Roll No.**24** Exam Seat No._____

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg,Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that Mr. **NARENDER KESWANI [ROLL NO: 24]** of **SYMCA-1B** has satisfactorily completed a course of the necessary experiments in **MCAL32 - Distributed System and Cloud Computing Lab** under the supervision of **Mr. Sunny Nahar** in the Institute of Technology in the academic year **2022- 2023.**

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony, Chembur, Mumbai

Department of M.C.A

MCAL32 - DISTRIBUTED SYSTEM AND CLOUD COMPUTING LAB INDEX

Sr. No	Contents	Date Of Preparation	Date Of Submission	Marks	Sign
1	Remote Process Communication.	26/08/2022	29/08/2022	10	
2	Remote Procedure call using datagram (Calculator)	26/08/2022	29/08/2022	10	
3	Remote Procedure call using datagram (Date and Time)	29/08/2022	07/09/2022	10	
4	Remote Method Invocation (Stubs & Skeletons)	07/09/2022	16/09/2022	10	
5	Remote Method Invocation (Equation Solver)	16/09/2022	23/09/2022	10	
6	Implementation of Remote Method Communication using JDBC and RMI.	23/09/2022	30/09/2022	10	
7	Implementation of mutual exclusion using the Token ring algorithm.	30/09/2022	07/10/2022	10	
8	Implementation of Storage as a Service using Google Docs.	07/10/2022	14/10/2022	10	
9	Application using Google App Engine.	14/10/2022	21/10/2022	10	
10	Identity Access Management in Amazon Web Services.	28/10/2022	04/11/2022	10	

Remote Process Communication

AIM: To develop a multi-client chat server application where multiple clients chat with each other concurrently, where the messages sent by different clients are first communicated to the server and then the server, on behalf of the source client, communicates the messages to the appropriate destination client

THEORY:

Remote Procedure Call:

• Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server-based applications.

• It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure.

• The two processes may be on the same system, or they may be on different systems with a network connecting them.

• When making a Remote Procedure Call:

1) The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.

2) When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from execution resumes as if returning from a regular procedure call.

java.net.ServerSocket Class:

• ServerSocket Class is used for providing system-independent implementation of the server-side of a client/server Socket Connection.

• The constructor for ServerSocket throws an exception if it can't listen on the specified port.

java.net.Socket Class:

• This class represents the socket that both the client and the server use to communicate with each other.

• The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

SOURCE CODE:

Client.java:

```
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

/**
 * @author NARENDER KESWANI
 */
public class Client {
```

```
    public static void main(String[] args) {
        String name = "empty";
        String reply = "empty";
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter your name (Please enter your name to join the chat): ");
        reply = sc.nextLine();
        name = reply;
        try (Socket socket = new Socket("localhost", 5000)) {
            PrintWriter cout = new PrintWriter(socket.getOutputStream(), true);
            ThreadClient threadClient = new ThreadClient(socket);
            new Thread((Runnable) threadClient).start(); // start thread to receive message
            cout.println(reply + ": has joined chat-room.");
            do {
                String message = (name + " : ");
                reply = sc.nextLine();
                if (reply.equals("logout")) {
                    cout.println("logout");
                    break;
                }
                cout.println(message + reply);
            } while (!reply.equals("logout"));
        } catch (Exception e) {
            System.out.println(e.getStackTrace());
        }
    }
}
```

Server.java:

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * @author NARENDER KESWANI
 */
public class Server {

    public static void main(String[] args) {
```

```
        ArrayList<Socket> clients = new ArrayList<>();
        HashMap<Socket, String> clientNameList = new HashMap<Socket, String>();
        try (ServerSocket serversocket = new ServerSocket(5000)) {
            System.out.println("Server is started...");
            while (true) {
                Socket socket = serversocket.accept();
                clients.add(socket);
                ThreadServer ThreadServer = new ThreadServer(socket, clients, clientNameList);
                ThreadServer.start();
            }
        } catch (Exception e) {
            System.out.println(e.getStackTrace());
        }
    }
}
```

ThreadClient.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.SocketException;

public class ThreadClient implements Runnable {

    private Socket socket;
    private BufferedReader cin;

    public ThreadClient(Socket socket) throws IOException {
        this.socket = socket;
        this.cin = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }

    @Override
    public void run() {
        try {
            while (true) {
                String message = cin.readLine();
                System.out.println(message);
            }
```

```
        } catch (SocketException e) {
            System.out.println("You left the chat-room");
        } catch (IOException exception) {
            System.out.println(exception);
        } finally {
            try {
                cin.close();
            } catch (Exception exception) {
                System.out.println(exception);
            }
        }
    }
}
```

ThreadServer.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.SocketException;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * @author NARENDER KESWANI
 */
public class ThreadServer extends Thread {

    private Socket socket;
    private ArrayList<Socket> clients;
    private HashMap<Socket, String> clientNameList;

    public ThreadServer(Socket socket, ArrayList<Socket> clients, HashMap<Socket, String>
clientNameList) {
        this.socket = socket;
        this.clients = clients;
        this.clientNameList = clientNameList;
    }

    @Override
    public void run() {
        try {
```

```
        BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        while (true) {
            String outputString = input.readLine();
            if (outputString.equals("logout")) {
                throw new SocketException();
            }
            if (!clientNameList.containsKey(socket)) {
                String[] messageString = outputString.split(":", 2);
                clientNameList.put(socket, messageString[0]);
                System.out.println(messageString[0] + messageString[1]);
                showMessageToAllClients(socket, messageString[0]
                        + messageString[1]);
            } else {
                System.out.println(outputString);
                showMessageToAllClients(socket, outputString);
            }
        }
    } catch (SocketException e) {
        String printMessage = clientNameList.get(socket) + " left the chat room";
        System.out.println(printMessage);
        showMessageToAllClients(socket, printMessage);
        clients.remove(socket);
        clientNameList.remove(socket);
    } catch (Exception e) {
        System.out.println(e.getStackTrace());
    }
}

private void showMessageToAllClients(Socket sender, String outputString) {
    Socket socket;
    PrintWriter printWriter;
    int i = 0;
    while (i < clients.size()) {
        socket = clients.get(i);
        i++;
        try {
            if (socket != sender) {
                printWriter = new PrintWriter(socket.getOutputStream(), true);
                printWriter.println(outputString);
            }
        } catch (IOException ex) {
            System.out.println(ex);
        }
```

```
    }
  }
}
```

OUTPUT:

Server:

```
run:
Server is started...
```

Client 1 [Narender]:

```
run:
Enter your name (Please enter your name to join the chat):
narender
```

Client 2 [Neel]:

```
Enter your name (Please enter your name to join the chat):
neel
```

Client 3 [Wilson]:

```
Enter your name (Please enter your name to join the chat):
wilson
```

COMMUNICATION:

```
Server is started...
narender has joined chat-room.
neel has joined chat-room.
wilson has joined chat-room.
narender : hi sir
neel : hi sir
wilson : what is the status of the project
narender : backend part is done, now we are focusing on ui
neel : for hosting we are planing to go with aws
wilson : good
```

CONCLUSION:

We are able to develop a multi-client chat server application where multiple clients chat with each other concurrently, where the messages sent by different clients are first communicated to the server and then the server, on behalf of the source client, communicates the messages to the appropriate destination client.

Remote Procedure call using Datagram

Aim: To implement a Server calculator containing ADD (), MUL (), SUB (), DIV (), etc

THEORY:

In distributed computing, RPC is when a computer program causes a procedure to execute in a different address space, which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction. This is a form of client–server interaction, typically implemented via a request–response message-passing system. RPC is an interprocess communication technique that is used for client-server-based applications. A client has a request message that the RPC translates and sends
to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.
A common pattern of communication used by application programs structured as a client/server pair isthe request/reply message transaction: A client sends a request message to a server, and the server responds with a reply message, with the client blocking to wait for the reply. A transport protocol that supports the request/reply paradigm is much more than a UDP message going in one direction followed by a UDP message going in the other direction. It needs to deal with correctly identifying processes on remote hosts and correlating requests with responses. While TCP overcomes these limitations by providing a reliable byte-stream service, it doesn't perfectly match the request/reply paradigm either. This section describes a third category of transport protocol, called RPC, that more closely matches the needs of an application involved in a request/reply message exchange.

The sequence of events in a remote procedure call are given as follows:
1. The client stub is called by the client.
2. The client stub makes a system call to send the message to the server and puts the parameters in the message.
3. The message is sent from the client to the server by the client's operating system.
4. The message is passed to the server stub by the server operating system.
5. The parameters are removed from the message by the server stub.
6. Then, the server procedure is called by the server stub.

Datagram
A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. Datagrams plays a vital role as an alternative. They are bundles of information passed between machines. Once the datagram has been released to its intended target, there is no assurance that it will arrive or even that someone will

be there to catch it. Likewise, when the datagram is received, there is no assurance that it hasn't been damaged in transit or that whoever sent it is still there to receive a response and it is crucial point to note. A datagram is a basic transfer unit associated with a packet-switched network. Datagrams are typically structured in header and payload sections. Datagrams provide a connectionless communication service across a packet-switched network. The delivery, arrival time, and order of arrival of datagrams need not be guaranteed by the network.

Java implements datagrams on top of the UDP (User Datagram Protocol) protocol by using two classes:

• DatagramPacket object is the data container.
• DatagramSocket is the mechanism used to send or receive the DatagramPackets.

DatagramSocket defines four public constructors. They are shown here:
• DatagramSocket () throws SocketException
• DatagramSocket (int port) throws SocketException
• DatagramSocket (int port, InetAddress ipAddress) throws SocketException
• DatagramSocket (SocketAddress address) throws SocketException

SocketAddress is an abstract class that is implemented by the concrete class InetSocketAddress. InetSocketAddress encapsulates an IP address with a port number. All can throw a SocketException if an error occurs while creating the socket. DatagramSocket defines many methods. Two of the most important are send () and receive (), which are shown here:
• void send (DatagramPacket packet) throws IOException
• void receive (DatagramPacket packet) throws IOException

The send () method sends packet to the port specified by packet. The receive method waits for a packet to be received from the port specified by packet and returns the result.

Client-Side Programming
1. Establish a Socket Connection
To connect to another machine, we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port. The java.net.Socket class represents a Socket. To open a socket:
Socket socket = new Socket("127.0.0.1", 5000)
First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).
Second argument – TCP Port. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

2. Communication
To communicate over a socket connection, streams are used to both input and output the data.

3. Closing the connection

The socket connection is closed explicitly once the message to server is sent

Server Programming
1. Establish a Socket Connection
To write a server application two sockets are needed.
➢ A ServerSocket which waits for the client requests (when a client makes a new Socket())
➢ A plain old Socket socket to use for communication with the client.
2. Communication
getOutputStream() method is used to send the output through the socket.
3. Close the Connection
After finishing, it is important to close the connection by closing the socket as well as input/output streams.

SOURCE CODE:

Client.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class Client {

    public static void main(String args[])
            throws IOException {
        Scanner sc = new Scanner(System.in);
        DatagramSocket ds = new DatagramSocket();
        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;
        while (true) {
            System.out.print("Enter the equation in the format:");
            String inp = sc.nextLine();
            buf = new byte[65535];
            buf = inp.getBytes();
            DatagramPacket DpSend = new DatagramPacket(buf, buf.length, ip, 1234);
            ds.send(DpSend);
            if (inp.equals("bye")) {
                break;
            }
```

```
            buf = new byte[65535];
            DatagramPacket DpReceive = new DatagramPacket(buf, buf.length);
            ds.receive(DpReceive);
            System.out.println("Answer = " + new String(buf, 0, buf.length));
        }
    }
}
```

Server.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.StringTokenizer;

class Server {

    public static void main(String[] args)
            throws IOException {
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] buf = null;
        DatagramPacket DpReceive = null;
        DatagramPacket DpSend = null;
        while (true) {
            buf = new byte[65535];
            DpReceive = new DatagramPacket(buf, buf.length);
            ds.receive(DpReceive);
            String inp = new String(buf, 0, buf.length);
            inp = inp.trim();
            System.out.println("Equation Received:- " + inp);
            if (inp.equals("close")) {
                System.out.println("Client sent closing");
                break;
            }
            int result;
            StringTokenizer st = new StringTokenizer(inp);
            int oprnd1 = Integer.parseInt(st.nextToken());
            String operation = st.nextToken();
            int oprnd2 = Integer.parseInt(st.nextToken());
```

```
        if (operation.equals("+")) {
            result = oprnd1 + oprnd2;
        } else if (operation.equals("-")) {
            result = oprnd1 - oprnd2;
        } else if (operation.equals("*")) {
            result = oprnd1 * oprnd2;
        } else {
            result = oprnd1 / oprnd2;
        }
        System.out.println("Sending the result...");
        String res = Integer.toString(result);
        buf = res.getBytes();
        int port = DpReceive.getPort();
        DpSend = new DatagramPacket(
                buf, buf.length, InetAddress.getLocalHost(), port);
        ds.send(DpSend);
      }
    }
}
```

OUTPUT:

Client:

```
run:
Enter the equation in the format:5 + 15
Answer = 20
Enter the equation in the format:25 - 5
Answer = 20
Enter the equation in the format:15 / 5
Answer = 3
Enter the equation in the format:10 * 10
Answer = 100
```

Server:

```
run:
Equation Received:- 5 + 15
Sending the result...
Equation Received:- 25 - 5
Sending the result...
Equation Received:- 15 / 5
Sending the result...
Equation Received:- 10 * 10
Sending the result...
```

CONCLUSION:

Hence, we have successfully implemented a Server calculator containing ADD (), MUL (), SUB () & DIV () using the RPC concept.

Remote Procedure call using Datagram

Aim: To implement a Date Time Server containing date () and time ().

SOURCE CODE:

Client.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.io.*;
import java.net.*;

public class Client {

   Client() {
      try {

         InetAddress ia = InetAddress.getLocalHost();
         DatagramSocket ds = new DatagramSocket();
         byte b1[] = new byte[50];
         DatagramSocket ds1 = new DatagramSocket(1300);
         System.out.println("\nRPC Client\n");
         while (true) {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String str = br.readLine();
            byte b[] = str.getBytes();
            DatagramPacket dp = new DatagramPacket(b, b.length, ia, 1200);
            ds.send(dp);
            dp = new DatagramPacket(b1, b1.length);
            ds1.receive(dp);
            String s = new String(dp.getData(), 0, dp.getLength());
            System.out.println("\nResult = " + s);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

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

Server.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.util.*;
import java.net.*;
import java.text.SimpleDateFormat;

public class Server {

   DatagramSocket ds;
   DatagramPacket dp;
   String str, methodName, result;
   int val1, val2;

   Server() {
      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;
               }
            }
            Calendar c = Calendar.getInstance();
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
            Date d = c.getTime();
            InetAddress ia = InetAddress.getLocalHost();
            if (methodName.equalsIgnoreCase("date")) {
               result = "" + dateFormat.format(d);
            } else if (methodName.equalsIgnoreCase("time")) {
               result = "" + d.getHours() + " : " + d.getMinutes() + " : " + d.getSeconds();
            }
```

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

OUTPUT:

SERVER:

```
C:\Users\NARENDER KESWANI>cd C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src

C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src>javac *.java
Note: Server.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src>start rmiregistry
```

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src>java Server
result : 23/09/2022

result : 0 : 35 : 43
```

CLIENT:

```
C:\Users\NARENDER KESWANI>cd C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src

C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP3\src>java Client

RPC Client

date

Result = 23/09/2022
time

Result = 0 : 35 : 43
```

CONCLUSION:

Hence, we have successfully implemented a Date Time Server using RPC Datagram.

Remote Method Invocation (Stubs & Skeletons)

Aim: To find date and time using Remote Method Invocation (Use stubs and skeletons)

RMI (Remote Method Invocation):
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton. RMI uses stub and skeleton object for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

Stub:
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine(JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton:
The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

The 6 steps to write
1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

SOURCE CODE:

DateTime.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
public interface DateTime extends Remote {
    public String date(int op) throws RemoteException;
}
```

DateTimeRemote.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.text.SimpleDateFormat;

public class DateTimeRemote extends UnicastRemoteObject implements DateTime {

    DateTimeRemote() throws RemoteException {
        super();
    }
    @Override
    public String date(int op) {
        Date dateObj = new Date();
        if (op == 1) {
            SimpleDateFormat df = new SimpleDateFormat("MM/dd/YYYY");
            String formattedDate = df.format(dateObj);
            return formattedDate;
        } else if (op == 2) {
            SimpleDateFormat df = new SimpleDateFormat("hh:mm:ss");
            String formattedTime = df.format(dateObj);
            return formattedTime;
        }
        return "a";
    }
}
```

MyClient.java:

```
/**
 * @author NARENDER KESWANI
 */
```

```
import java.rmi.*;

public class MyClient {

   public static void main(String args[]) {
      try {
         DateTime stub=(DateTime)Naming.lookup("rmi://localhost:5000/narender");
         System.out.println("Current Date: " + stub.date(1) + "\n" + "Current Time:" +
stub.date(2));
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

MyServer.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.rmi.registry.*;

public class MyServer {

   public static void main(String args[]) {
      try {
         DateTime stub = new DateTimeRemote();
         Naming.rebind("rmi://localhost:5000/narender",stub);
      } catch (Exception e) {
         System.out.print("exception on server");
         System.out.println(e);
      }
   }
}
```

OUTPUT:

For running this rmi example,
1) compile all the java files
javac *.java

```
C:\Users\NARENDER KESWANI>cd C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src

C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src>javac *.java
```

2)create stub and skeleton object by rmic tool
rmic AdderRemote

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src>rmic DateTimeRemote
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

3)start rmi registry in one command prompt
rmiregistry 5000

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src>rmiregistry 5000
```

4)start the server in another command prompt
java MyServer

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src>java MyServer
```

5)start the client application in another command prompt
java MyClient

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP4\src>java MyClient
Current Date: 09/23/2022
Current Time:02:51:04
```

CONCLUSION:

Hence we successfully calculated date and time using Remote Method Invocation.

Remote Method Invocation (Stubs & Skeletons)

Aim: Implementation of equation solver using Remote Method Invocation (RMI).

