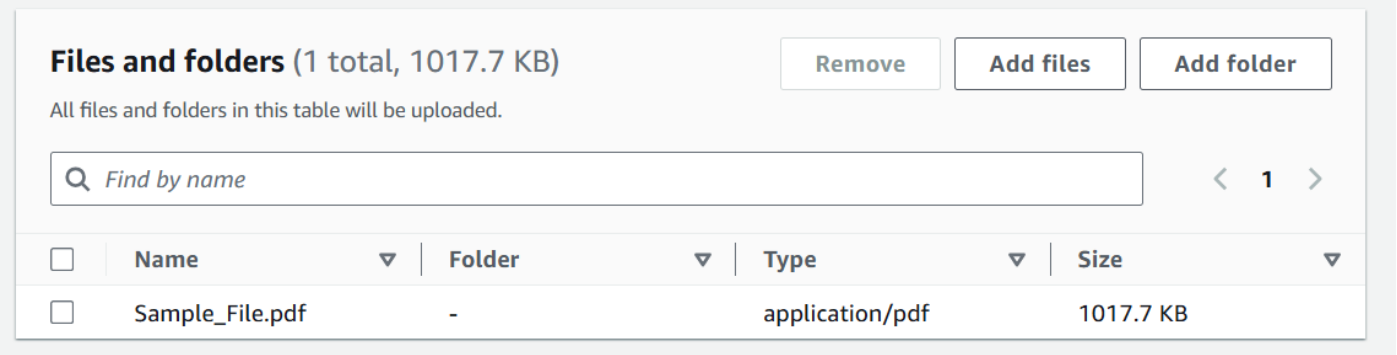


Step 7: Click on upload to add the file to the storage bucket.

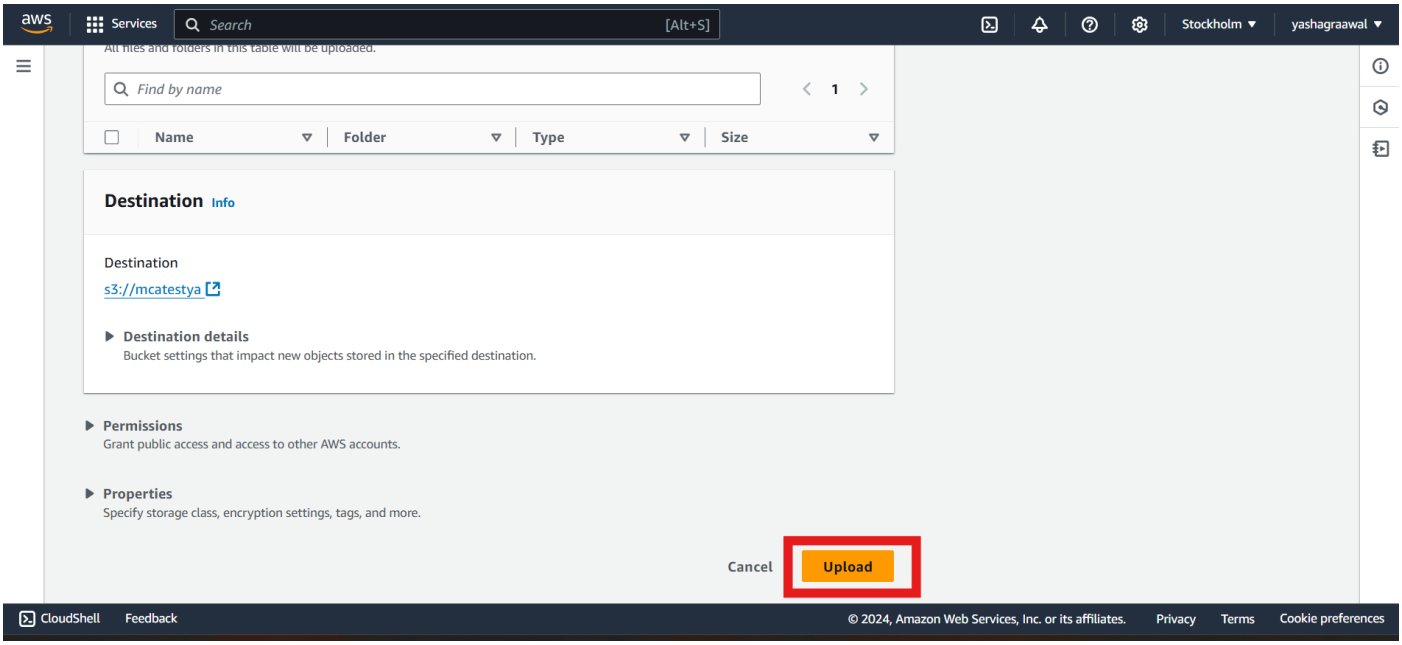


Step 8: Click on Add Files or Folder which needs to be upload, Which will prompt for browsing the file on local machine.

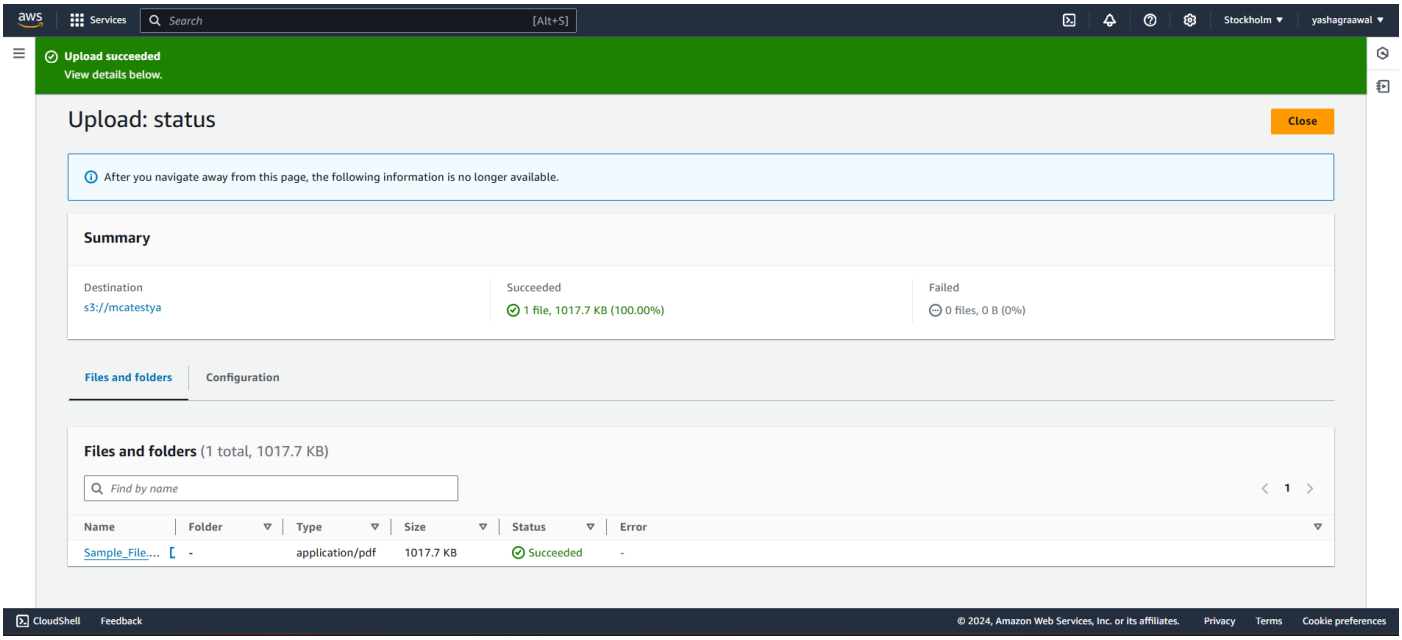




Step 9: Click on the uplad button.