RMI (Remote Method Invocation):

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton. RMI uses stub and skeleton object for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

Stub:

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine(JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton:

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

The 6 steps to write

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

SOURCE CODE:

EqualtionSolverInterface.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.util.ArrayList;

public interface EquationSolverInterface extends Remote {
   public int EvaluateEquation(String equation, ArrayList<Integer> params) throws
RemoteException;
}
```

EquationSolverClass.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

public class EquationSolverClass extends UnicastRemoteObject implements
      EquationSolverInterface {

   EquationSolverClass() throws RemoteException {
      super();
   }

   @Override
   public int EvaluateEquation(String equation, ArrayList<Integer> params) {
      System.out.print("Equation recieved: " + equation);
      int a = 0, b = 0, c = 0;
      HashMap<String, Integer> map = new HashMap<>();
      if (params.size() == 2) {
         params.add(c);
      }
      map.put("a^2-b^2", (params.get(0) - params.get(1)) * (params.get(0) + params.get(1)));
      map.put("a^2+b^2", (int) (Math.pow((params.get(0) - params.get(1)), 2)
            + 2 * params.get(0) * params.get(1)));

      map.put("(a+b)^2", (int) (Math.pow(params.get(0), 2) + 2 * params.get(0) * params.get(1)
            + Math.pow(params.get(1), 2)));

      map.put("(a-b)^2", (int) (Math.pow(params.get(0), 2) - 2 * params.get(0) * params.get(1)
            + Math.pow(params.get(1), 2)));
```

```
        map.put("(a+b+c)^2", (int) (Math.pow(params.get(0), 2) + Math.pow(params.get(1), 2)
                + Math.pow(params.get(2), 2) + 2 * params.get(0) * params.get(1)
                + 2 * params.get(1) * params.get(2) + 2 * params.get(0) * params.get(2)));

        map.put("(a-b-c)^2", (int) (Math.pow(params.get(0), 2) + Math.pow(params.get(1), 2)
                + Math.pow(params.get(2), 2) - 2 * params.get(0) * params.get(1) + 2 * params.get(1)
* params.get(2)
                - 2 * params.get(0) * params.get(1)));

        map.put("a^3-b^3", (int) ((params.get(0) - params.get(1))
                * (Math.pow(params.get(0), 2) + params.get(0) * params.get(1) +
Math.pow(params.get(1), 2))));

        map.put("a^3+b^3", (int) ((params.get(0) + params.get(1)) * (Math.pow(params.get(0), 2)
                - params.get(0) * params.get(1) + Math.pow(params.get(1), 2))));

        map.put("(a+b)^3", (int) (Math.pow(params.get(0), 3) + 3 * Math.pow(params.get(0), 2)
                * params.get(1) + 3 * params.get(0) * Math.pow(params.get(1), 2) +
Math.pow(params.get(1), 3)));

        map.put("(a-b)^3", (int) (Math.pow(params.get(0), 3) - 3 * Math.pow(params.get(0), 2)
                * params.get(1) + 3 * params.get(0) * Math.pow(params.get(1), 2) -
Math.pow(params.get(1), 3)));
        int answer = map.get(equation);
        System.out.print(answer);
        return answer;
    }
}
```

MyServer.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;

public class MyServer {

    public static void main(String args[]) {
        try {
            System.out.print("Server started, waiting for client to enter equation");
            EquationSolverInterface stub = new EquationSolverClass();
            Naming.rebind("rmi://localhost:5000/narenderEquationSolver", stub);
        } catch (Exception e) {
```

```
            System.out.print("exception on server");
            System.out.println(e);
        }
    }
}
```

MyClient.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner; // Import the Scanner class
import java.util.Set;
import java.util.TreeSet;

public class MyClient {

    public static void main(String args[]) {
        while (true) {
            try {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter equation: ");
                String equation = sc.nextLine();
                Set<String> tree = new TreeSet<>();
                ArrayList<Integer> params = new ArrayList<Integer>();
                for (char c : equation.toCharArray()) {
                    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
                        tree.add(Character.toString(c));
                    }
                }
                int n = tree.size();
                Iterator value = tree.iterator();
                while (value.hasNext()) {
                    System.out.println("Enter value for " + value.next());
                    int temp = sc.nextInt();
                    params.add(temp);
                }
                EquationSolverInterface stub = (EquationSolverInterface)
Naming.lookup("rmi://localhost:5000/narenderEquationSolver");
                System.out.println("Answer: " + stub.EvaluateEquation(equation, params));
            } catch (Exception e) {
                System.out.println(e);
```

```
        }
      }
    }
}
```

OUTPUT:

For running this rmi example,

1) compile all the java files

javac *.java

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP5\src>javac *.java
```

2)create stub and skeleton object by rmic tool

rmic AdderRemote

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP5\src>rmic EquationSolverClass
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

3)start rmi registry in one command prompt

rmiregistry 5000

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP5\src>rmiregistry 5000
```

4)start the server in another command prompt

java MyServer

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP5\src>java MyServer
Server started, waiting for client to enter equationEquation recieved: (a+b)^225Equation recieved: (a+b+c)^2441Equation recieved: (a+b)^3512
```

5)start the client application in another command prompt
java MyClient

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP5\src>java MyClient
Enter equation: (a+b)^2
Enter value for a
2
Enter value for b
3
Answer: 25
Enter equation: (a+b+c)^2
Enter value for a
5
Enter value for b
9
Enter value for c
7
Answer: 441
Enter equation: (a+b)^3
Enter value for a
5
Enter value for b
3
Answer: 512
Enter equation:
```

CONCLUSION:

Hence we successfully implemented equation solver using remote method invocation

Aim: Implementation of Remote Method Communication using JDBC and RMI.

Theory:

Java Database Connectivity (JDBC): JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. Java JDBC Architecture is

1) JDBC Application- The JDBC application is in the topmost position of the JDBC architecture. JDBC application is the data processing application that intends to access the data from the different data storage units.

2) JDBC API- The JDBC API plays a significant role in the JDBC architecture. The JDBC API ensures that a stable connection is established between the data storage unit and the JDBC application.

3) JDBC Manager- The JDBC manager takes care of selecting the appropriate driver manager to ensure that the driver software chosen supports the data storage unit's requirements to offer an uninterrupted connection.

4) JDBC Drivers-The JDBC driver is the crucial unit in the JDBC architecture. JDBC driver is the first layer of JDBC architecture that has direct contact with the data storage units.

5) Data Storage Units-Data storage units are the base of JDBC. The data storage unit is the place where all the data is kept accessible for the JDBC Applications.

Steps to Connect Java JDBC

Step 1: Import the database

Step 2A: Add the mysql connector to project

Step 2B: Loading the drivers- In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program.

Class.forName()- Here we load the driver's class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the MYSQL driver as shown below as follows:

Class.forName("com.mysql.jdbc.Driver");

Step 3: Establish a connection using the Connection class object After loading the driver, establish connections via as shown below as follows:

Connection con = DriverManager.getConnection(url,user,password)

1) user: Username from which your SQL command prompt can be accessed.

2) password: password from which the SQL command prompt can be accessed.

3) con: It is a reference to the Connection interface.

4) url- Uniform Resource Locator. For eg. jdbc:mysql://localhost:3306/mysql

Step 4: Create a statement- Once a connection is established you can interact with the

database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.
Use of JDBC Statement is as follows:
Statement st = con.createStatement();

Step 5: Execute the query
Now comes the most important part i.e executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:
1) executeQuery() - The executeQuery() method of the Statement interface is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.
2) executeUpdate(sql query)- The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

Step 6: Closing the connections- So finally we have sent the data to the specified location and now we are on the verge of completing of our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is as shown below as follows:
con.close();

RMI (Remote Method Invocation):
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton. RMI uses stub and skeleton object for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

Stub: The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.
When the caller invokes method on the stub object, it does the following tasks:
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM)
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally returns the value to the caller.
Skeleton: The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:
1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshalls) the result to the caller.
The 6 steps to write
1. Create the remote interface

2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

A) Using MySQL create a Library database. Create table Book (Book_id, Book_name, Book_author) and retrieve the Book information from the Library database.

DATABASE :

```
create table books (
books_id int NOT NULL AUTO_INCREMENT,
books_name varchar(55) NOT NULL,
books_author varchar(55) NOT NULL,
PRIMARY KEY(books_id)
);
insert into books (books_name, books_author) values ("INTERNATIONAL RELATIONS WITH MODERN WORLD", "KHEMCHAND KESMANI");
 insert into books (books_name, books_author) values ("DATABASE MANGEMENT SYSTEM", "SUNITA JENA");
```

```
mysql> create database p6;
Query OK, 1 row affected (0.02 sec)

mysql> use p6;
Database changed
mysql> create table books (
    -> books_id int NOT NULL AUTO_INCREMENT,
    -> books_name varchar(55) NOT NULL,
    -> books_author varchar(55) NOT NULL,
    -> PRIMARY KEY(books_id)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into books(books_name,books_author) values("INTERNATIONAL RELATIONS WITH MODERN WORLD", "KHEMCHAND KESWANI");
Query OK, 1 row affected (0.00 sec)

mysql> insert into books(books_name,books_author) values("DATABASE MANGEMENT SYSTEM", "SUNITA JENA");
Query OK, 1 row affected (0.04 sec)
```

IDB.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
public interface IDB extends Remote {
   public String getData() throws RemoteException;
   public String getData(int id) throws RemoteException;
}
```

DBImpl.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;

public class DBImpl extends UnicastRemoteObject implements IDB {
   String str, str1;
   public DBImpl() throws RemoteException {
   }
   public String getData() {
      String URL = "jdbc:mysql://localhost:3308/p6";
      String UName = "root";
      String Pass = "Narend-10";
      try {
         Class.forName("com.mysql.jdbc.Driver");
         Connection con = DriverManager.getConnection(URL, UName, Pass);
         Statement s = con.createStatement();
         ResultSet rs = s.executeQuery("select * from books");
         ResultSetMetaData rsmd = rs.getMetaData();
         str = "";
         str1 = "";
         for (int i = 1; i <= rsmd.getColumnCount(); i++) {
            str1 = str1 + rsmd.getColumnName(i) + "\t";
         }
         System.out.println();
         while (rs.next()) {
            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
               str = str + rs.getString(i) + "\t";
            }
            str = str + "\n";
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      return (str1 + "\n" + str);
   }

   @Override
   public String getData(int id) throws RemoteException {
            String URL = "jdbc:mysql://localhost:3308/p6";
```

```
        String UName = "root";
        String Pass = "Narend-10";
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(URL, UName, Pass);
            PreparedStatement ps = con.prepareStatement("select * from books where
books_id=?");
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            ResultSetMetaData rsmd = rs.getMetaData();
            str = "";
            str1 = "";
            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                str1 = str1 + rsmd.getColumnName(i) + "\t";
            }
            System.out.println();
            while (rs.next()) {
                for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                    str = str + rs.getString(i) + "\t";
                }
                str = str + "\n";
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return (str1 + "\n" + str);
    }
}
```

Client.java:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.rmi.Naming;
/**
 * @author NARENDER KESWANI
 */
public class Client {

    public static void main(String[] args) {
        String ch1 = "", res = "";
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String url = "rmi://127.0.0.1/DBServer";
```

```
            System.out.println("Retriving Books Information....");
            while (true) {
                System.out.print("Enter choice:\n1. Get All Books\n2. Get Book By Id ");
                ch1 = br.readLine();
                if (ch1.equals("1")) {
                    IDB id = (IDB) Naming.lookup(url);
                    res = id.getData();
                    System.out.println(res);
                } else if (ch1.equals("2")) {
                    BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));
                    IDB id1 = (IDB) Naming.lookup(url);
                    res = id1.getData(Integer.parseInt(br1.readLine()));
                    System.out.println(res);
                } else {
                    System.out.println("Please select an option");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Server.java:

```
import java.rmi.Naming;
/**
 * @author NARENDER KESWANI
 */
public class Server {
    public static void main(String[] args) {
        try {
            DBImpl di = new DBImpl();
            Naming.rebind("rmi://127.0.0.1/DBServer", di);
            System.out.println("Server Registered.");
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
}
```

OUTPUT:

CMD:

```
C:\Users\NARENDER KESWANI>cd C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP7A\src

C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP7A\src>javac *.java
```

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP7A\src>start rmiregistry
```

SERVER:

```
run:
Server Registered.
```

CLIENT:

```
run:
Retriving Books Information....
Enter choice:
1. Get All Books
2. Get Book By Id 1
books_id        books_name      books_author
1       INTERNATIONAL RELATIONS WITH MODERN WORLD       KHEMCHAND KESMANI
2       DATABASE MANGEMENT SYSTEM       SUNITA JENA

Enter choice:
1. Get All Books
2. Get Book By Id 2
1
books_id        books_name      books_author
1       INTERNATIONAL RELATIONS WITH MODERN WORLD       KHEMCHAND KESMANI

Enter choice:
1. Get All Books
2. Get Book By Id 2
2
books_id        books_name      books_author
2       DATABASE MANGEMENT SYSTEM       SUNITA JENA

Enter choice:
1. Get All Books
2. Get Book By Id
```

B) Using MySQL create Electric_Bill database. Create table Bill. (bill_id, consumer_name, bill_due_date, bill_amount) and retrieve the Bill information from the Electril_Bill database.

DATABASE:

```
create table Bill(
bill_id int not null auto_increment,
consumer_name varchar(55) not null,
bill_due_date varchar(55) not null,
bill_amount int not null,
PRIMARY KEY(bill_id)
);

INSERT INTO BILL (consumer_name, bill_due_date, bill_amount) values("NARENDER
KESWANI","26-07-2022",2640);
INSERT INTO BILL (consumer_name, bill_due_date, bill_amount) values("RAJESH
KESWANI","30-07-2022", 3640);
```

```
mysql> create database Electril_Bill;
Query OK, 1 row affected (0.00 sec)

mysql> use Electril_Bill;
Database changed
mysql> create table Bill(
    -> bill_id int not null auto_increment,
    -> consumer_name varchar(55) not null,
    -> bill_due_date varchar(55) not null,
    -> bill_amount int not null,
    -> PRIMARY KEY(bill_id)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> INSERT INTO BILL(consumer_name, bill_due_date, bill_amount) values("NARENDER KESWANI","26-07-2022",2640);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO BILL(consumer_name, bill_due_date, bill_amount) values("RAJESH KESWANI","30-07-2022",3640);
Query OK, 1 row affected (0.00 sec)
```

IDB.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.rmi.*;
public interface IDB extends Remote {
  public String getData(int id) throws RemoteException;
}
```

DBImpl.java:

```
package p7b;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

/**
 * @author NARENDER KESWANI
 */
public class DBImpl extends UnicastRemoteObject implements IDB {
    String str, str1;
    public DBImpl() throws RemoteException {
    }

    @Override
    public String getData(int id) throws RemoteException {
        String URL = "jdbc:mysql://localhost:3308/Electril_Bill";
        String UName = "root";
        String Pass = "Narend-10";
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(URL, UName, Pass);
            PreparedStatement ps = con.prepareStatement("select * from bill where bill_id=?");
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            ResultSetMetaData rsmd = rs.getMetaData();
            str = "";
            str1 = "";
            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                str1 = str1 + rsmd.getColumnName(i) + "\t";
            }
            System.out.println();
            while (rs.next()) {
                for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                    str = str + rs.getString(i) + "\t";
                }
                str = str + "\n";
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
        return (str1 + "\n" + str);
    }
}
```

Client.java:

```
package p7b;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.rmi.Naming;
/**
 * @author NARENDER KESWANI
 */
public class Client {
    public static void main(String[] args) {
        String res = "";
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String url = "rmi://127.0.0.1/DBServerBill";
            System.out.println("Retriving Bill Information....");
            while (true) {
                System.out.print("Enter choice:\n1. Get Bill By Id ");
                    BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));
                    IDB id1 = (IDB) Naming.lookup(url);
                    res = id1.getData(Integer.parseInt(br1.readLine()));
                    System.out.println(res);
                }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Server.java:

```
package p7b;
import java.rmi.Naming;
/**
 * @author NARENDER KESWANI
 */
public class Server {
```

```
    public static void main(String[] args) {
      try {
          DBImpl di = new DBImpl();
          Naming.rebind("rmi://127.0.0.1/DBServerBill", di);
          System.out.println("Server Registered.");
      } catch (Exception e1) {
          e1.printStackTrace();
      }
   }
}
```

OUTPUT:

CMD:

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP7A\src\p7b>javac *.java
```

```
C:\Users\NARENDER KESWANI\Documents\NetBeansProjects\DsccP7A\src>start rmiregistry
```

SERVER:

```
run:
Server Registered.
```

CLIENT:

```
run:
Retriving Bill Information....
Enter choice:
1. Get Bill By Id 1
bill_id consumer_name   bill_due_date   bill_amount
1       NARENDER KESWANI        26-07-2022      2640

Enter choice:
1. Get Bill By Id 2
bill_id consumer_name   bill_due_date   bill_amount
2       RAJESH KESWANI  30-07-2022      3640
```

CONCLUSION:

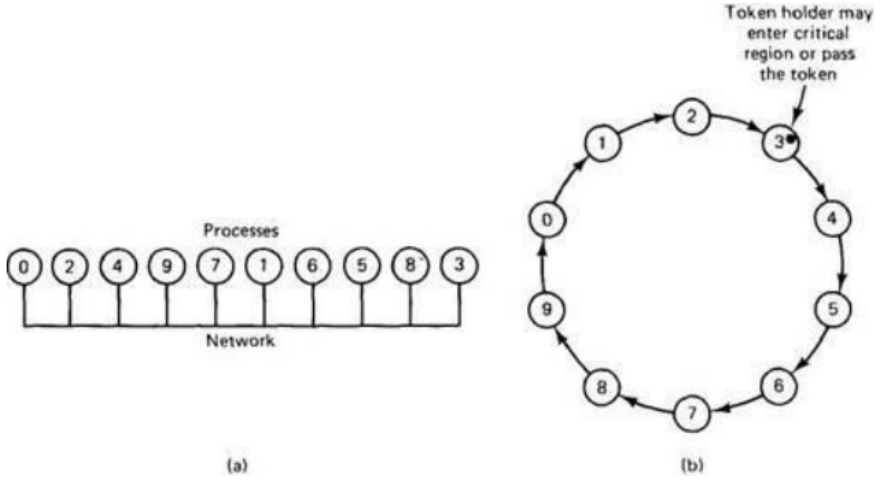
Thus, we successfully created a client-server-based application which helps us understand functioning of Remote Object Communication for database access

Aim: Implementation of mutual exclusion using the Token ring algorithm.

THEORY:

Token Ring Algorithm:

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each assigned a position in the ring. Each process knows who is next in line after itself. When the process ring is initialized, process 0 is given a token. The token circulates around the ring. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the work it needs to, and leaves the region. After it has exited, it passes the token to the next process in the ring. It is not allowed to enter the critical region again using the same token. If a process is handed the token by its neighbor and is not interested in entering a critical region, it just passes the token along to the next process.



➢ Advantages:

- The correctness of this algorithm is evident. Only one process has the token at any instant, so only one process can be in a CS
- Since the token circulates among processes in a well-defined order, starvation cannot occur.

➢ Disadvantages

- Once a process decides it wants to enter a CS, at worst it will have to wait for every other process to enter and leave one critical region.
- If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is not constant. The fact that the token has not been spotted for an hour does not mean that it has been lost; some process may still be using it.
- The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases. If we require a process receiving the token to acknowledge receipt, a dead

process will be detected when its neighbour tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can pass the token to the next member down the line.

SOURCE CODE:

TokenServer.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.net.*;
import java.io.*;
class TokenServer {
    public static DatagramSocket ds;
    public static DatagramPacket dp;
    public static void main(String[] args) throws Exception {
        try {
            ds = new DatagramSocket(1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        while (true) {
            byte buff[] = new byte[1024];
            ds.receive(dp = new DatagramPacket(buff, buff.length));
            String str = new String(dp.getData(), 0, dp.getLength());
            System.out.println("Message from " + str);
        }
    }
}
```

TokenClient1.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.net.*;
import java.io.*;
class TokenClient1 {
    public static DatagramSocket ds;
    public static DatagramPacket dp;
    public static BufferedReader br;
    static int cp = 100;
    public static void main(String[] args) throws Exception {
```

```
        boolean hasToken;
        try {
            ds = new DatagramSocket(100);
        } catch (Exception e) {
            e.printStackTrace();
        }
        hasToken = true;
        while (true) {
            if (hasToken == true) {
                System.out.println("Do you want to enter data...(yes/no):");
                br = new BufferedReader(new InputStreamReader(System.in));
                String ans = br.readLine();
                if (ans.equalsIgnoreCase("yes")) {
                    System.out.println("ready to send");
                    System.out.println("sending");
                    System.out.println("Enter the data");
                    br = new BufferedReader(new InputStreamReader(System.in));
                    String str = "Client-1===> " + br.readLine();
                    byte buff[] = new byte[1024];
                    buff = str.getBytes();
                    ds.send(new DatagramPacket(buff, buff.length, InetAddress.getLocalHost(), 1000));
                    System.out.println("now sending");
                } else if (ans.equalsIgnoreCase("no")) {
                    System.out.println("I am busy state");
//sending msg to client-2
                    String msg = "Token";
                    byte bf1[] = new byte[1024];
                    bf1 = msg.getBytes();
                    ds.send(new DatagramPacket(bf1, bf1.length, InetAddress.getLocalHost(), 200));
                    hasToken = false;
//recevingmsg from client-2
                    byte bf2[] = new byte[1024];
                    ds.receive(dp = new DatagramPacket(bf2, bf2.length));
                    String clientmsg = new String(dp.getData(), 0, dp.getLength());
                    System.out.println("The data is " + clientmsg);

                    if (clientmsg.equals("Token")) {
                        hasToken = true;
                    }
                    System.out.println("I am leaving busy state");
                }
            } else {
                System.out.println("Entering in receive mode.");
                byte bf[] = new byte[1024];
```

```
            ds.receive(dp = new DatagramPacket(bf, bf.length));
            String clientmsg1 = new String(dp.getData(), 0, dp.getLength());
            System.out.println("The data is " + clientmsg1);
            if (clientmsg1.equals("Token"));
            {
               hasToken = true;
            }
         }
      }
   }
}
```

TokenClient2.java:

```
/**
 * @author NARENDER KESWANI
 */
import java.net.*;
import java.io.*;
class TokenClient2 {
   static DatagramSocket ds;
   static DatagramPacket dp;
   static BufferedReader br;
   public static void main(String[] args) throws Exception {
      try {
         ds = new DatagramSocket(200);
      } catch (Exception e) {
         e.printStackTrace();
      }
      boolean hasToken = true;
      while (true) {
//System.out.println("Entering if");
         if (hasToken == true) {
            System.out.println("Do you want to enter data(Yes/No):");
            br = new BufferedReader(new InputStreamReader(System.in));
            String str = br.readLine();
            if (str.equalsIgnoreCase("yes")) {
               System.out.println("Enter Data; ");
               br = new BufferedReader(new InputStreamReader(System.in));
               String msg = "Client-2===>" + br.readLine();
               byte bf1[] = new byte[1024];
               bf1 = msg.getBytes();
               ds.send(new DatagramPacket(bf1, bf1.length, InetAddress.getLocalHost(), 1000));
               System.out.println("Data sent");
```

```
            } else {
//send to client 1.
                String clientmsg = "Token";
                byte bf2[] = new byte[1024];
                bf2 = clientmsg.getBytes();
                ds.send(new DatagramPacket(bf2, bf2.length, InetAddress.getLocalHost(), 100));
                hasToken = false;
            }
        } else {
            try {
                byte buff[] = new byte[1024];
                System.out.println("Entering in receiving mode.");
                ds.receive(dp = new DatagramPacket(buff, buff.length));
                String clientmsg1 = new String(dp.getData(), 0, dp.getLength());
                System.out.println("The data is " + clientmsg1);
                if (clientmsg1.equals("Token")) {
                    hasToken = true;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
  }
}
```

OUTPUT:

Server:

```
run:
Message from Client-1===> hi, i am narender
Message from Client-2===>hi, narender
Message from Client-1===> what is the status of the project
Message from Client-2===>completed today!!!
Message from Client-1===> good!
Message from Client-2===>hm
'
```

Client1:

```
Do you want to enter data...(yes/no):
yes
ready to send
sending
Enter the data
hi, i am narender
now sending
Do you want to enter data...(yes/no):
yes
ready to send
sending
Enter the data
what is the status of the project
now sending
Do you want to enter data...(yes/no):
yes
ready to send
sending
Enter the data
good!
now sending
Do you want to enter data...(yes/no):
NO
I am busy state
The data is Token
I am leaving busy state
Do you want to enter data...(yes/no):
```

Client2:

```
run:
Do you want to enter data(Yes/No):
yes
Enter Data;
hi, narender
Data sent
Do you want to enter data(Yes/No):
yes
Enter Data;
completed today!!!
Data sent
Do you want to enter data(Yes/No):
yes
Enter Data;
hm
Data sent
Do you want to enter data(Yes/No):
NO
Entering in receiving mode.
The data is Token
Do you want to enter data(Yes/No):
```

CONCLUSION:

Thus, we successfully implemented Token ring mutual exclusion.

Aim: Implementation of Storage as a Service using Google Docs

THEORY:

Google Drive

Google Drive is a file storage and synchronization service developed by Google. Launched on April 24, 2012, Google Drive allows users to store files in the cloud (on Google's servers), synchronize files across devices, and share files. In addition to a web interface, Google Drive offers apps with offline capabilities for Windows and macOS computers, and Android and iOS smartphones and tablets. Google Drive encompasses Google Docs, Google Sheets, and Google Slides, which are a part of the Google Docs Editors office suite that permits collaborative editing of documents, spreadsheets, presentations, drawings, forms, and more. Files created and edited through the Google Docs suite are saved in Google Drive.

Google Drive offers users 15 GB of free storage through Google One. Google One also offers 100 GB, 200 GB, 2 TB, offered through optional paid plans. Files uploaded can be up to 750 GB in size. Users can change privacy settings for individual files and folders, including enabling sharing with other users or making content public. On the website, users can search for an image by describing its visuals, and use natural language to find specific files, such as "find my budget spreadsheet from last December".

The website and Android app offer a Backups section to see what Android devices have data backed up to the service, and a completely overhauled computer app released in July 2017 allows for backing up specific folders on the user's computer. A Quick Access feature can intelligently predict the files users need.

Google Drive is a key component of Google Workspace, Google's monthly subscription offering for businesses and organizations that operated as G Suite until October 2020. As part of select Google Workspace plans, Drive offers unlimited storage, advanced file audit reporting, enhanced administration controls, and greater collaboration tools for teams.

Benefits of Google drive:

1. Ability to Access Files from Everywhere

The most significant advantage of using Google drive to store your data is that you can access it anywhere. Google servers allow you to sign in using your Gmail account and access your data anytime. What you need is an internet connection and a device that can go online. This ensures you never lose your files and important data even if your computer, smartphone, or other gadget is stolen or damaged.

2. Ability to edit files

The ability to edit files in Google Drive is an added advantage that you will not find with other data storage options. You get instant access to various suite editing tools, including Google Spreadsheets, Google Docs and Google Slides and many others. The access to edit your files allows making any changes you wish to before saving them again.

3. Compatibility with most devices

You are not limited to using a single device to access Google Drive. You can use different gadgets like an Android phone, iPad, Mac, iPhone or a PC to access your stored data anytime from anywhere. This makes Google Drive beneficial and highly efficient.

4. Quick Files Search

Searching a specific file in Google Drive is made easy by the search feature that allows you to filter the file type and use keywords to find the file. Even a word on an image included in your file can help you locate it fast. With the ability to store thousands of documents, it may be impossible to find a file by scanning through them all. Thanks to the quick file search options, you can locate your file within seconds.

5. Ability to view different file types

This is a crucial pro of Google Drive. You do not have to install different file types to view documents on your device; Google can open many types of files, including Adobe, HD video, Photoshop, and Illustrator.

6. Easy sharing

It takes a few clicks to share files and data with other people. This makes it easy for people to work as a group. You need to set up permissions to allow your collaborators to edit, view or comment on specific files. It is a great feature that also ensures Google Drive is safe because only those permitted can access your files.

7. Open discussion

If you have a project that requires participants to carry out discussions, Google Drive makes it possible. It has a comments feature where users can make comments or reply to others and get feedback instantly. A group of people can work on a project without the need for other forms of communication e, thanks to the comments feature that enables them to collaborate smoothly and achieve their common goal.

8. Free Storage space of up to 15 GB

When you sign for a Google account, you get 15 GB of storage space for free. It is shared across Google Photos, Gmail and Google Drive. If you have plenty of files and data to store, you can upgrade at the cost of $2.49 for 25 GB, $4.99 for 100 GB or$49.99 for 1 TB. The plans are monthly, and choose what suits you depending on your needs.

9. Excellent User interface

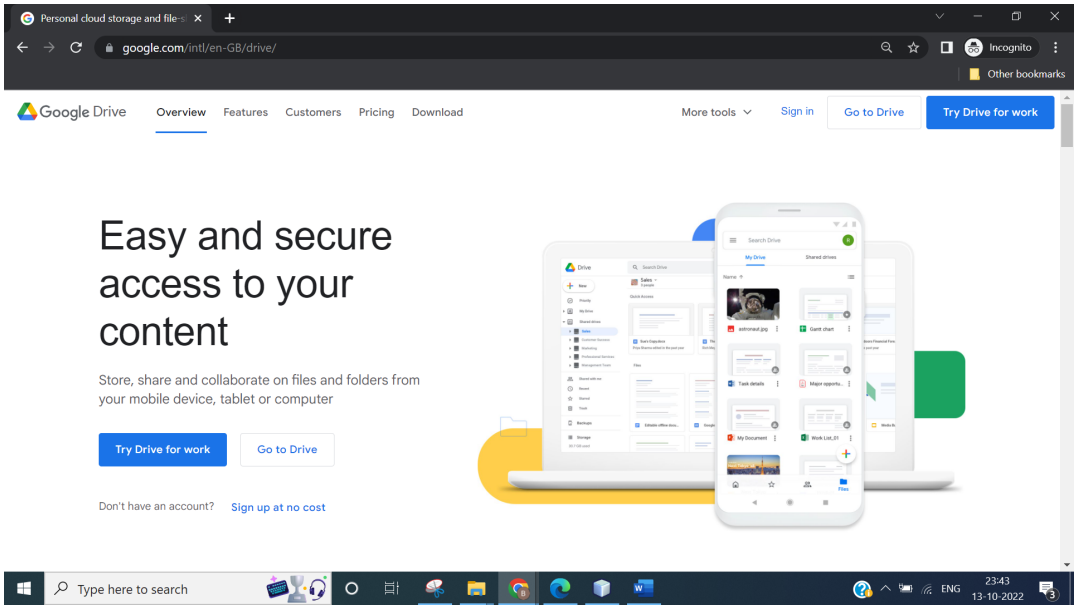
Easy to navigate interface is another great advantage of Google Drive. Even first-time users do not get lost because this online storage option is easy to manage. You can manoeuvre through the files and data and locate whatever you are looking for very fast. The user interface is friendly and made for anyone who wishes to store their information using cloud computing.

10. Affordability and great usability

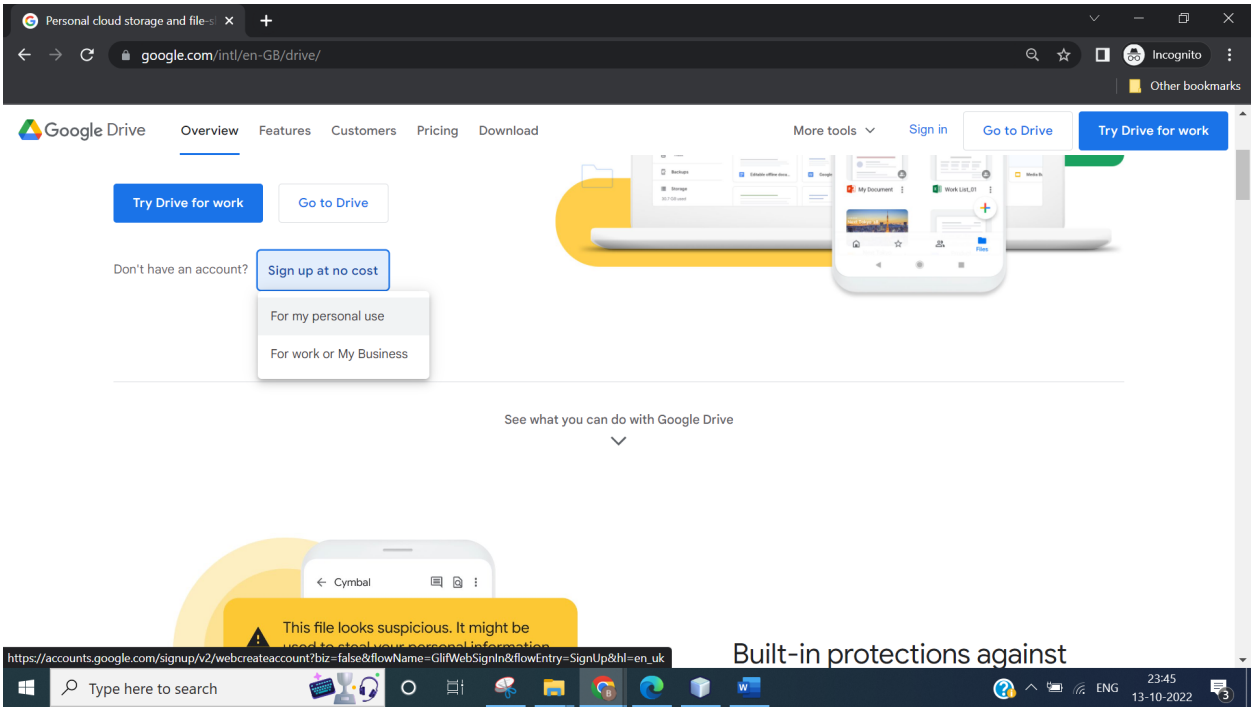
Google Drive comes with a wide range of features that encourages everyone, including new users, to browse through it effortlessly. It is free, and you have up to 15 GB of storage space. This makes it affordable and great for users who do not have much to store and looking for free storage.

Steps to create Google drive account:

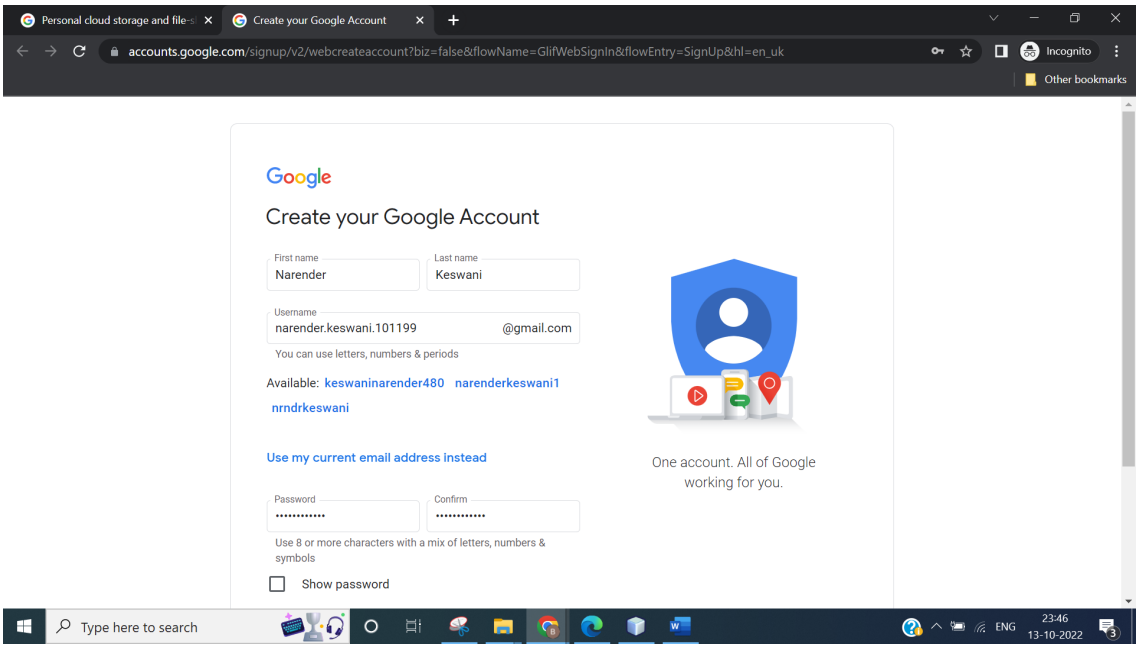
1. Search google drive in chrome browser and visit the first link, select individual and then go to drive.