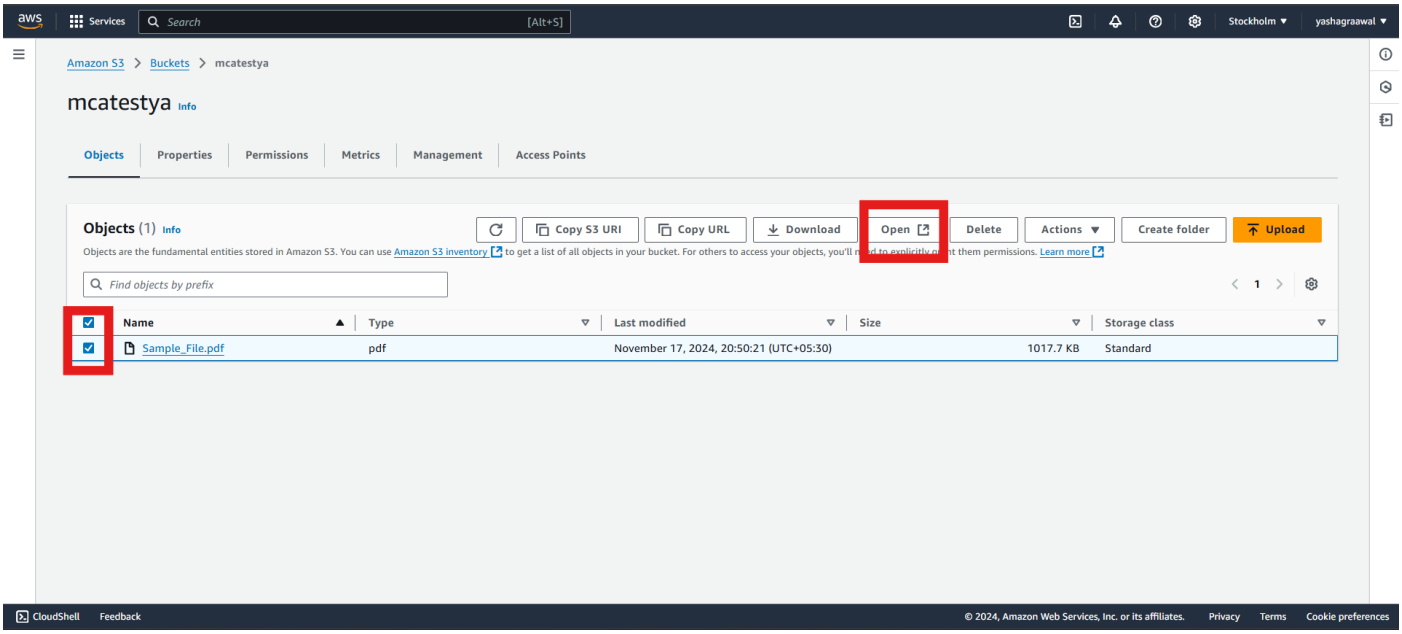


Once Upload successfully you can see this:

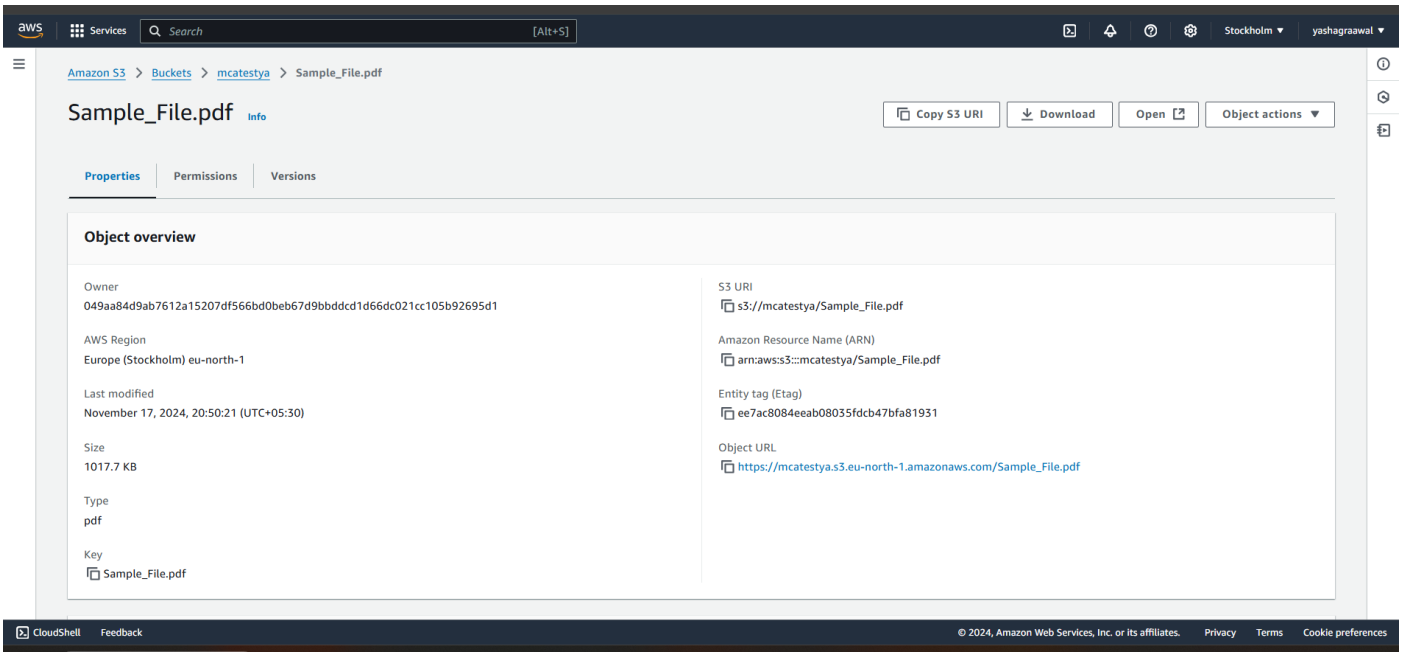


•

Step 10: After file is upload it will be visible in the bucket files list. You can Select the file and click on open to View the file.



Step 11: Click on filename it will show the details on the file where you copy and share the URL of the file where it needs to be accessed.



Note. For File added in S3 bucket to be accessible publicly, the policy needs to be updated.

Practical 7

AIM: Implementation of Identity Access Management (IAM) using AWS Cloud.

Steps to Implement Identify Access Management (IAM) using AWS.

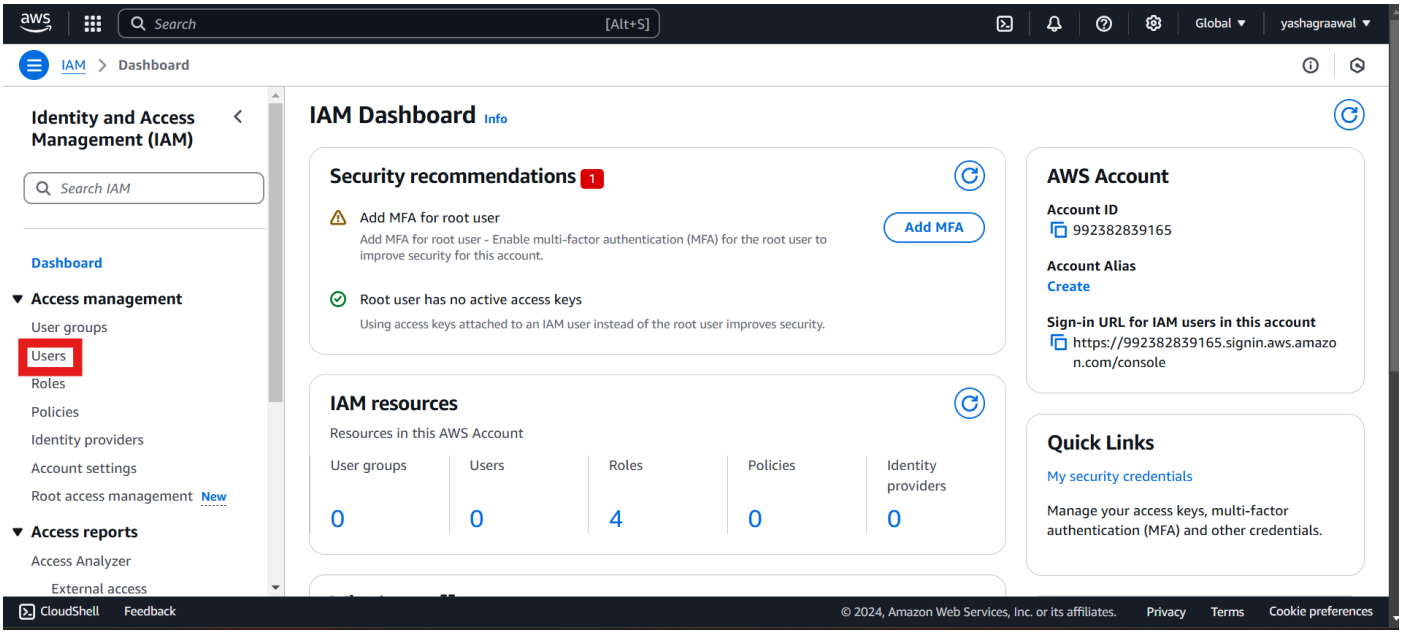
AWS Identity and Access Management (IAM) is a secure and flexible service that helps you manage access to AWS resources. It allows you to control who can perform actions (authentication) and what they can do (authorization) within your AWS environment.