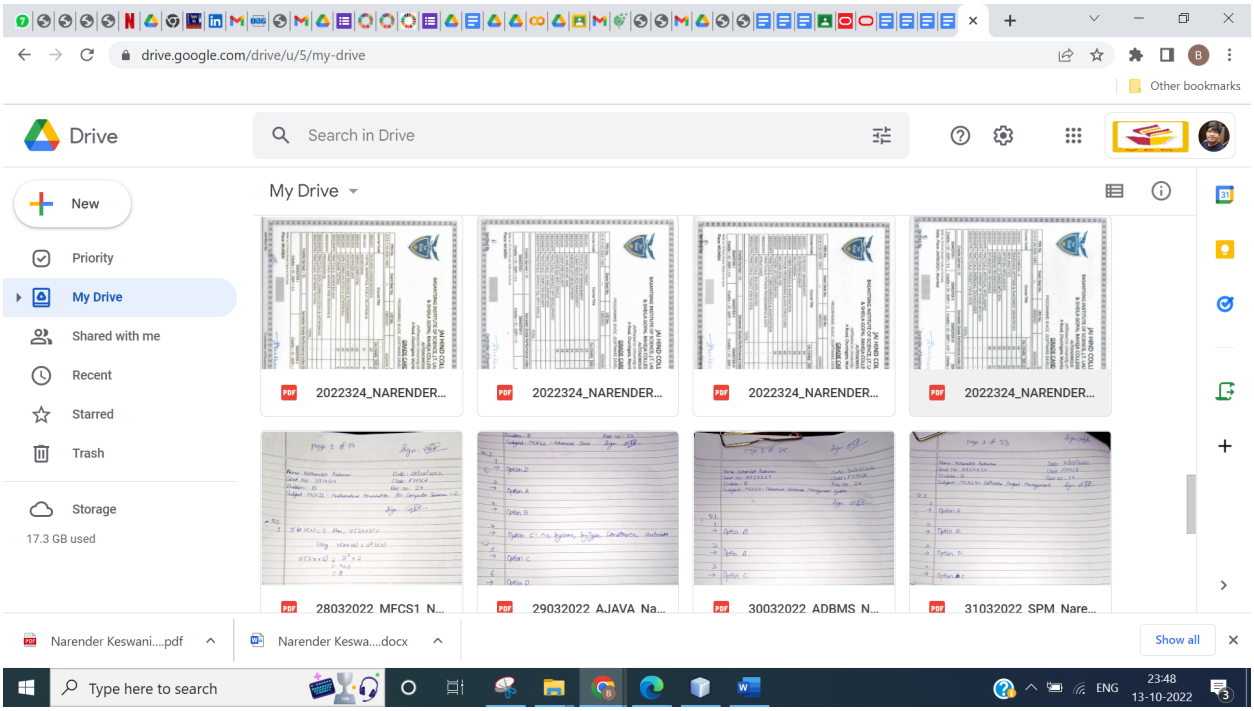
2. Next, Create new account for myself



3. Fill in all the necessary personal details, choose any username and password for the account.

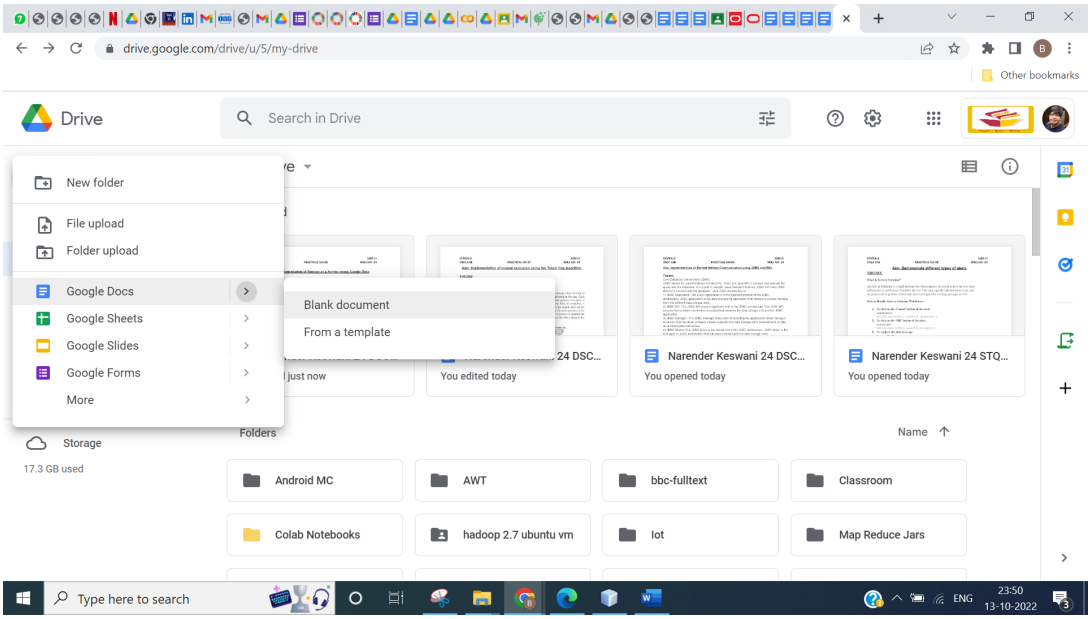


4. Once your account is created, we can access all free services of google drive.

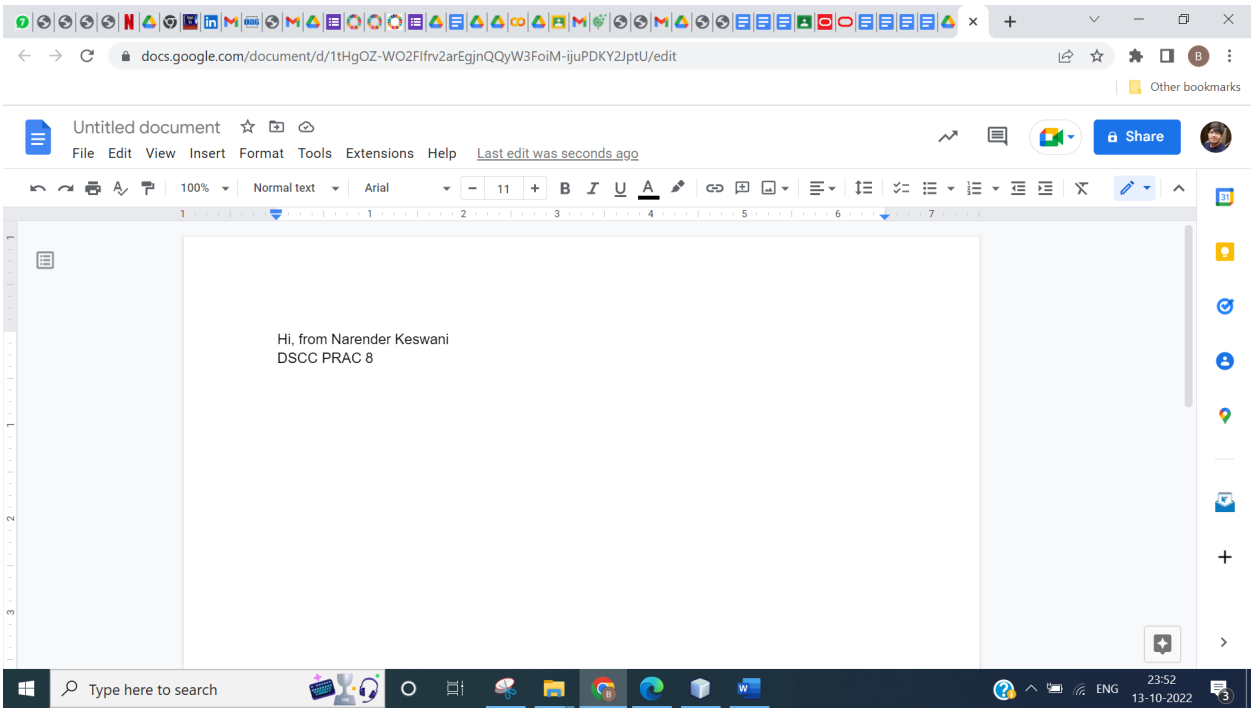


Steps to create Google Doc:

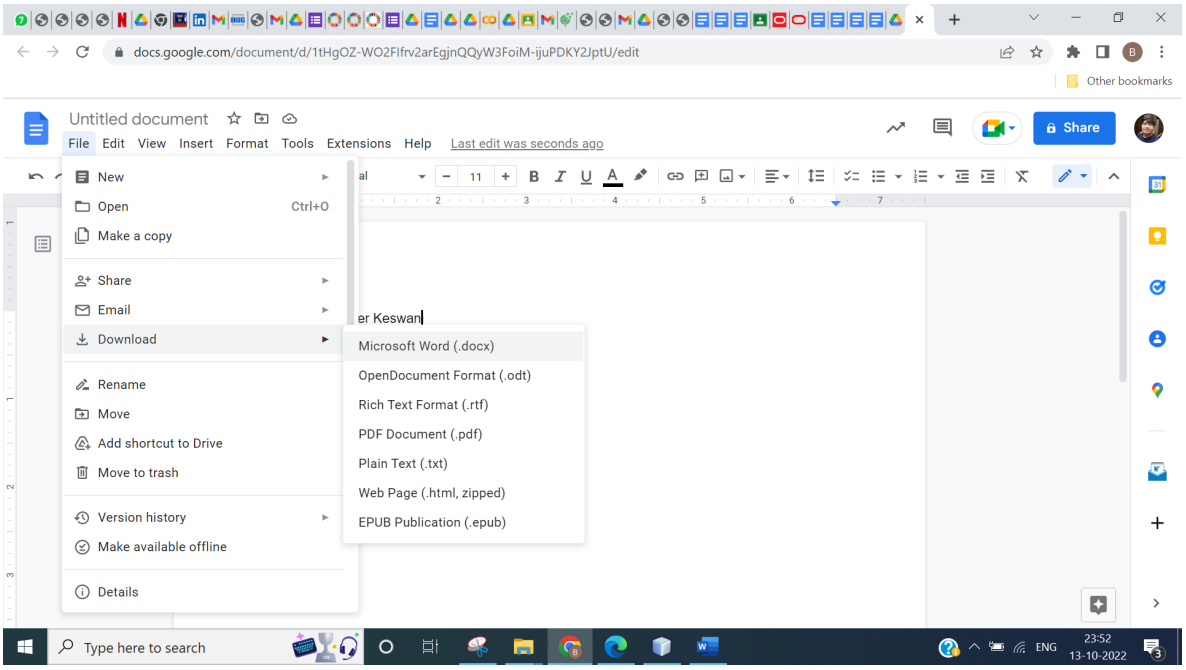
1. To create a new doc click on new, select google doc, select new blank document.



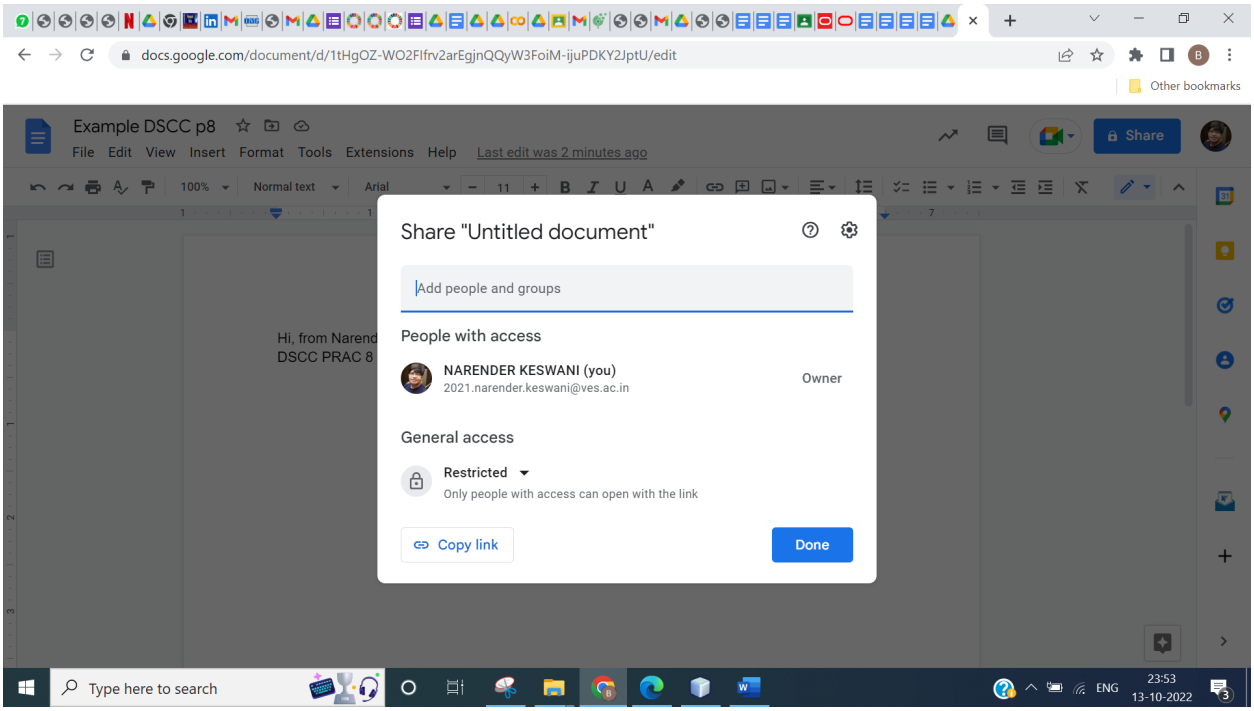
2. Now you can type text, add images, add header/footer and many more features are available in google doc.



3. We can download doc in any format.



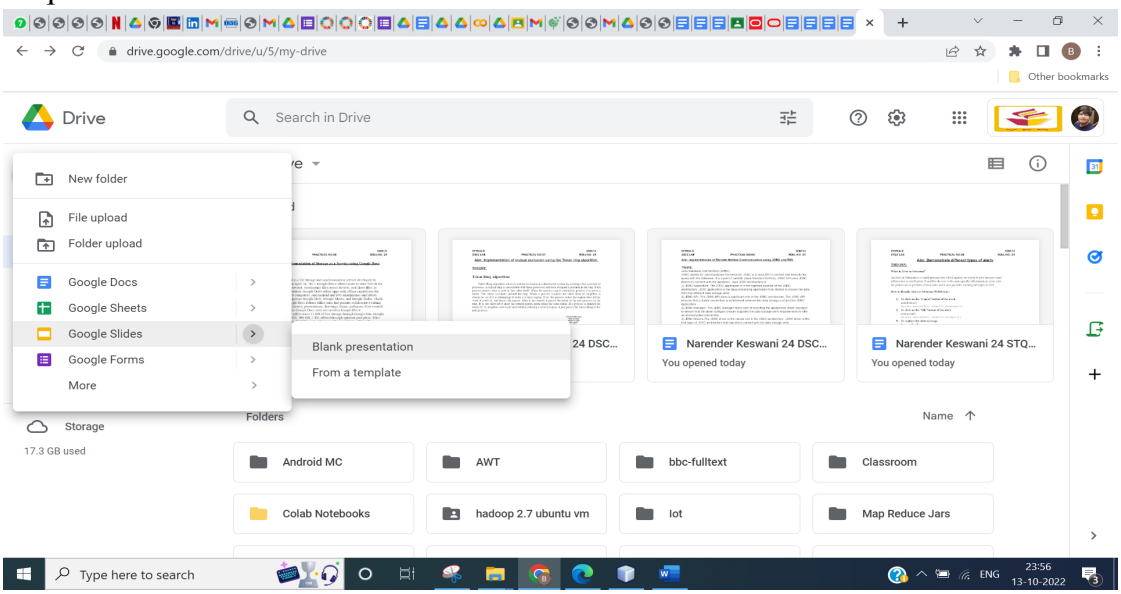
4. Share doc is also available in google doc



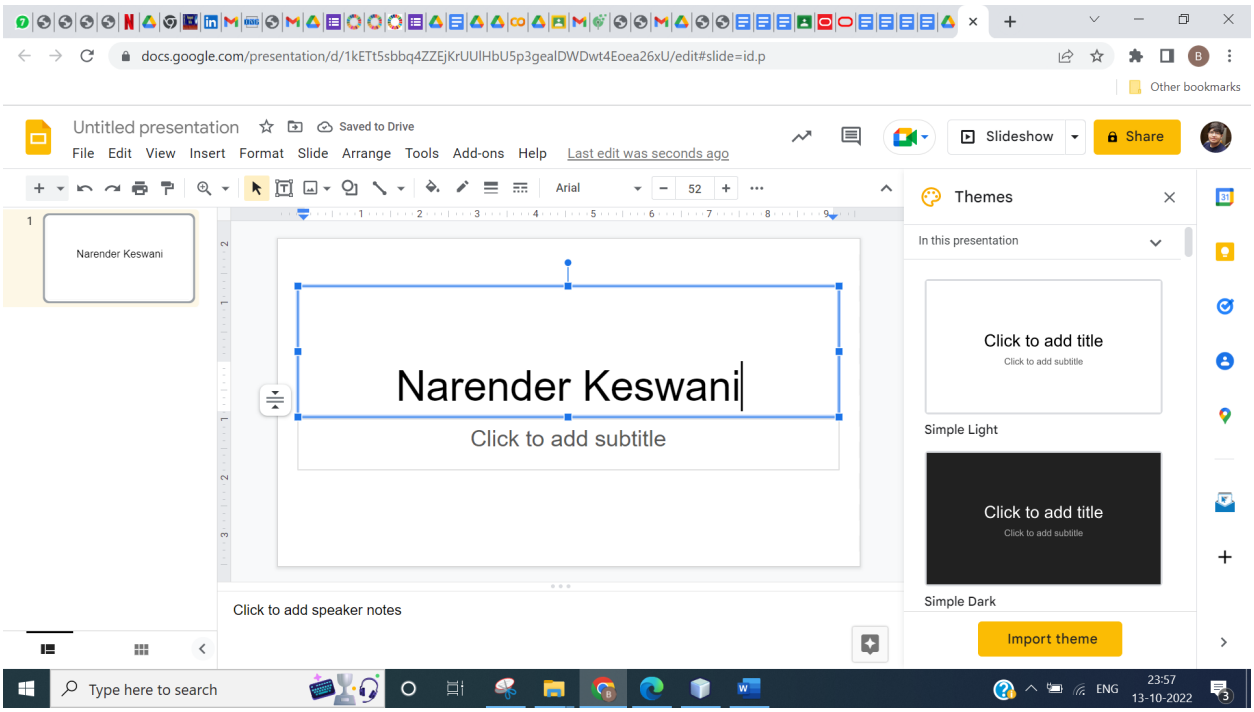
5. All docs automatically save into our google drive folder so we can access it anytime and anywhere.

Steps to create Google Slides:

1. To create a new slide, click on new, select Google Slides, select Blank presentation.



2. Now you can choose any theme, type text and many features are available in Google slides.



3. Similar to Google Docs in Google Slide you can download slide in various formats and also share them in the same way as mentioned above.
4. All slides are saved in my drive and we can access it anywhere and anytime.

CONCLUSION:

Thus, we successfully implemented Storage as a Service using Google Docs.

Aim: To develop applications using Google App Engine by using Eclipse IDE. Find the position of a target value within a sorted integer array. (Binary Search)

Title: Application using Google App Engine

Theory:

Platform as a Service – PaaS:

Platform as a Service (PaaS) provides a runtime environment. It allows programmers to easily create, test, run, and deploy web applications. PaaS specifically provides a platform for customers to develop, run, and manage applications without building and maintaining the cloud infrastructure required to develop and launch an app.

PaaS can be delivered in three different formats.

• First is as a cloud service from the provider. In this configuration, the customer controls software deployment with minimal configuration options. The Platform as a Service provider supplies the networking, servers, storage, operating system (OS), middleware (e.g., Java runtime, .NET runtime, integration, etc.), database and other services to host the consumer's application.

• The second PaaS configuration can be run as a private service (software or appliance) behind a firewall.

• The third PaaS configuration can be run as software deployed on public infrastructure as a service such as AWS.

Google App Engine:

Google App Engine is a Platform as a Service (PaaS) product that provides Web app developers and enterprises with access to Google's scalable

hosting and tier 1 Internet service. The App Engine requires that apps be written in Java or Python, store data in Google BigTable and use the Google query language. Non-compliant applications require modification to use App Engine. Google App Engine provides more infrastructure than other scalable hosting services such as Amazon Elastic Compute Cloud (EC2). The App Engine also eliminates some system administration and developmental tasks to make it easier to write scalable applications. Google App Engine is free up to a certain amount of resource usage. Users exceeding the per-day or per-minute usage rates for CPU resources, storage, number of API calls or requests and concurrent requests can pay for more of these resources.

Steps:

1. Go to Eclipse
2. Click on Help -> Eclipse Marketplace
3. Search for Google Cloud tools for Eclipse and install
4. Download and install Google cloud sdk
5. Through google cloud sdk shell install
app-engine-java components using commands:

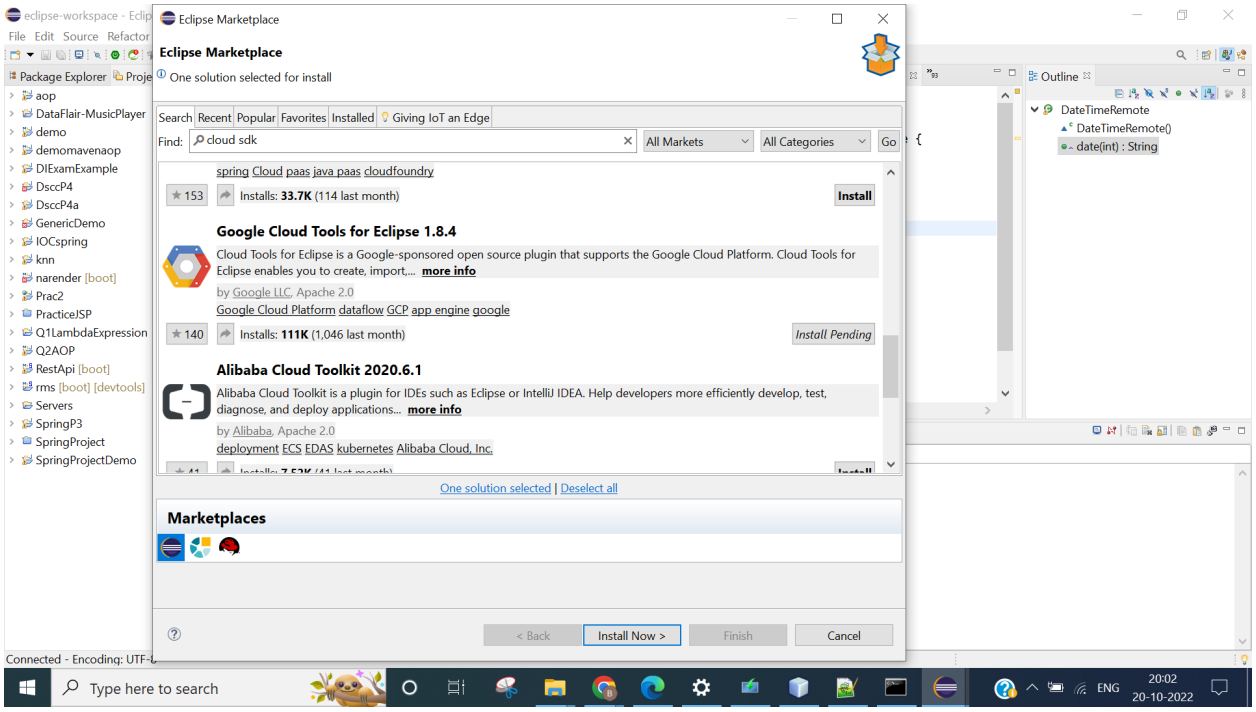
▪ gcloud components list (to check if app-engine-java
(components are already installed or not)

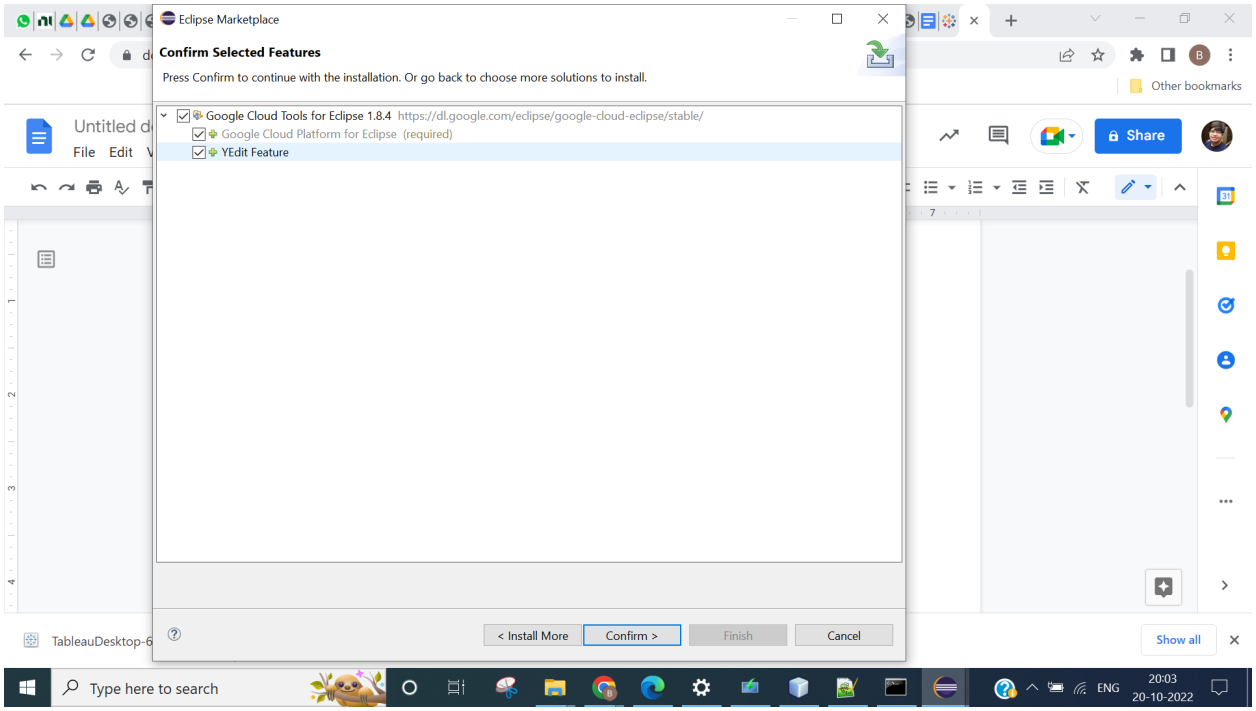
▪ gcloud components install app-engine-java

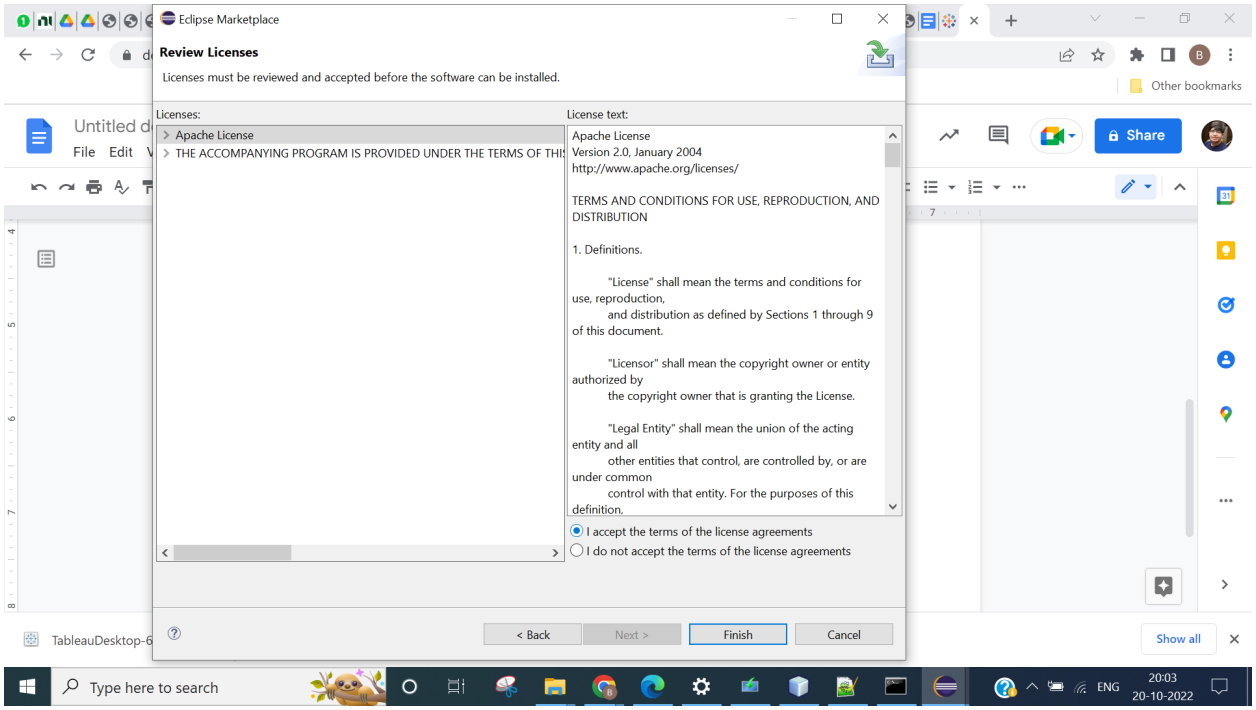
6. Click on Windows -> Preferences and search for Google Cloud Tool

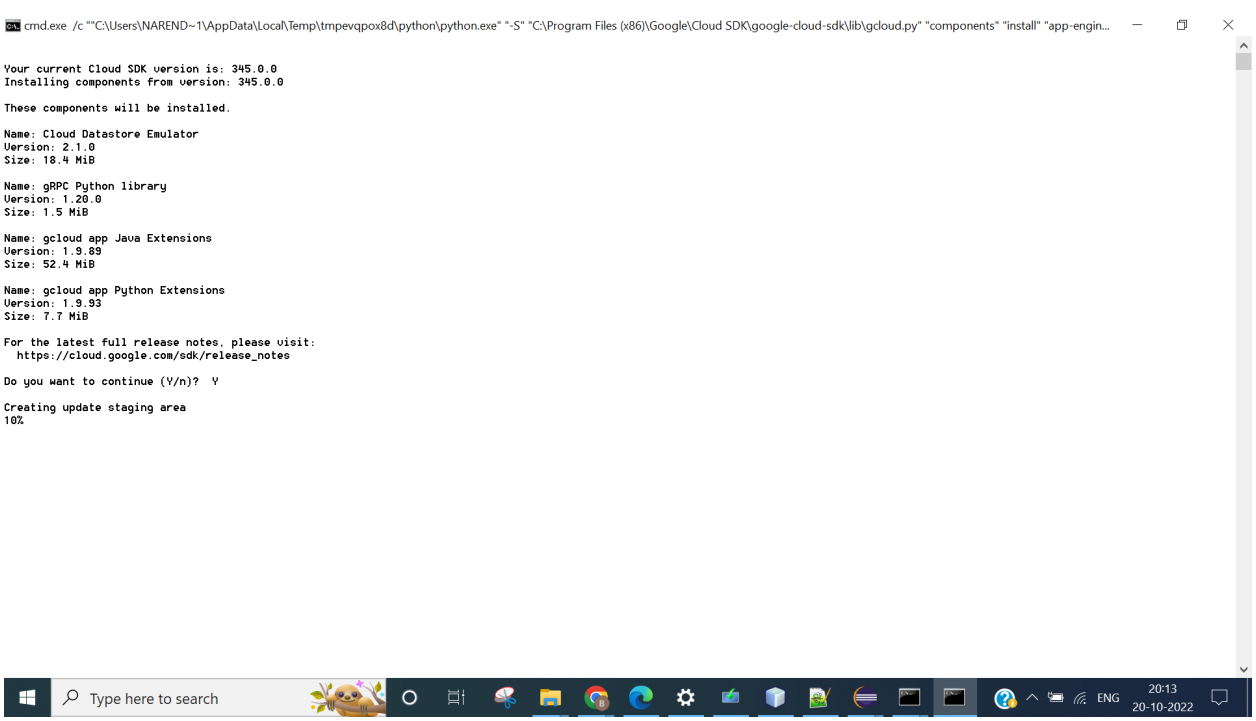
Click on Google Cloud Platform -> Create New Project -> Select Standard Java

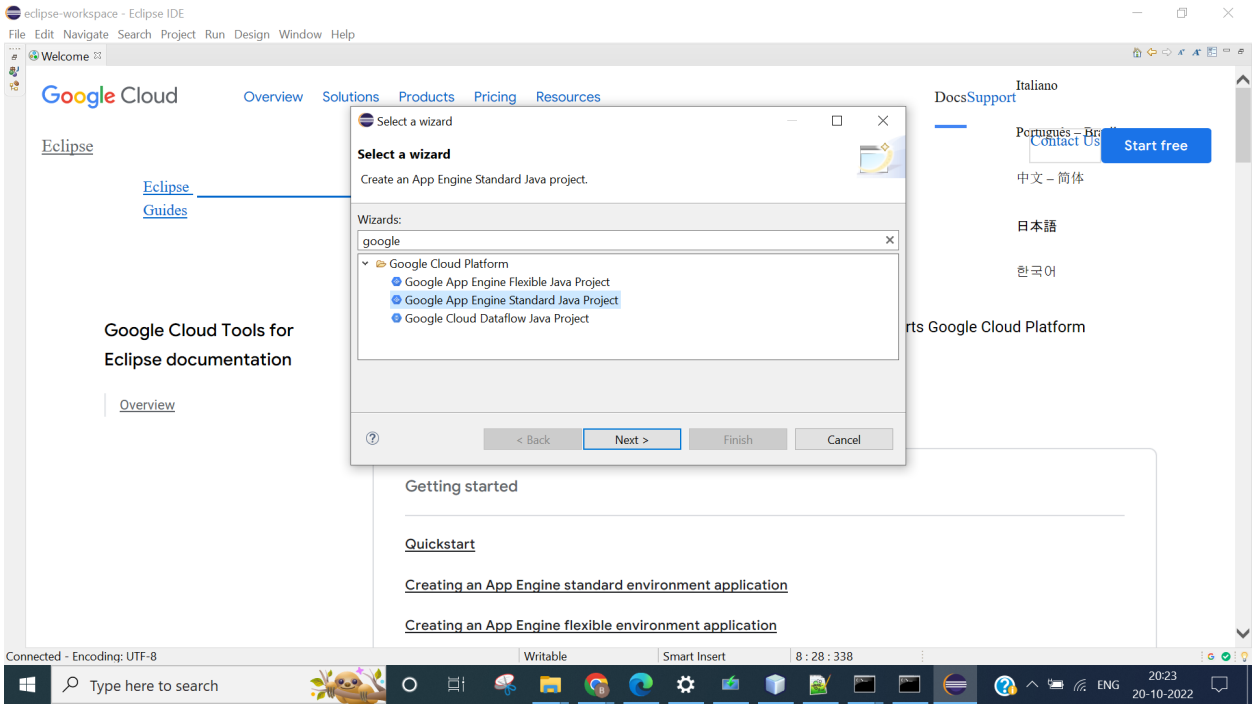
Project

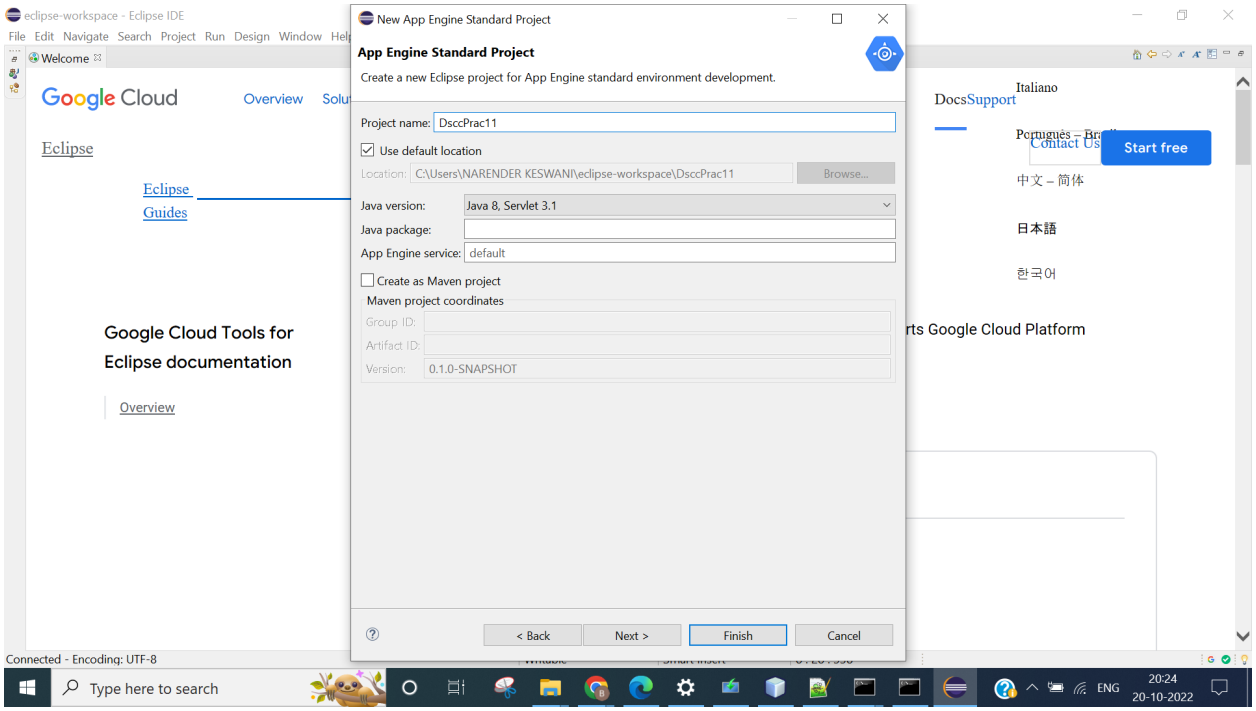

Eclipse Marketplace
Eclipse Marketplace
One solution selected for install
Search Recent Popular Favorites Installed Giving IoT an Edge
Find: cloud sdk
All Markets
All Categories
Go
spring Cloud paas java paas cloudfoundry
153
Installs: 33.7K (114 last month)
Install
Google Cloud Tools for Eclipse 1.8.4
Cloud Tools for Eclipse is a Google-sponsored open source plugin that supports the Google Cloud Platform. Cloud Tools for Eclipse enables you to create, import,... more info
by Google LLC, Apache 2.0
Google Cloud Platform dataflow GCP app engine google
140
Installs: 111K (1,046 last month)
Install Pending
Alibaba Cloud Toolkit 2020.6.1
Alibaba Cloud Toolkit is a plugin for IDEs such as Eclipse or IntelliJ IDEA. Help developers more efficiently develop, test, diagnose, and deploy applications... more info
by Alibaba, Apache 2.0
deployment ECS EDAS kubernetes Alibaba Cloud, Inc.
One solution selected | Deselect all
Marketplaces
< Back
Install Now >
Finish
Cancel
Type here to search
ENG
20:02
20-10-2022

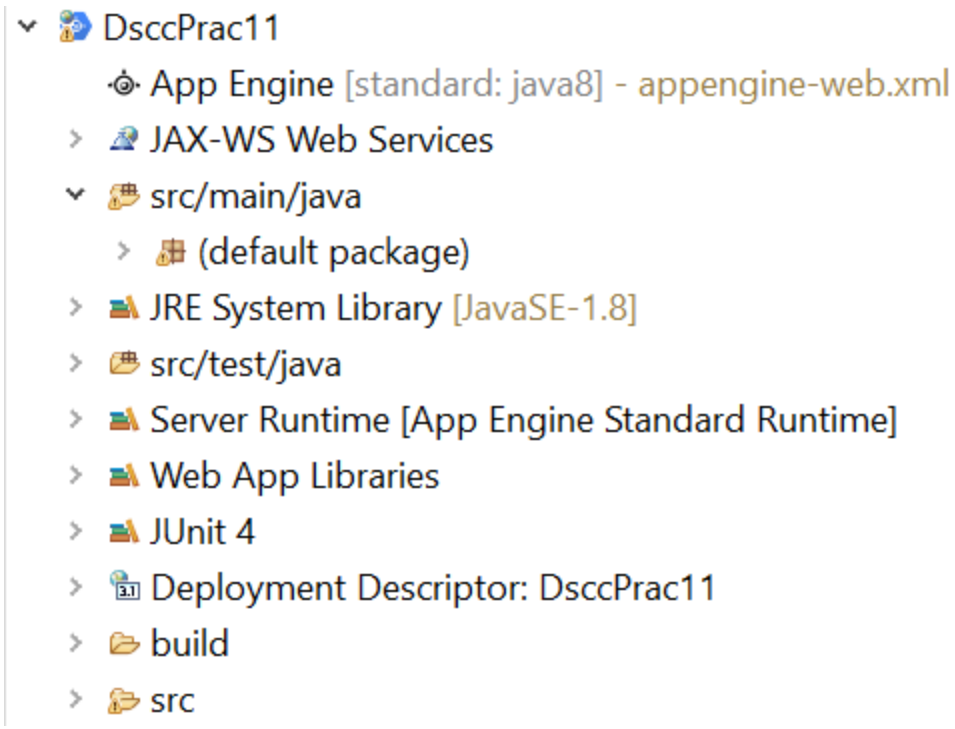

Eclipse Marketplace
Confirm Selected Features
Press Confirm to continue with the installation. Or go back to choose more solutions to install.
Google Cloud Tools for Eclipse 1.8.4 https://dl.google.com/eclipse/google-cloud-eclipse/stable/
Google Cloud Platform for Eclipse (required)
YEdit Feature
< Install More
Confirm >
Finish
Cancel
Share
Show all
Type here to search
ENG
20:03
20-10-2022