Key features include:

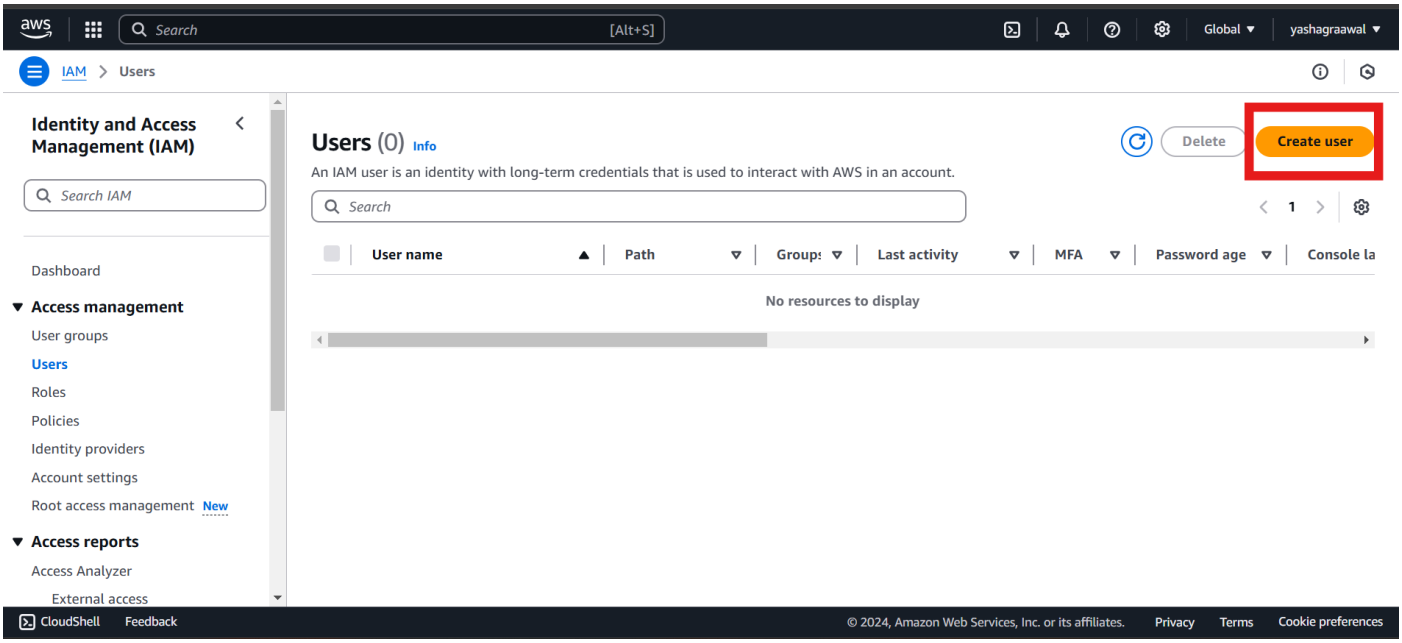
- Fine-Grained Access Control: Define granular permissions for users, groups, and roles to access specific AWS services and resources.
- Secure Authentication: Supports multifactor authentication (MFA) and integration with identity providers for enhanced security.
- Role-Based Access: Use roles to grant temporary access to AWS resources, reducing the need to share long-term credentials.
- Policy Management: Create and attach policies to users, groups, and roles to enforce permissions using JSON-based policy documents.
- Cross-Account Access: Share resources securely across different AWS accounts without needing to duplicate access credentials.

IAM is crucial for maintaining security and governance in AWS, helping organizations implement the principle of least privilege and comply with regulatory requirements.

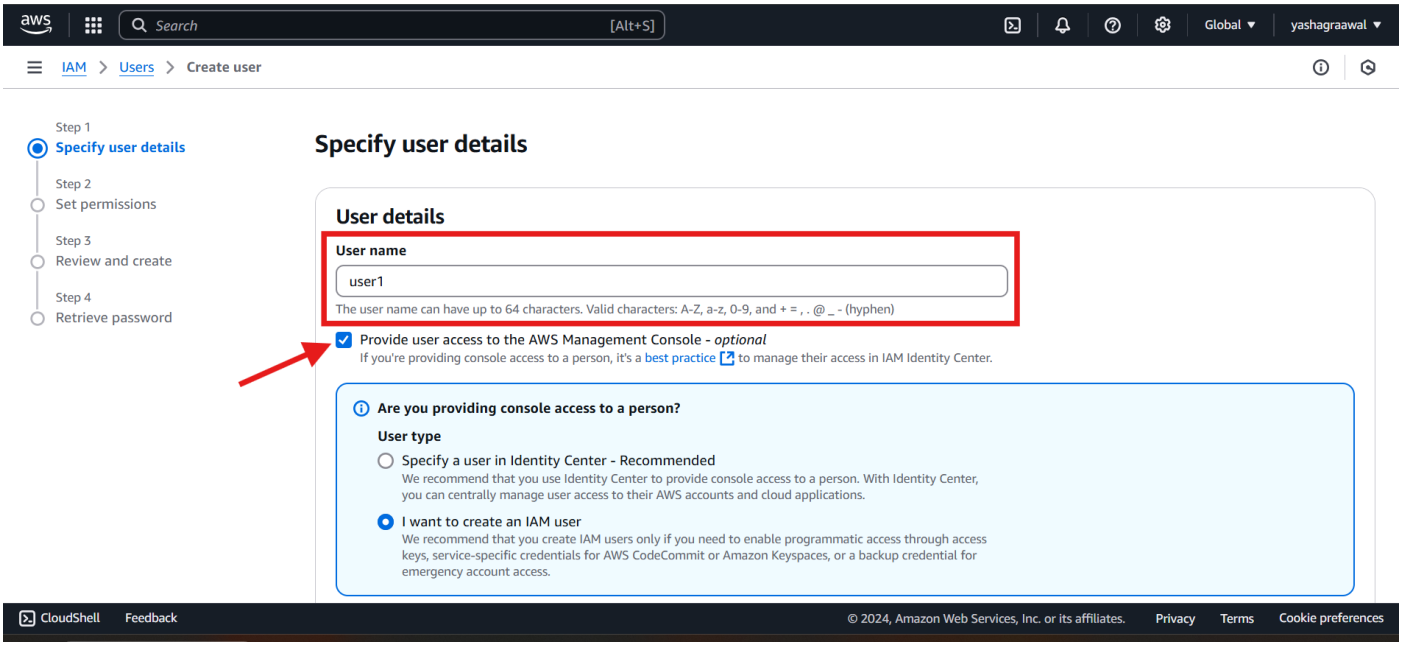
Step 1: Login as Root user to AWS Console and Navigate to IAM Service. Click on Users.



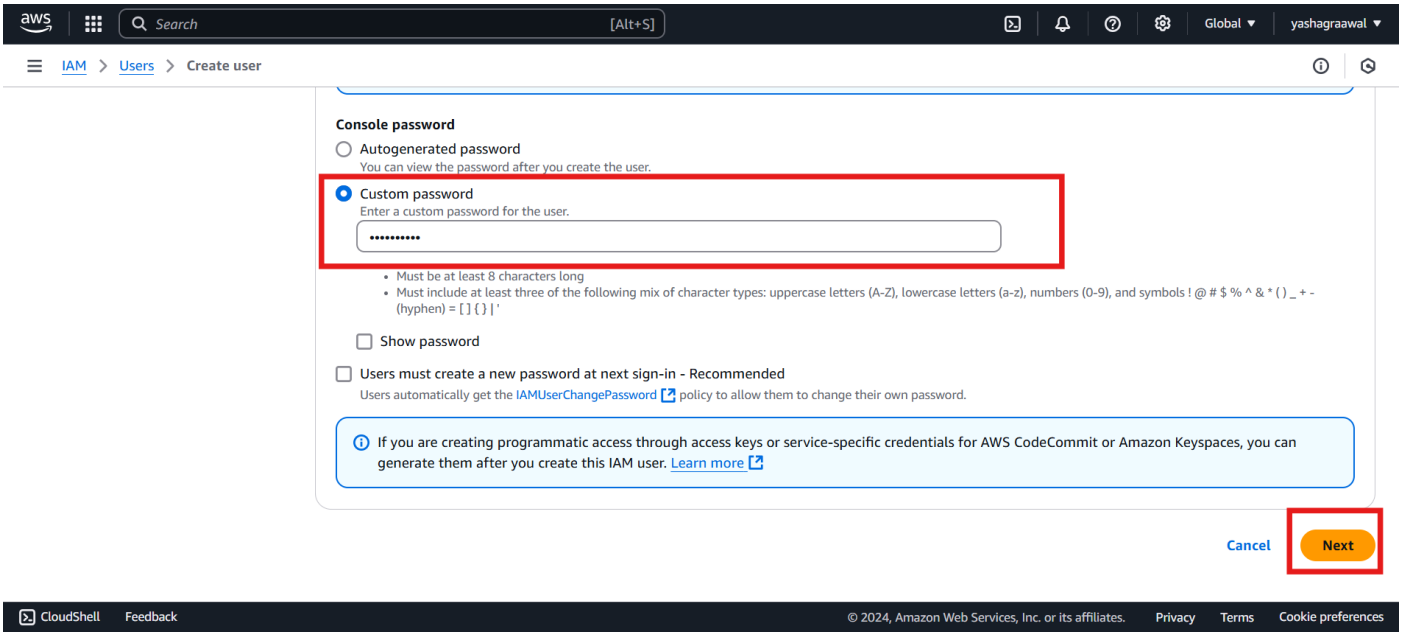
Step 2: Click on Create User Button.



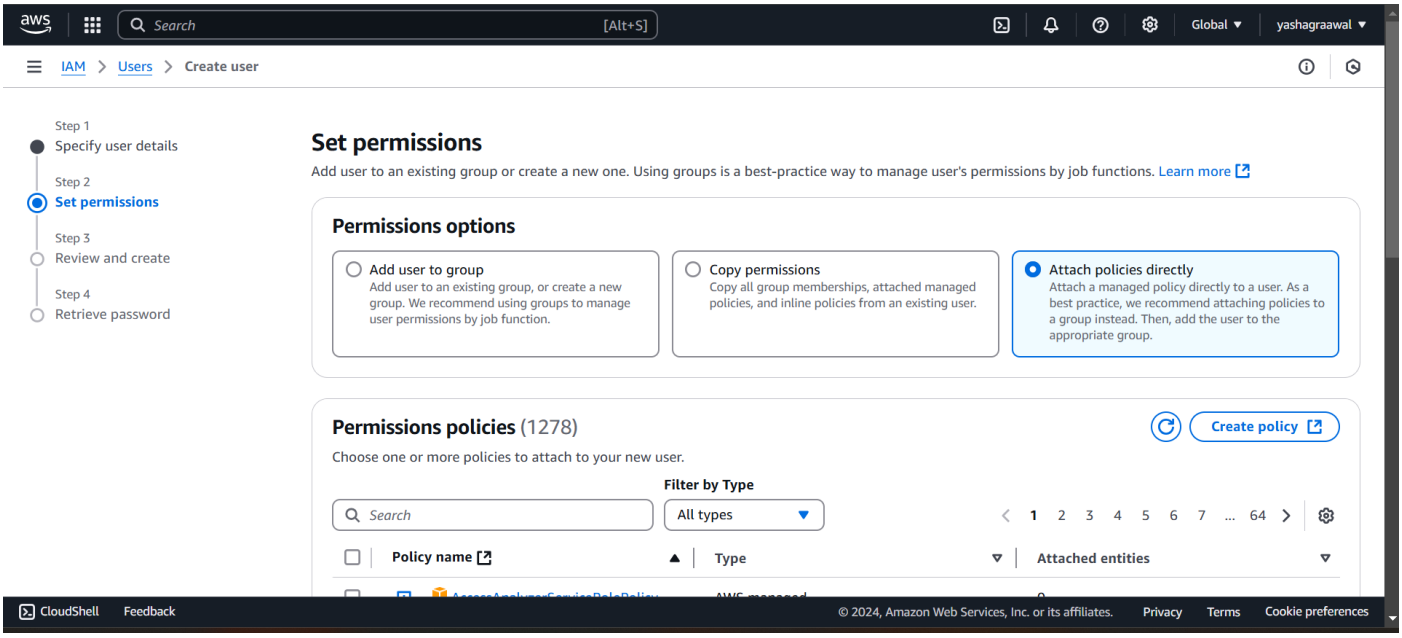
Step 3: Type the Username for the User you are creating and Check the checkbox for password protected Access.



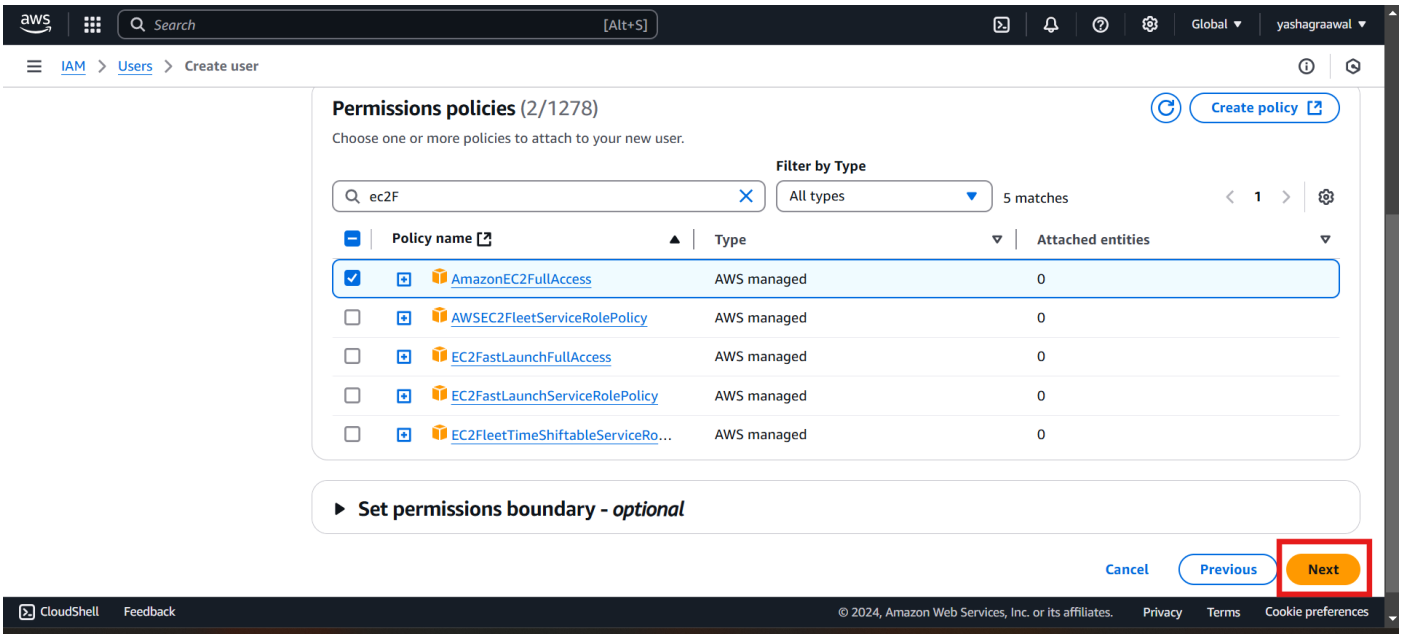
Step 4: Give password for the user and click on Next.