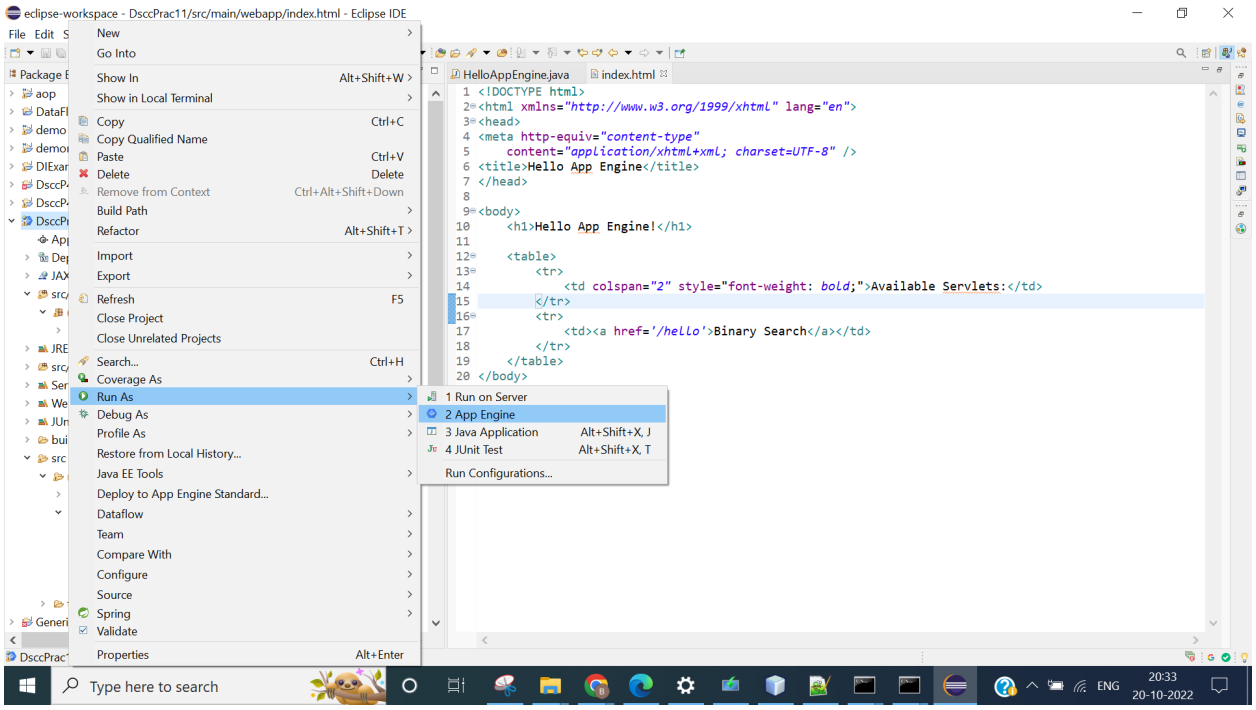

Eclipse Marketplace
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
Apache License
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS
License text:
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.
"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
I accept the terms of the license agreements
I do not accept the terms of the license agreements
< Back
Next >
Finish
Cancel
Share
Other bookmarks
Show all
Type here to search
ENG
20:03
20-10-2022


cmd.exe /c ""C:\Users\NAREND~1\AppData\Local\Temp\tmpevqpox8d\python\python.exe" "-S" "C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\lib\gcloud.py" "components" "install" "app-engin...
Your current Cloud SDK version is: 345.0.0
Installing components from version: 345.0.0
These components will be installed.
Name: Cloud Datastore Emulator
Version: 2.1.0
Size: 18.4 MiB
Name: gRPC Python library
Version: 1.20.0
Size: 1.5 MiB
Name: gcloud app Java Extensions
Version: 1.9.89
Size: 52.4 MiB
Name: gcloud app Python Extensions
Version: 1.9.93
Size: 7.7 MiB
For the latest full release notes, please visit:
https://cloud.google.com/sdk/release_notes
Do you want to continue (Y/n)? Y
Creating update staging area
10%
Type here to search
ENG
20:13
20-10-2022


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Design Window Help
Welcome
Google Cloud
Overview Solutions Products Pricing Resources
Eclipse
Eclipse Guides
Google Cloud Tools for Eclipse documentation
Overview
Select a wizard
Select a wizard
Create an App Engine Standard Java project.
Wizards:
google
Google Cloud Platform
Google App Engine Flexible Java Project
Google App Engine Standard Java Project
Google Cloud Dataflow Java Project
< Back
Next >
Finish
Cancel
Getting started
Quickstart
Creating an App Engine standard environment application
Creating an App Engine flexible environment application
Connected - Encoding: UTF-8
Writable
Smart Insert
8 : 28 : 338
Type here to search
ENG
20:23
20-10-2022


eclipse-workspace - Eclipse IDE
New App Engine Standard Project
App Engine Standard Project
Create a new Eclipse project for App Engine standard environment development.
Project name: DsccPrac11
Use default location
Location: C:\Users\NARENDER KESWANI\eclipse-workspace\DsccPrac11
Browse...
Java version: Java 8, Servlet 3.1
Java package:
App Engine service: default
Create as Maven project
Maven project coordinates
Group ID:
Artifact ID:
Version: 0.1.0-SNAPSHOT
< Back
Next >
Finish
Cancel
Google Cloud Tools for Eclipse documentation
Overview
Connected - Encoding: UTF-8
Type here to search
ENG
20:24
20-10-2022


DsccPrac11
App Engine [standard: java8] - appengine-web.xml
JAX-WS Web Services
src/main/java
(default package)
JRE System Library [JavaSE-1.8]
src/test/java
Server Runtime [App Engine Standard Runtime]
Web App Libraries
JUnit 4
Deployment Descriptor: DsccPrac11
build
src


eclipse-workspace - DsccPrac11/src/main/webapp/index.html - Eclipse IDE
File Edit
New
Go Into
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Search... Ctrl+H
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Deploy to App Engine Standard...
Dataflow
Team
Compare With
Configure
Source
Spring
Validate
Properties Alt+Enter
1 Run on Server
2 App Engine
3 Java Application Alt+Shift+X, J
4 JUnit Test Alt+Shift+X, T
Run Configurations...
HelloAppEngine.java
index.html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<meta http-equiv="content-type"
content="application/xhtml+xml; charset=UTF-8" />
<title>Hello App Engine</title>
</head>
<body>
<h1>Hello App Engine!</h1>
<table>
<tr>
<td colspan="2" style="font-weight: bold;">Available Servlets:</td>
</tr>
<tr>
<td>Binary Search</td>
</tr>
</table>
</body>
Type here to search
ENG
20:33
20-10-2022

HelloAppEngine.java:

```
import java.io.IOException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloAppEngine", urlPatterns = { "/hello" })
public class HelloAppEngine extends HttpServlet {

	@Override
	public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {

		response.setContentType("text/plain");
		response.setCharacterEncoding("UTF-8");

		int arr[] = { 10, 20, 30, 40, 50 };
		int key = 30;
		response.getWriter().print("Array : ");
		for (int i = 0; i < 5; i++) {
			response.getWriter().print(" " + arr[i]);
		}
		response.getWriter().println("\n\nSearch for element: " + key + "\n");
		int last = arr.length - 1;
		int first = 0;
		int mid = (first + last) / 2;
		while (first <= last) {
			if (arr[mid] < key) {
				first = mid + 1;
			} else if (arr[mid] == key) {
				response.getWriter().print("Element is found at index: " + mid +
"\n");
				break;

			} else {
				last = mid - 1;
			}
			mid = (first + last) / 2;
		}
		if (first > last) {
			response.getWriter().print("Element is not found!");
		}
```

```
        }
}
```

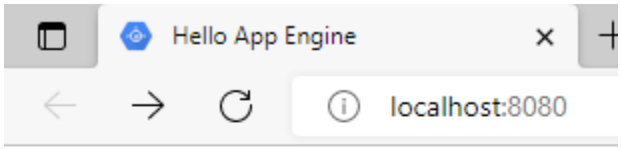
index.html:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<meta http-equiv="content-type"
        content="application/xhtml+xml; charset=UTF-8" />
<title>Hello App Engine</title>
</head>

<body>
        <h1>Hello App Engine!</h1>

        <table>
                <tr>
                        <td colspan="2" style="font-weight: bold;">Available Servlets:</td>
                </tr>
                <tr>
                        <td><a href='/hello'>Binary Search</a></td>
                </tr>
        </table>
        <a>Developed by Narender Keswani</a>
</body>
</html>
```

OUTPUT:

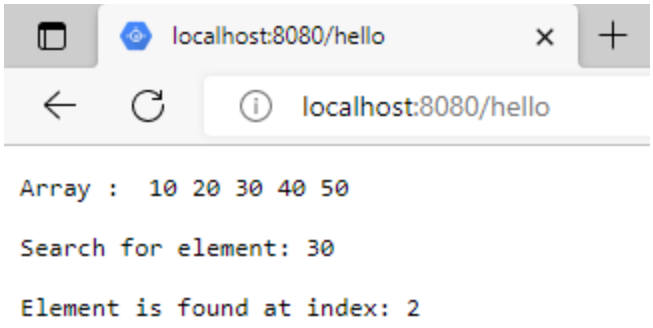


Hello App Engine!

Available Servlets:

Binary Search

Developed by Narender Keswani



CONCLUSION:

Successfully developed application using Google App Engine to implement Binary Search.

Aim: Description and Implementation of Identity Management in Amazon Web Services.

Title: Identity Access Management

THEORY:

Amazon Web Services (AWS): Amazon Web Services cloud provides a secure virtual platform where users can deploy their applications. Compared to an on-premises environment, AWS security provides a high level of data protection at a lower cost to its users.

What is AWS Security?

Cloud security is the highest priority in AWS. When you host your environment in the cloud, you can be assured that it's hosted in a data center or in a network architecture that's built to meet the requirements of the most security-sensitive organization. Additionally, this high level of security is available on a pay-as-you-go basis, meaning there is really no upfront cost, and the cost for using the service is a lot cheaper compared to an on-premises environment. There are many types of security services available but some of them are widely used by AWS, such as:

- IAM
- Key Management System (KMS)
- Cognito
- Web Access Firewall (WAF)

What is IAM?

- AWS Identity and Access Management (IAM) is a web service for securely controlling access to AWS resources.
- It enables you to create and control services for user authentication or limit access to a certain set of people who use your AWS resources.

How Does IAM Work?

The IAM workflow includes the following six elements:

Principal:

1. A principal is an entity that can perform actions on an AWS resource. A user, a role or an application can be a principal.

2. Authentication: Authentication is the process of confirming the identity of the principal trying to access an AWS product. The principal must provide its credentials or required keys for authentication.

3. Request: A principal sends a request to AWS specifying the action and which resource should perform it.

4. Authorization: By default, all resources are denied. IAM authorizes a request only if all parts of the request are allowed by a matching policy. After authenticating and authorizing the request, AWS approves the action.

5. Actions: Actions are used to view, create, edit or delete a resource.

6. Resources: A set of actions can be performed on a resource related to your AWS account.

Components of IAM:

Users:

An IAM user is an identity with an associated credential and permissions attached to it. This could be an actual person who is a user, or it could be an application that is a user. With IAM, you can securely manage access to AWS services by creating an IAM user name for each employee in your organization. Each IAM user is associated with only one AWS account.

By default, a newly created user is not authorized to perform any action in AWS. The advantage of having one-to-one user specification is that you can individually assign permissions to each user.

Groups:

A collection of IAM users is an IAM group. You can use IAM groups to specify permissions for multiple users so that any permissions applied to the group are applied to the individual users in that group as well. Managing groups is quite easy. You set permissions for the group, and those permissions are automatically applied to all the users in the group. If you add another user to the group, the new user will automatically inherit all the policies and the permissions already assigned to that group. This lessens the administrative burden.

Policies:

An IAM policy sets permission and controls access to AWS resources. Policies are stored in AWS as JSON documents. Permissions specify who has access to the resources and what actions they can perform. For example, a policy could allow an IAM user to access one of the buckets in Amazon S3. The policy would contain the following information:

- Who can access it

- What actions that user can take

- Which AWS resources that user can access

- When they can be accessed

There are two types of policies:

1) Managed policies:

A managed policy is a default policy that you attach to multiple entities (users, groups, and roles) in your AWS account. Managed policies, whether they are AWS-managed or customer-managed, are stand-alone identity-based policies attached to multiple users and/or groups.

2) Inline policies:

Inline policies are policies that you create that are embedded directly into a single entity (user, group or role).

Roles:

An IAM role is a set of permissions that define what actions are allowed and denied by an entity in the AWS console. It is similar to a user in that it can be accessed by any type of entity (an individual or AWS service). Role permissions are temporary credentials.

Features of IAM:

▪ Shared access to the AWS account. The main feature of IAM is that it allows you to create separate usernames and passwords for individual users or resources and delegate access. ▪ Granular permissions. Restrictions can be applied to requests. For example, you can allow the user to download information, but deny the user the ability to update information through the policies.

▪ Multifactor authentication (MFA). IAM supports MFA, in which users provide their username and password plus a one-time password from their phone—a randomly generated number used as an additional authentication factor.

▪ Identity Federation. If the user is already authenticated, such as through a Facebook or Google account, IAM can be made to trust that authentication method and then allow access based on it. This can also be used to allow users to maintain just one password for both onpremises and cloud environment work.

▪ Free to use. There is no additional charge for IAM security. There is no additional charge for creating additional users, groups or policies.

▪ PCI DSS compliance. The Payment Card Industry Data Security Standard is an information security standard for organizations that handle branded credit cards from the major card schemes. IAM complies with this standard.

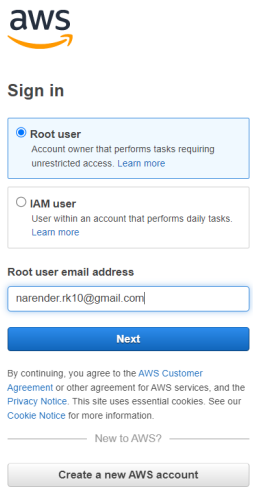
▪ Password policy. The IAM password policy allows you to reset a password or rotate passwords remotely. You can also set rules, such as how a user should pick a password or how many attempts a user may make to provide a password before being denied access.

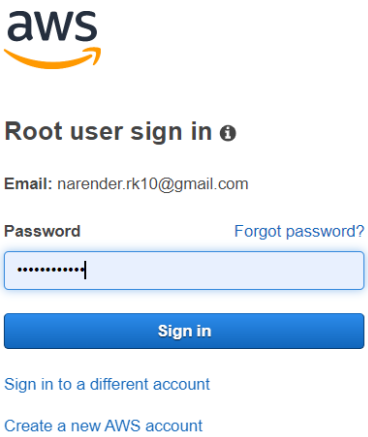
Multi-Factor Authentication (MFA):

MFA is an additional level of security process provided by AWS. Here, a user's identity is confirmed for AWS login only after performing two levels of verification. For ex.: For Email login first level of verification would be providing username and password and second level of verification will be the OTP send to Mobile.

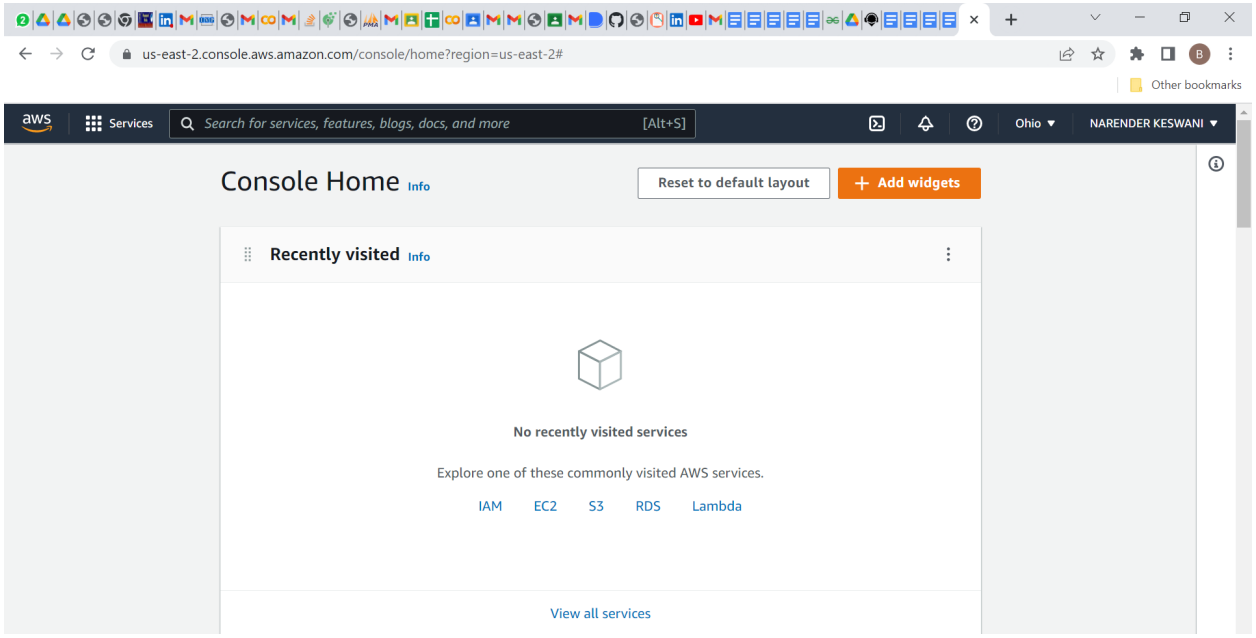
Create account on AWS:
https://portal.aws.amazon.com/billing/signup#/start/email

After creating account, Sign in on AWS:
https://signin.aws.amazon.com/signin

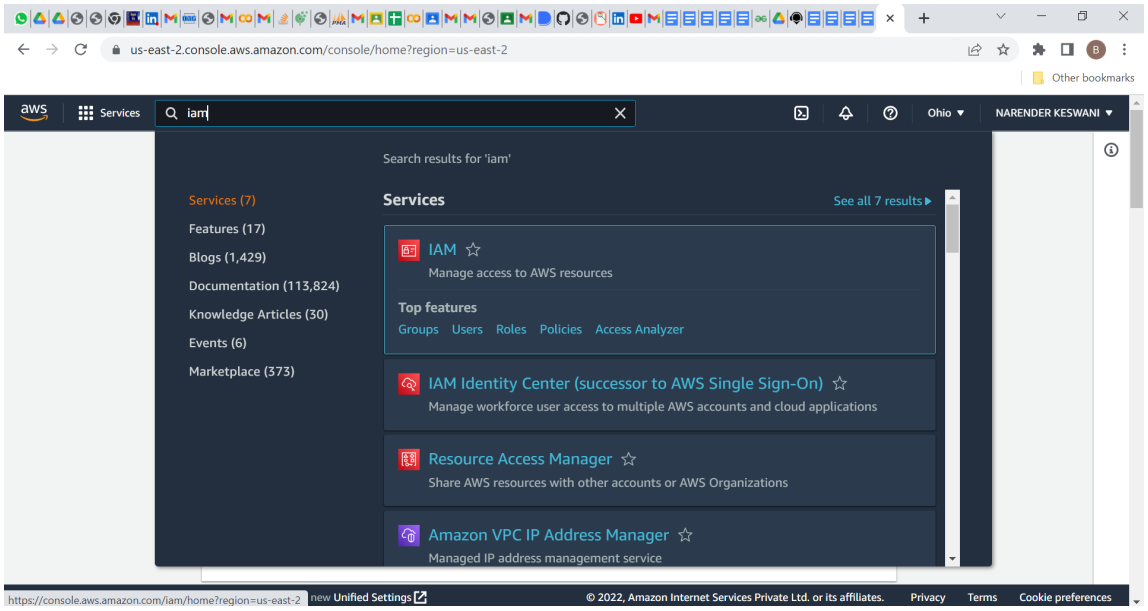


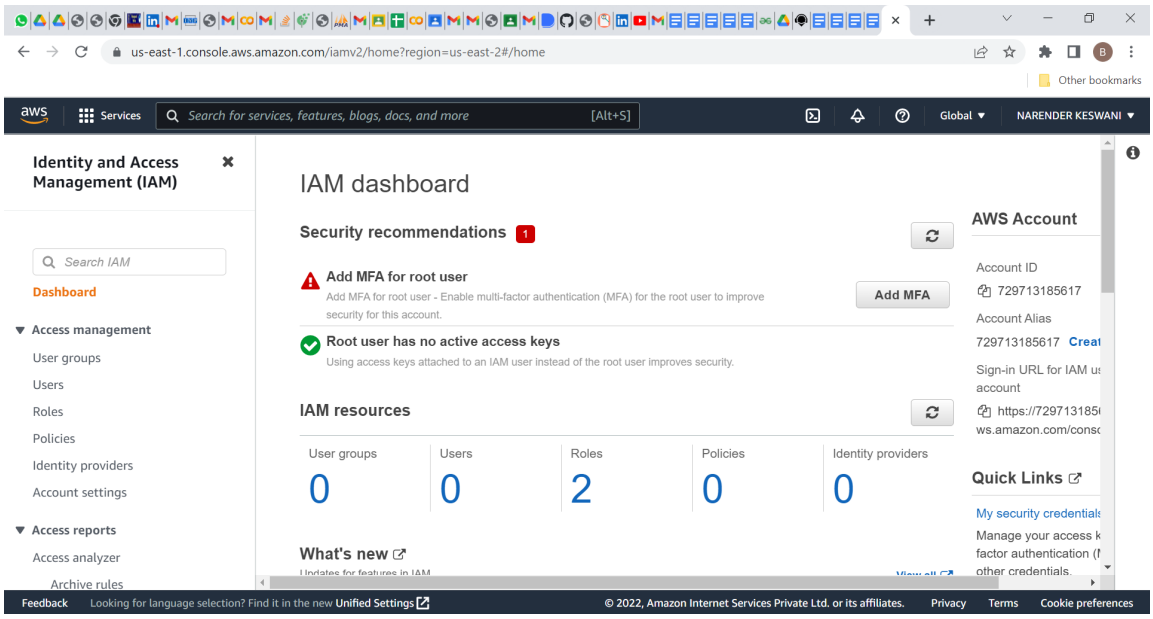


After login, you will be redirected to AWS Dashboard:

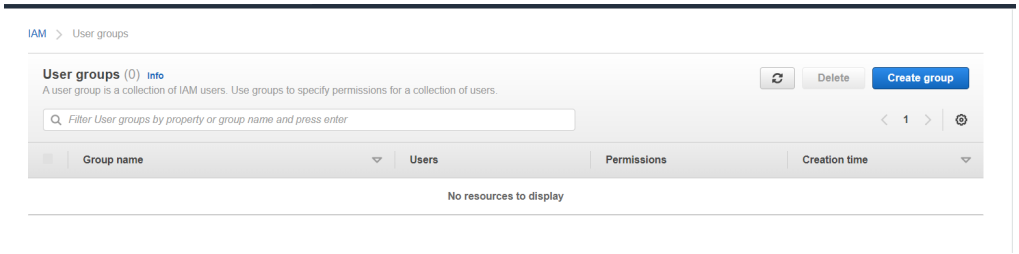


In search bar of service section, search “IAM”:
And click on IAM:

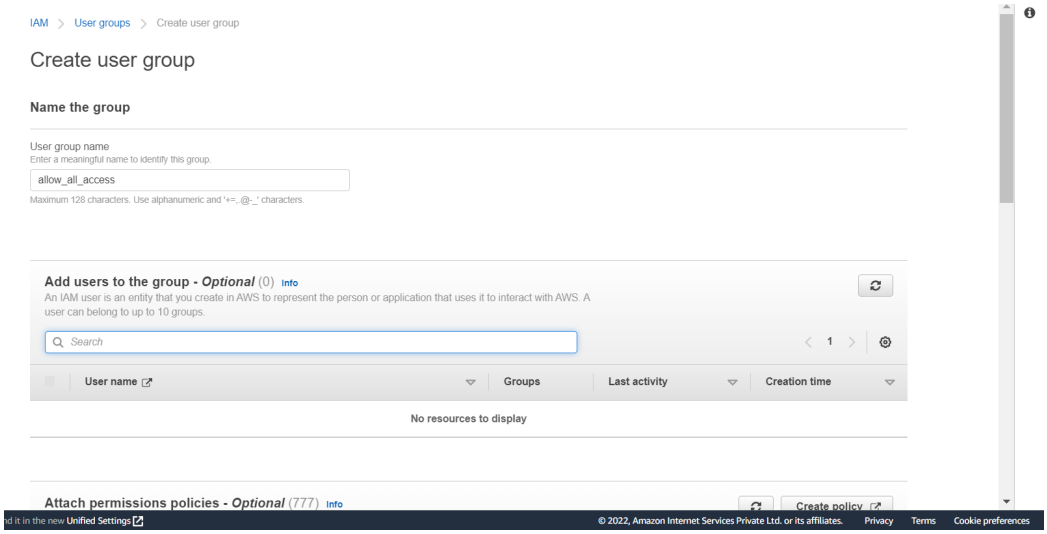




Click on user groups:

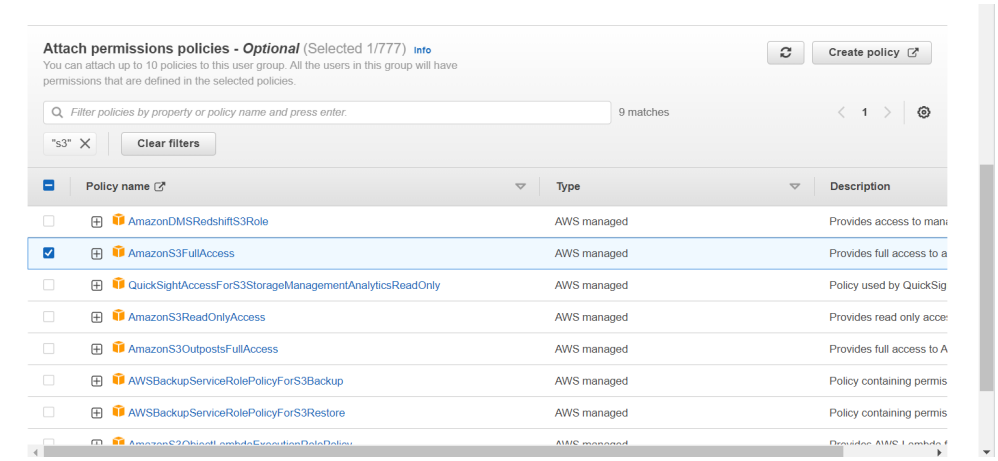


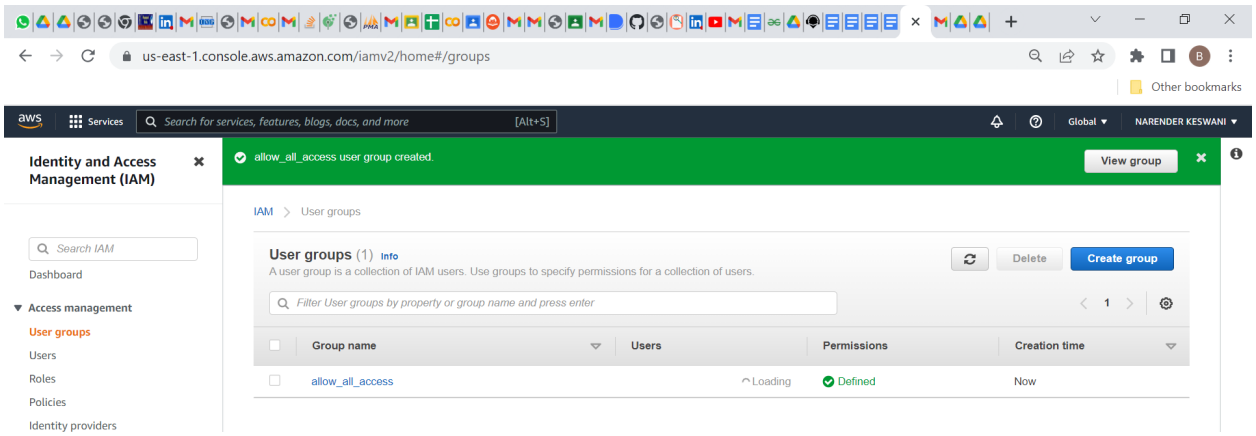
Click on Create Group
Give unique & meaningful name to the group: [allow_all_access]



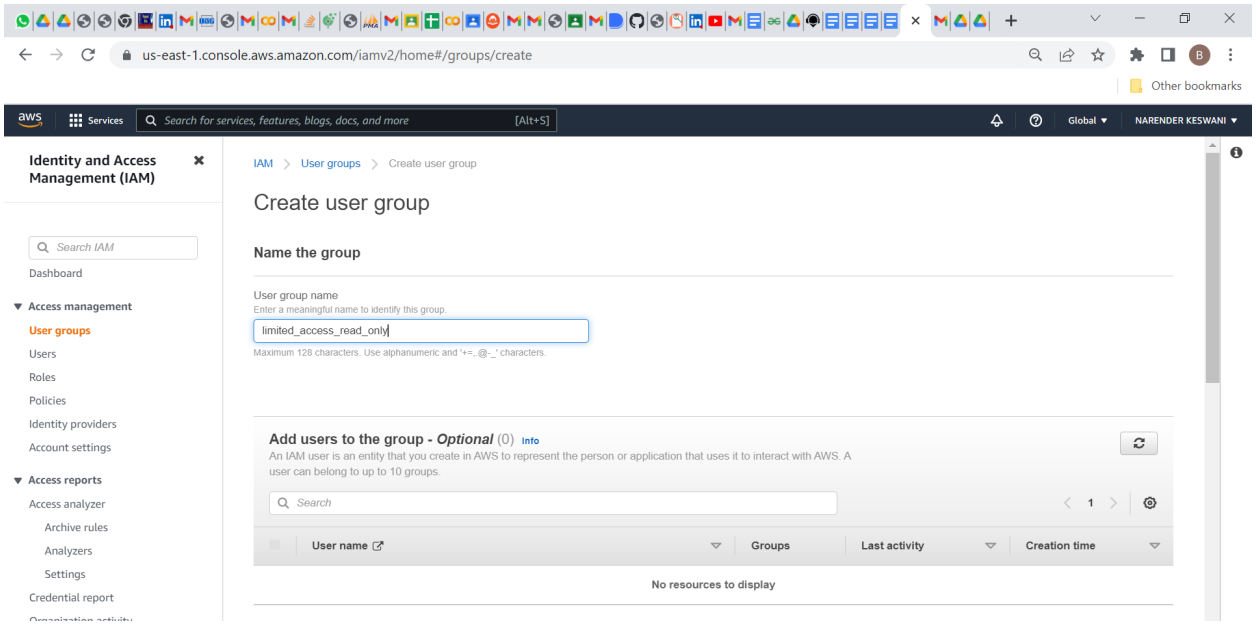
Select attach permissions policies

Select “AmazonS3FullAccess” [Reason: allowing to do all things such as create, update, delete, read s3 bucket]

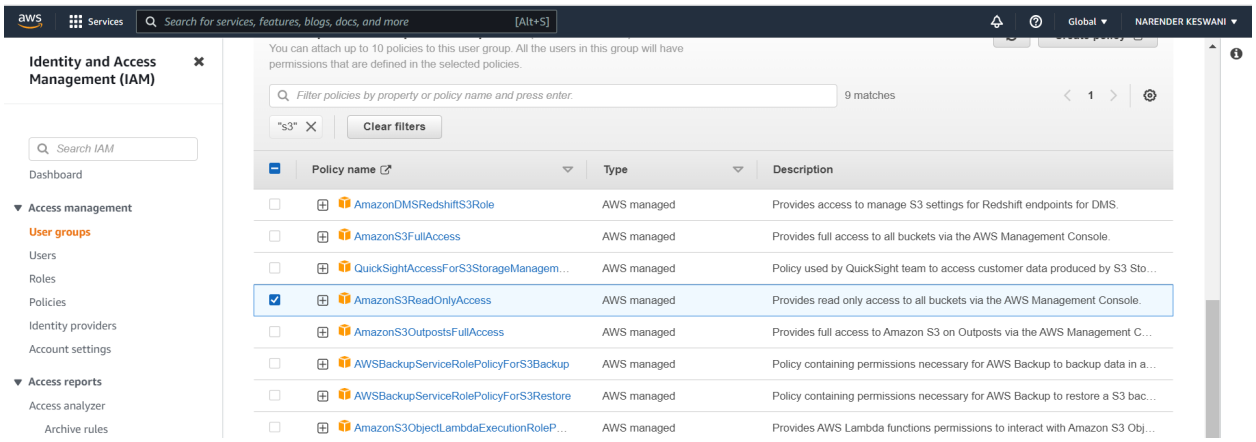




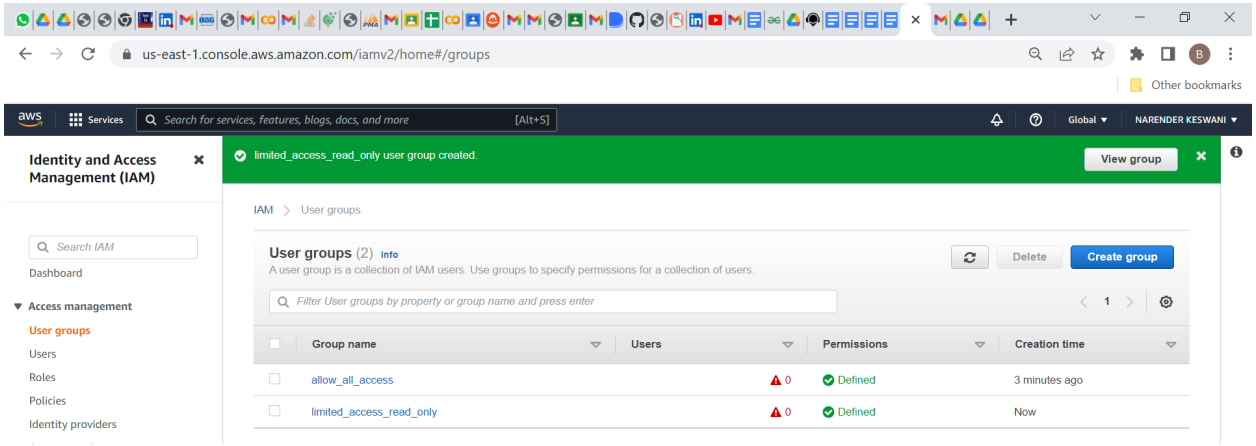
Create another group which have only limited to access: Give unique & meaningful name to the group: [limited_access_read_only]



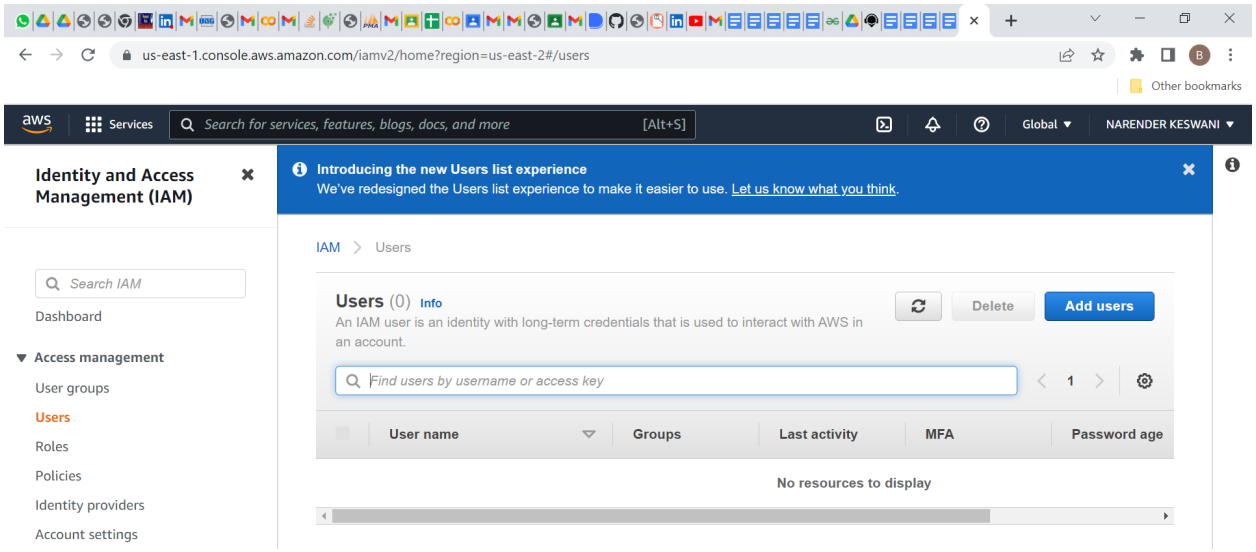
Select attach permissions policies Select “AmazonS3ReadyOnlyAccess” [Reason: only allowed to read s3 bucket]



Check on dashboard: [whether the groups are created successfully or not]



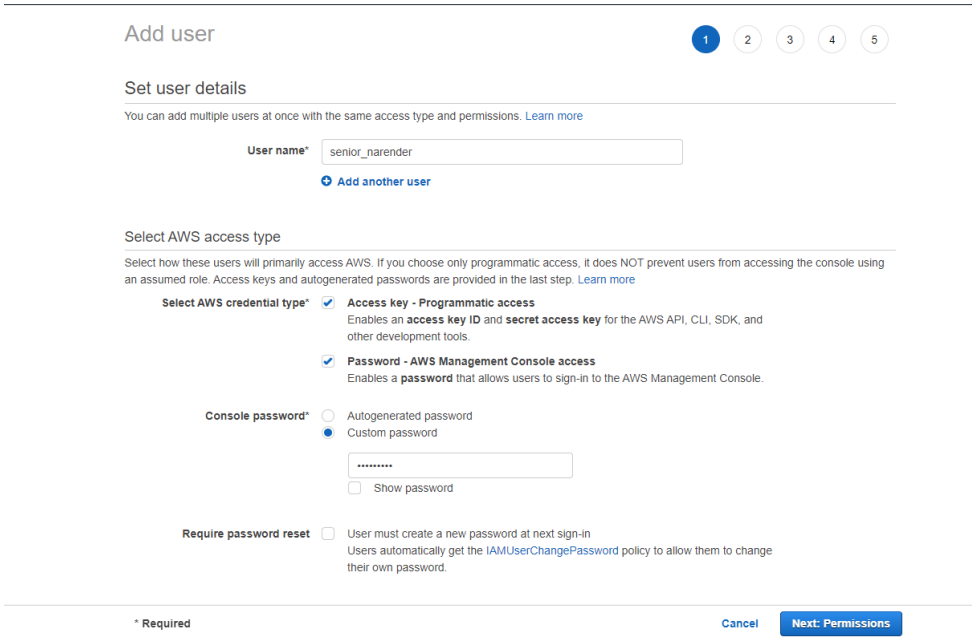
Now, click on the Users sections:



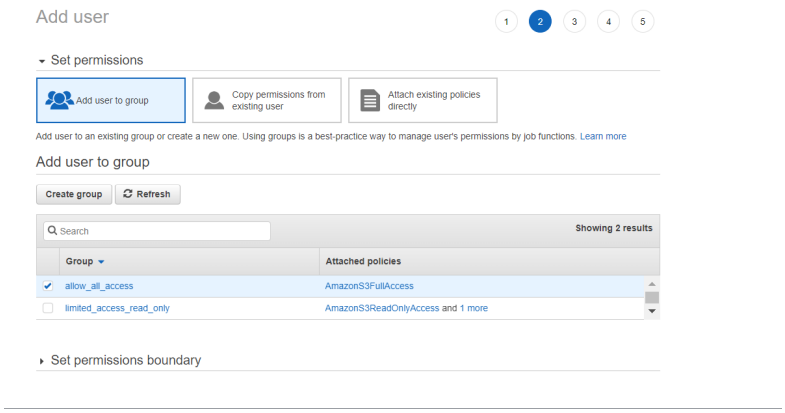
Click on “Add User”:

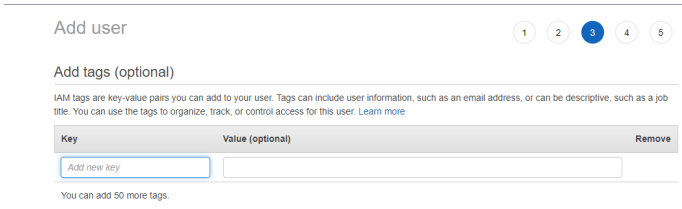
We need 2 users for demonstration of the Identity Access Management(IAM) & S3 bucket:

Create user “senior_narender”:

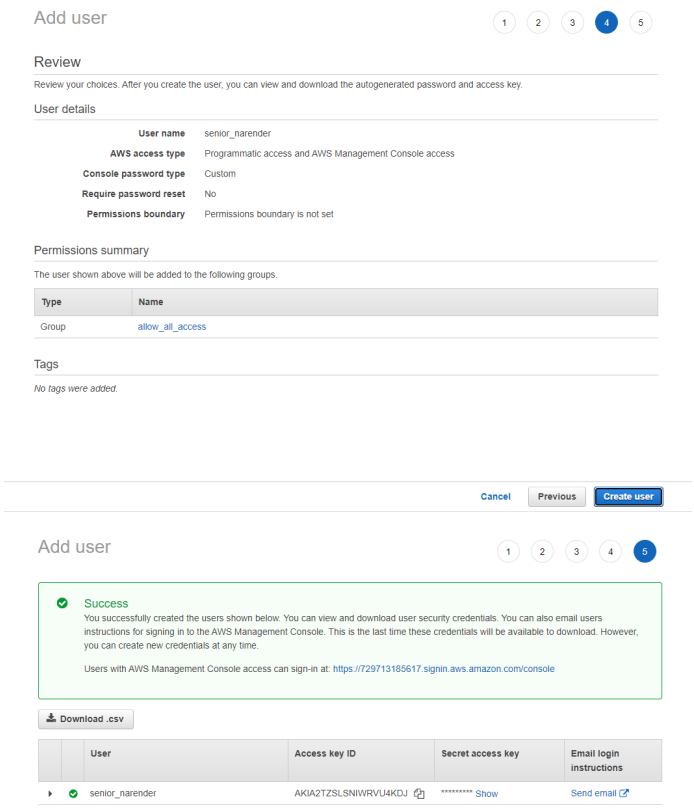


Add user to group: select “allow_all_access” [reason to choose this: senior will have the access to all s3 bucket services]

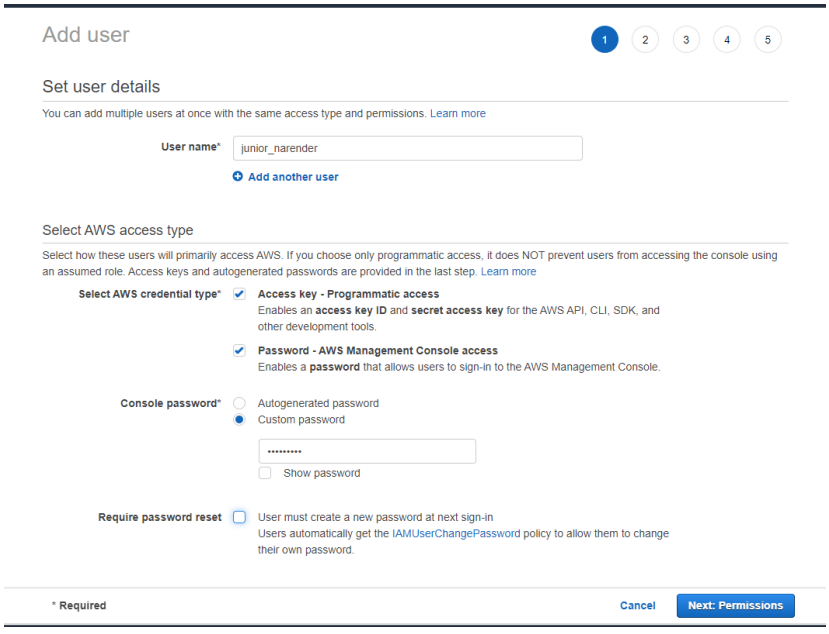




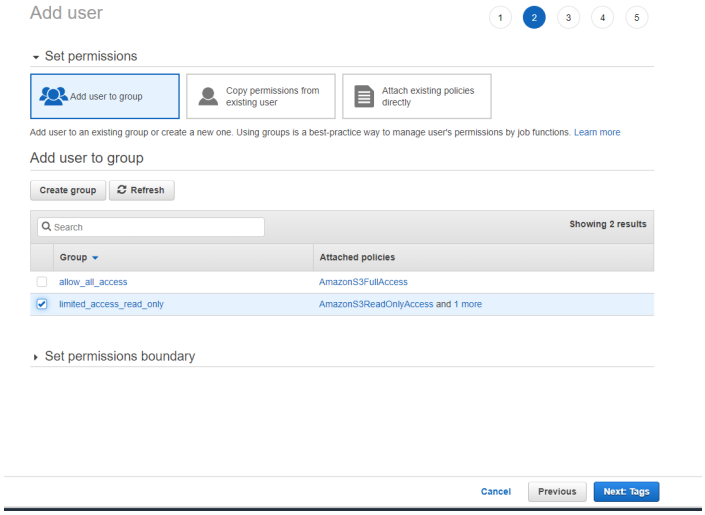
Click on create user:

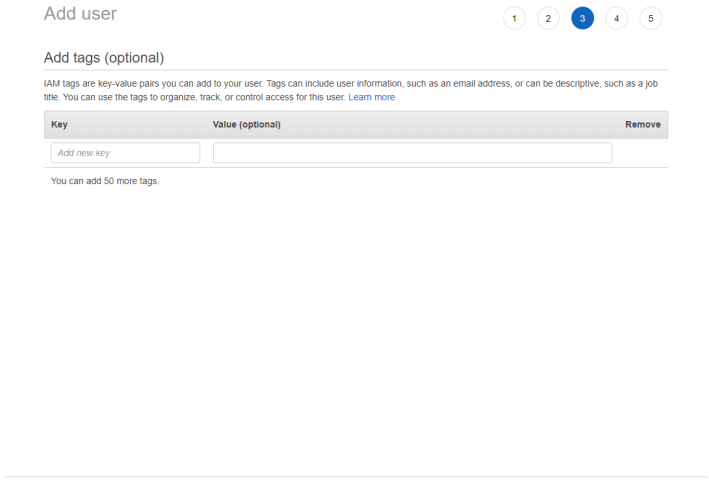


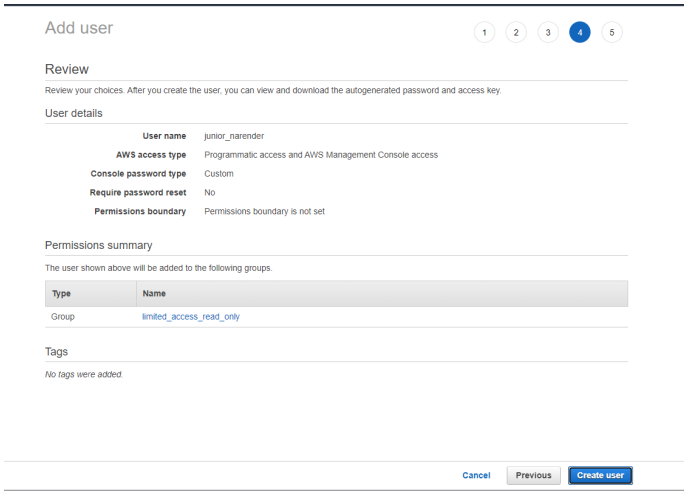
Click another user “junior_narender”:



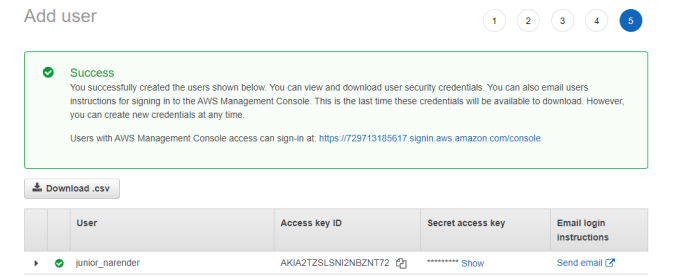
Add user to group: select “limited_access_ready_only” [reason to choose this: junior will have the limited access to s3 bucket services (only read)]





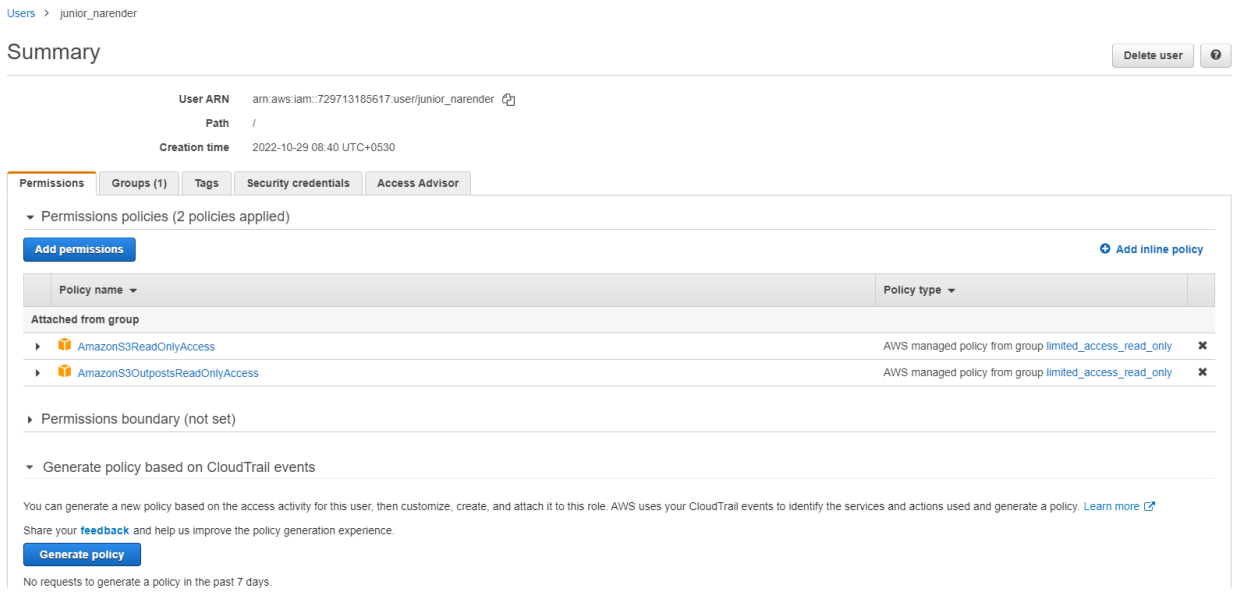


Click on the “Create User”

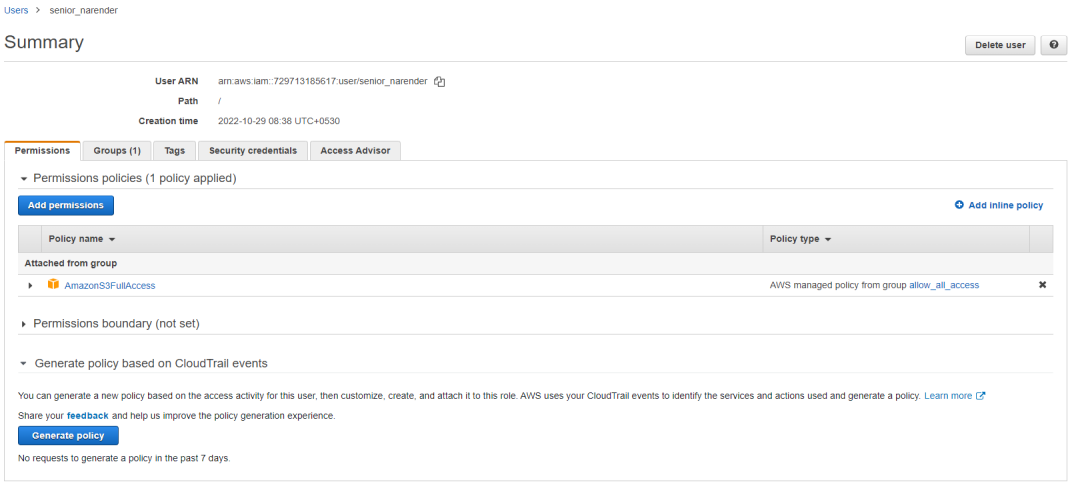


Check permissions [navigate to users section]:

Junior_narender:

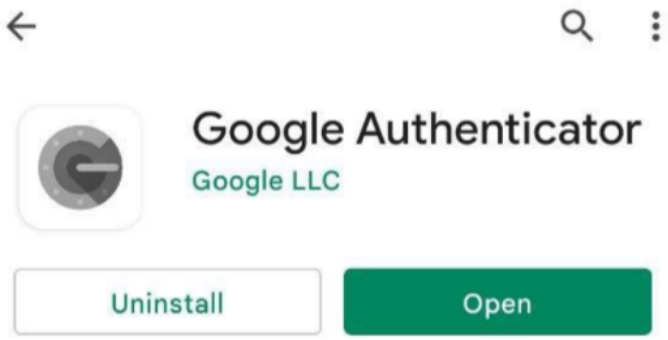


Senior_narender:



MFA - Multi Factor Authentication

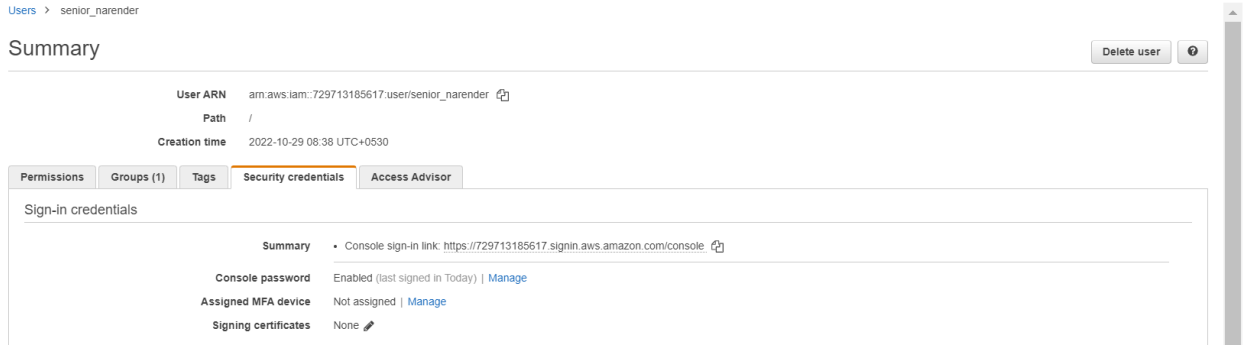
On your mobile Install Google Authenticator in google playstore



Attaching MFA to senior_narender

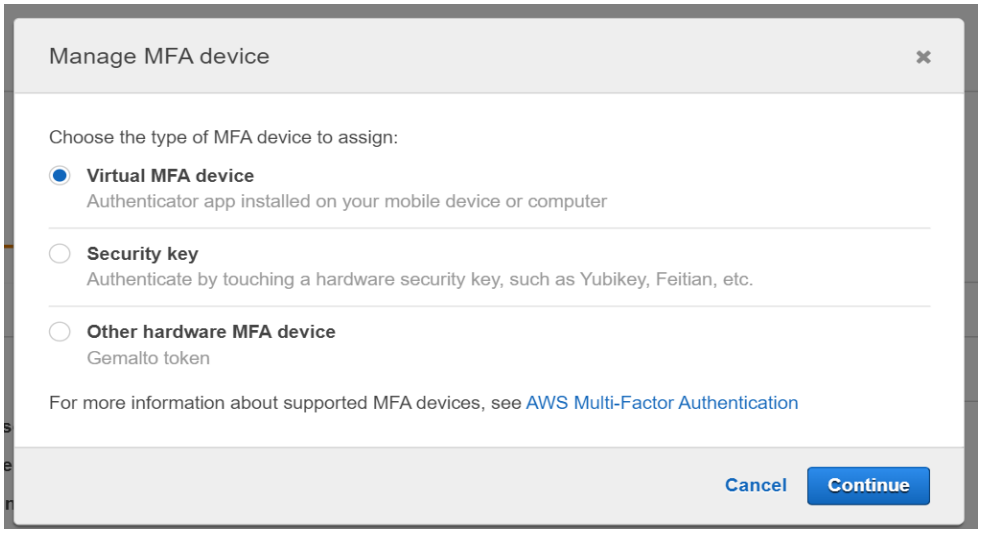
Go to senior_narender Summary [in search bar od service select "IAM", then select users section]

Click on Security credentials and go to Assigned MFA device and select virtual MFA device