Step 5: Select any of the 3 option as required which allows you to add current user to any existing user group, copy the permissions from other user to current user or give permission to the current user as required.

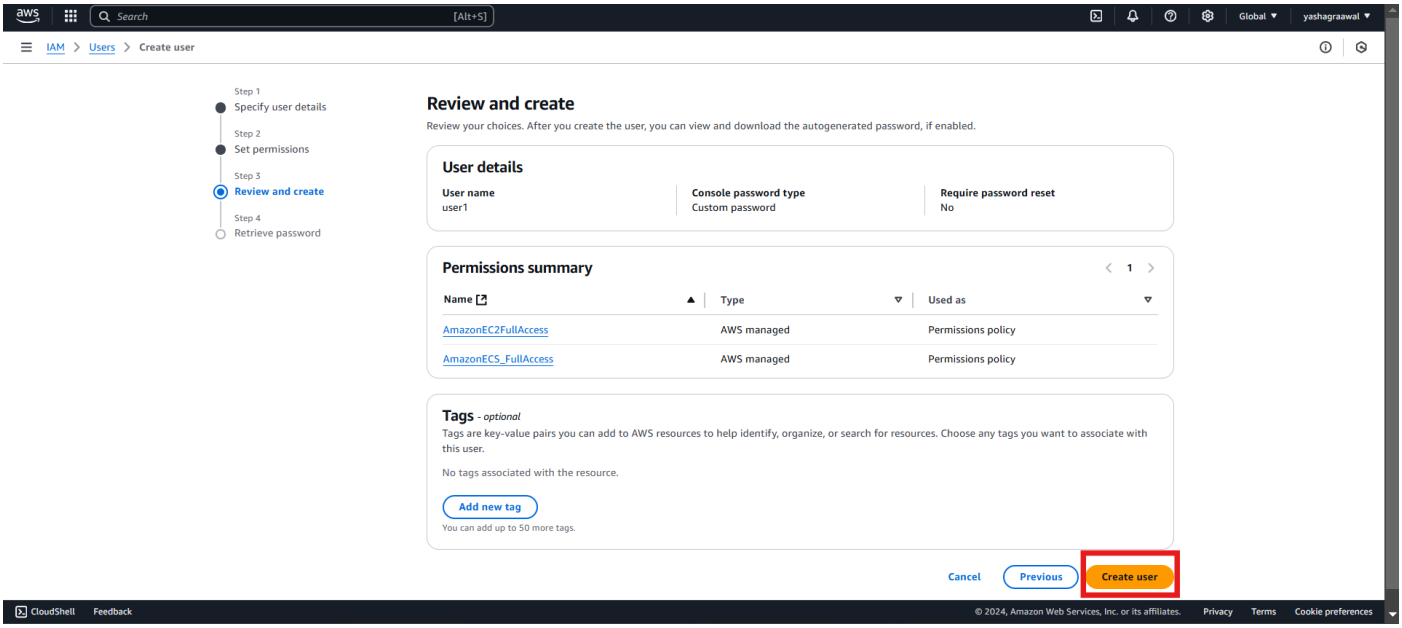


Step 6: Select all the permission you want to give to the user and click on Next. Here we have user access to EC2 full access.

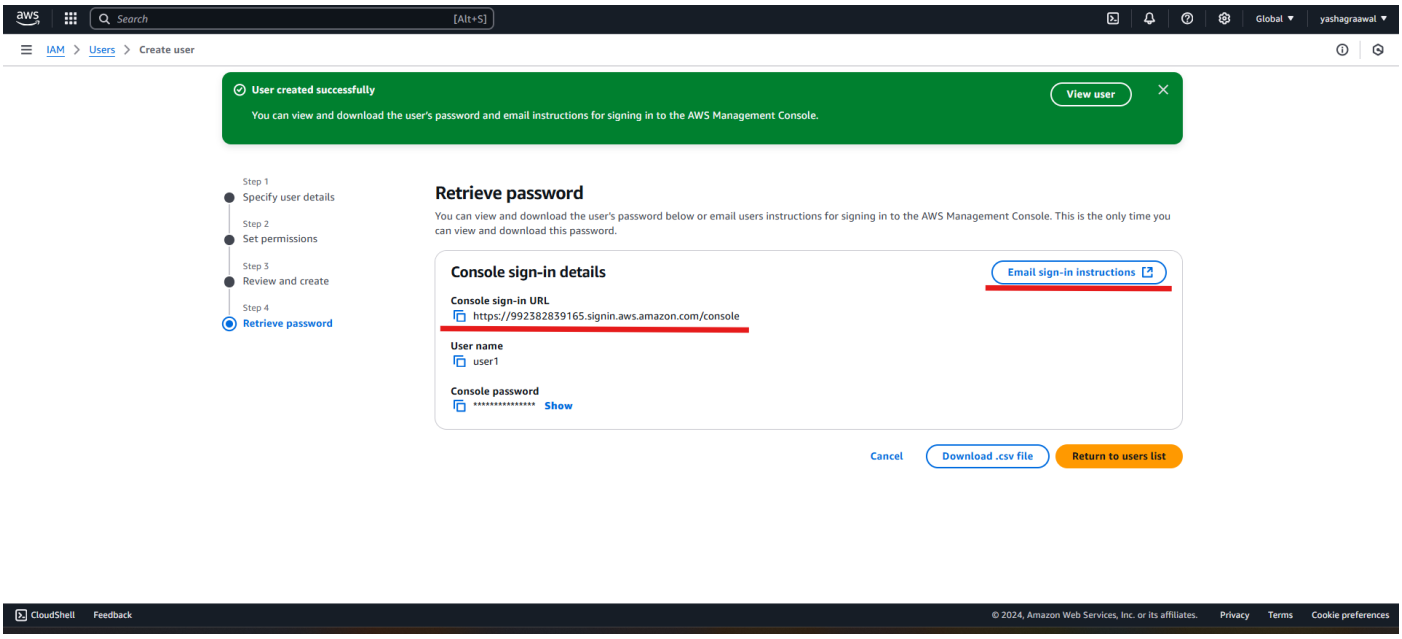


Note: User can still go to any service, but not be able to perform any activity other than service to which user has the permission.

Step 7: Review the permission and click on Create user.



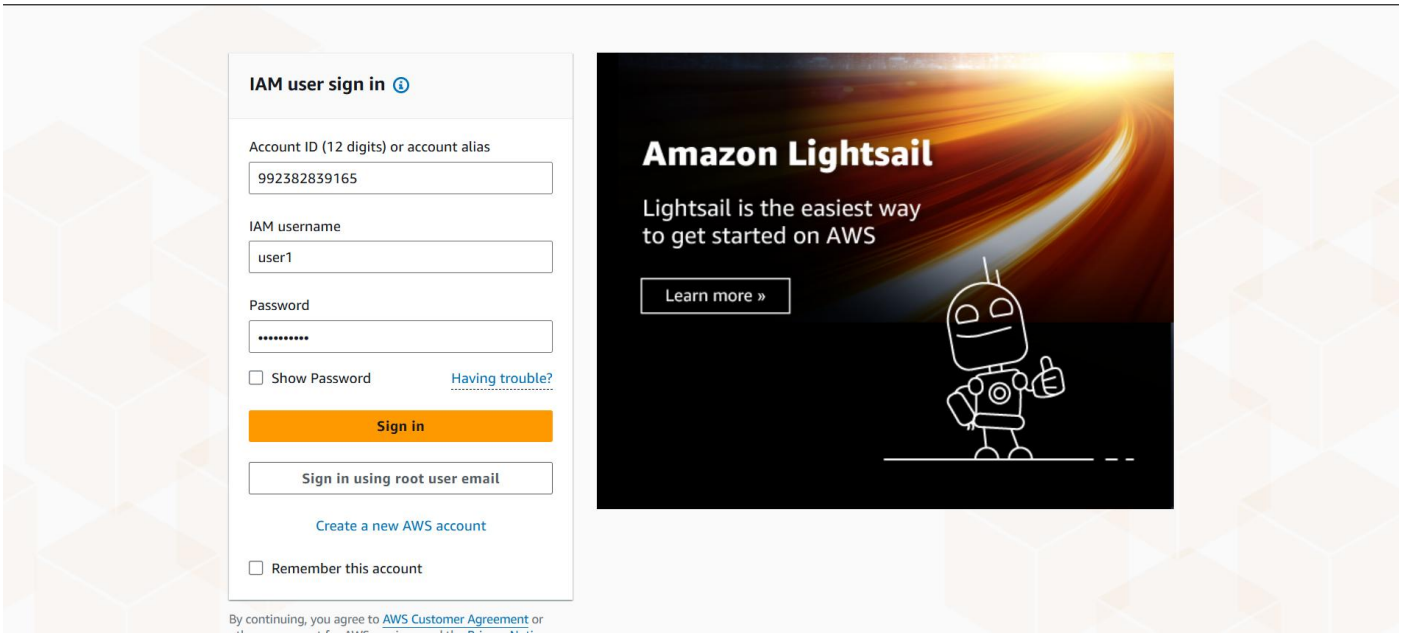
Step 8: User will be created. This screen will provide you with option to copy the link for the user to direct login. Email the user the details and sign-in instructions.



IAM User will require the 12-digit number in URL to Login.

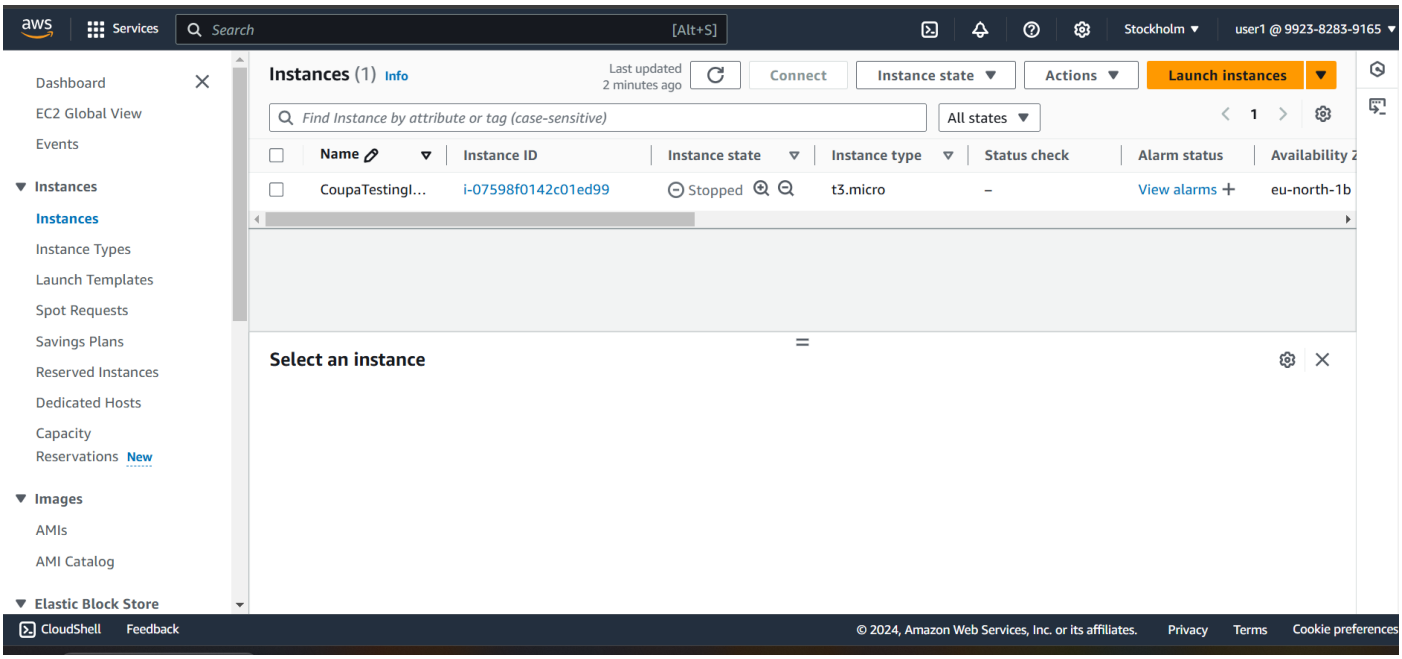
As IAM User:

Step 9: Open the Direct login URL and Enter the login details.

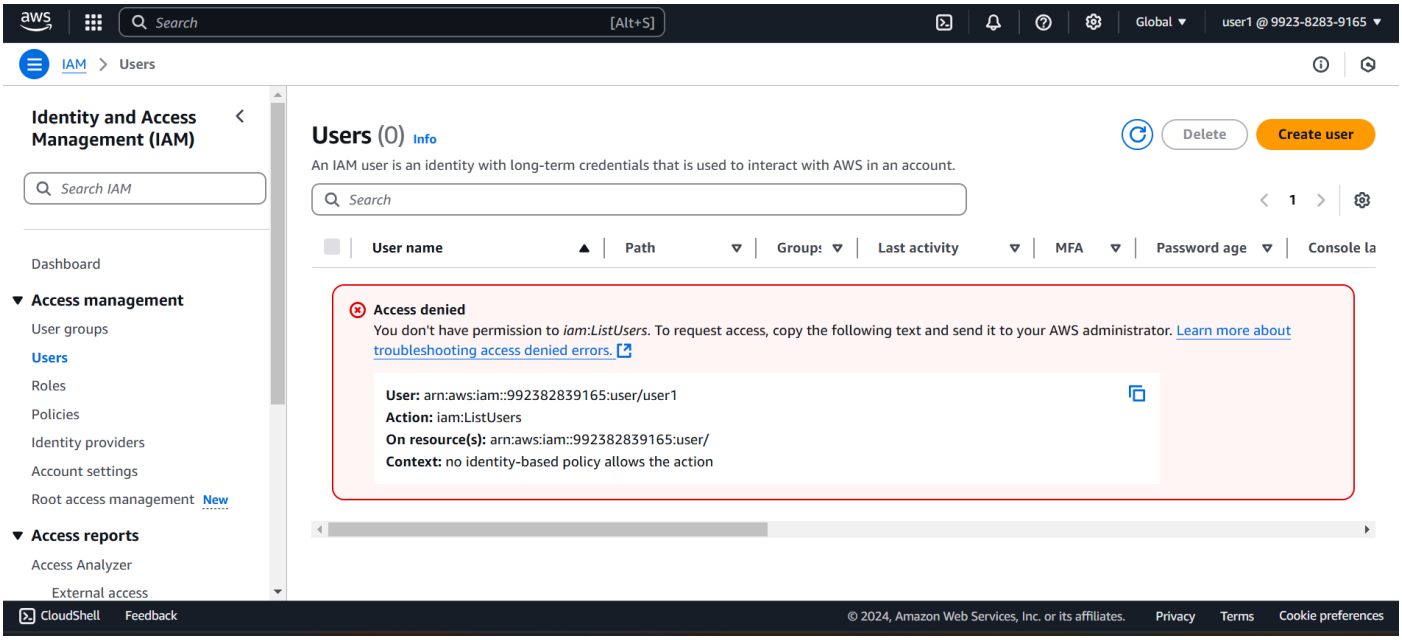


Step 10: After Login since user has EC2 Full Access, Instances from EC2 are visible, And User doesn't have IAM or any other access User will see errors on the services.

EC2:

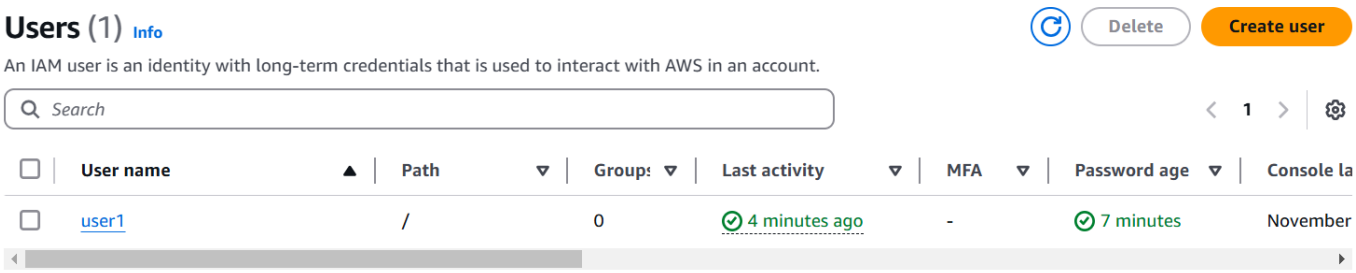


IAM:

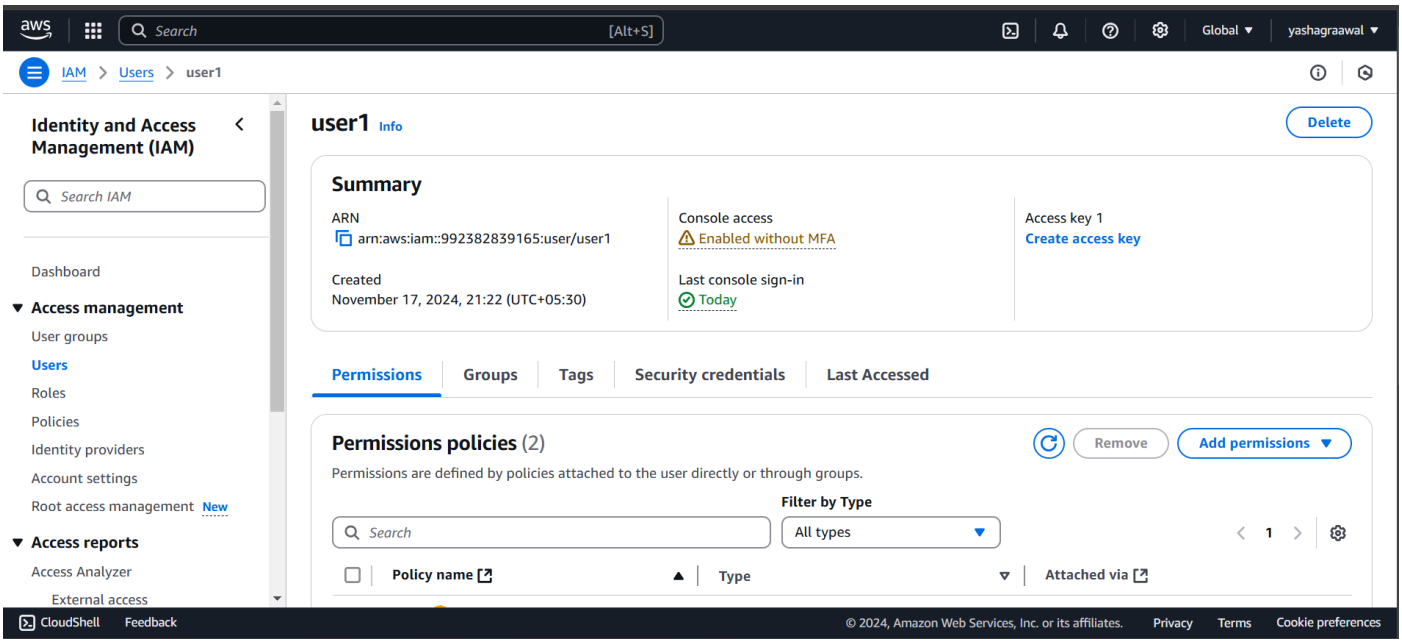


As Root User:

Step 11: Go to Users and you will be able to see the Created User.



Step 12: Open the User and you will be able to edit user details (e.g. Permissions).



Practical 8

AIM: To implement SaaS using some Java online terminal to run program.

Theory:
Software as a Service (SaaS) is a cloud computing model where software applications are delivered over the internet. Instead of installing and maintaining software on individual devices, users access it through a web browser, with the provider managing the infrastructure, updates, and security.
Key Features:

- Accessibility: SaaS applications can be accessed from any device with an internet connection and a web browser.
- Cost-Effective: Operates on a subscription-based model, reducing the need for upfront hardware or software investments.
- Automatic Updates: Providers handle software updates, ensuring users always have access to the latest features and security patches.
- Scalability: Easily scales to accommodate growing user bases or fluctuating workloads without requiring additional infrastructure.
- Integration and Collaboration: Often integrates with other cloud services and offers built-in collaboration tools for teams.

For the SaaS implementation, we will utilize a simple Java Spring Boot application, the Task Management System, which will be deployed on an AWS EC2 instance.

Application Overview:

he Task Management System allows users to efficiently manage their tasks. The application enables users to input a task name and description, storing the information in a database. As tasks are added, they are listed in the application, providing users with a clear overview of their tasks. Additionally, users can delete tasks by clicking the delete button next to the task name in the list.

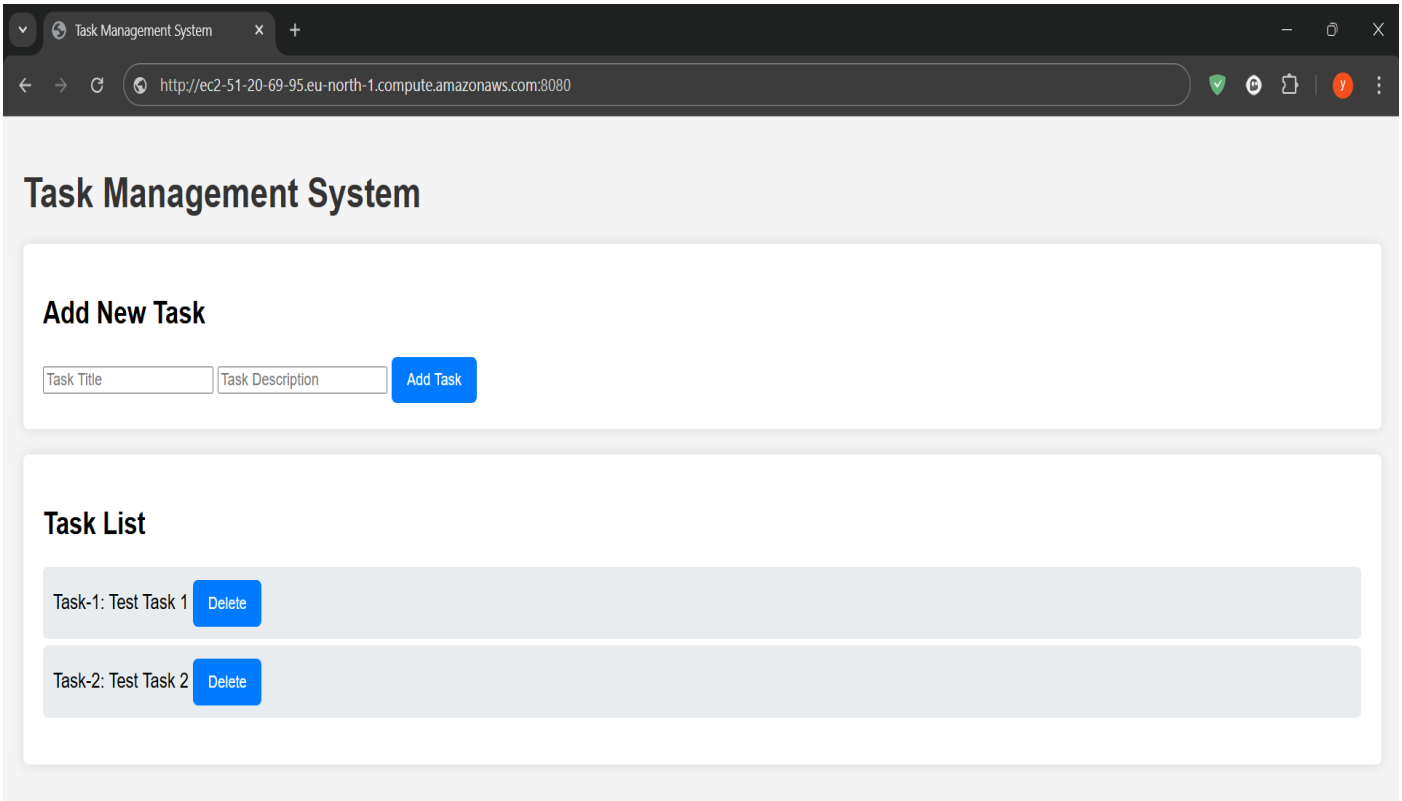
This system is designed to be a basic yet functional demonstration of task management within a SaaS environment.

Steps:

1. The application is built, tested, packaged, and exported as a JAR file using Maven.
2. The resulting JAR file is transferred to an AWS EC2 cloud instance at the following path: /home/user/TMS.
3. The network configuration for the EC2 instance must be updated to allow public traffic on port 8080, which is used by the application.
4. Ensure that Java JDK is installed on the instance before running the application.

5. The JAR file is executed using the following command: java -jar <filename>.jar.
6. The application will start successfully if there are no errors. Any errors must be resolved to ensure proper startup.
7. Once started, the application can be accessed from any client or browser using the URL: http://<Instance IPv4 Address (IP/DNS)>:8080/

Output:



Practical 9

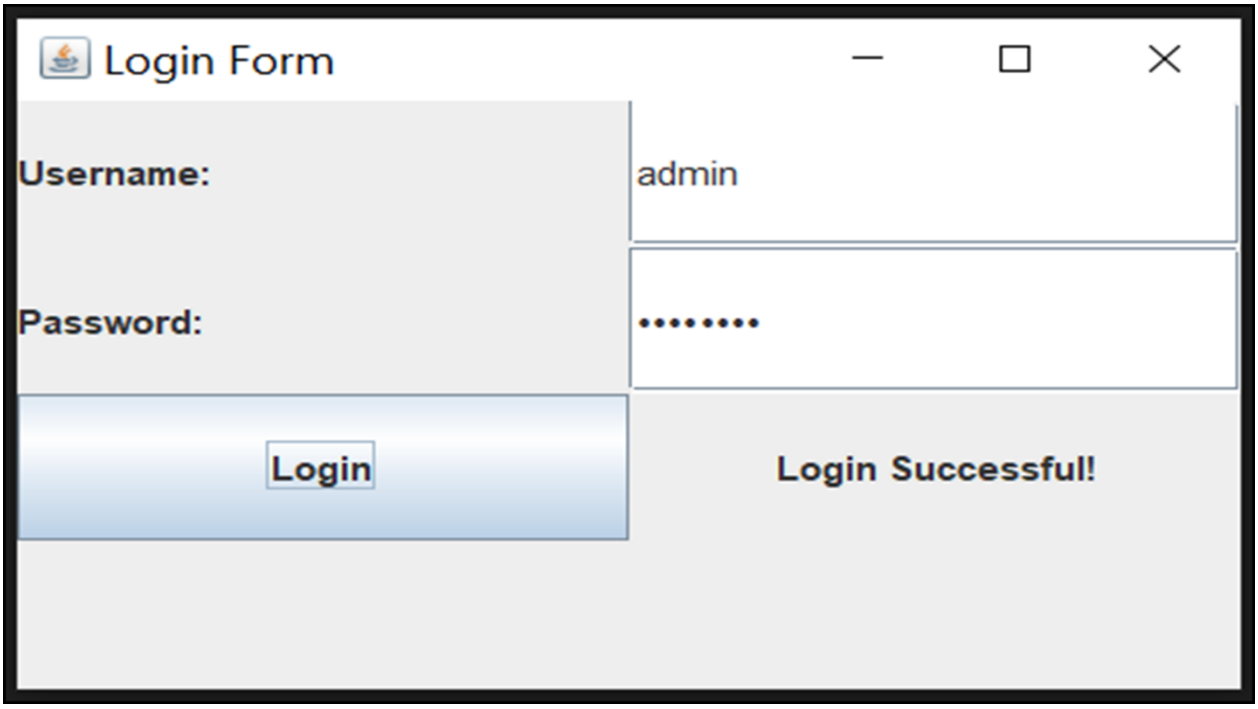
AIM: Write a code to take username as input and dynamically changing the text content of Web Page.

SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoginForm extends JFrame {
   // Components
   private JTextField usernameField;
   private JPasswordField passwordField;
   private JButton loginButton;
   private JLabel messageLabel;

   public LoginForm() {
      // Set up the form
      setTitle("Login Form");
      setSize(400, 250);  // Adjusted size for a more compact form
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLocationRelativeTo(null);
      setLayout(new GridLayout(4, 1));

      // Username Label and Field
      JLabel usernameLabel = new JLabel("Username:");
      usernameField = new JTextField();
      add(usernameLabel);
      add(usernameField);

      // Password Label and Field
      JLabel passwordLabel = new JLabel("Password:");
      passwordField = new JPasswordField();
      add(passwordLabel);
      add(passwordField);

      // Login Button
      loginButton = new JButton("Login");
      add(loginButton);

      // Message Label
      messageLabel = new JLabel("", SwingConstants.CENTER);
      add(messageLabel);

      // Login Button Action
      loginButton.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
```

```
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());

            // Example check for username and password
            if (username.equals("admin") && password.equals("password")) {
                messageLabel.setText("Login Successful!");
            } else {
                messageLabel.setText("Invalid username or password.");
            }
        }
    });
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new LoginForm().setVisible(true);
        }
    });
}
}
```

OUTPUT:



Practical 10

AIM: To demonstrate Web Application Development using NetBeans.

SOURCE CODE:

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Message Submission</title>
  </head>
  <body>
    <h1>Submit Your Message</h1>
    <form action="SubmitMessage" method="post">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" required><br><br>
      <label for="message">Message:</label>
      <textarea id="message" name="message" required></textarea><br><br>
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

display.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Display Message</title>
  </head>
  <body>
    <h1>Message Received</h1>
    <p><strong>Name:</strong> <%= request.getAttribute("name") %></p>
    <p><strong>Message:</strong> <%= request.getAttribute("message") %></p>
    <br>
    <a href="index.jsp">Submit another message</a>
  </body>
</html>
```

SubmitMessage.java

```
package com.simplewebapp;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(name = "SubmitMessage", urlPatterns = {"/SubmitMessage"})
public class SubmitMessage extends HttpServlet {
 @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // Get user inputs
      String name = request.getParameter("name");
      String message = request.getParameter("message");

      // Set attributes to pass data to JSP
      request.setAttribute("name", name);
      request.setAttribute("message", message);

      // Forward to display.jsp
      RequestDispatcher dispatcher = request.getRequestDispatcher("display.jsp");
      dispatcher.forward(request, response);
   }
}
```

OUTPUT: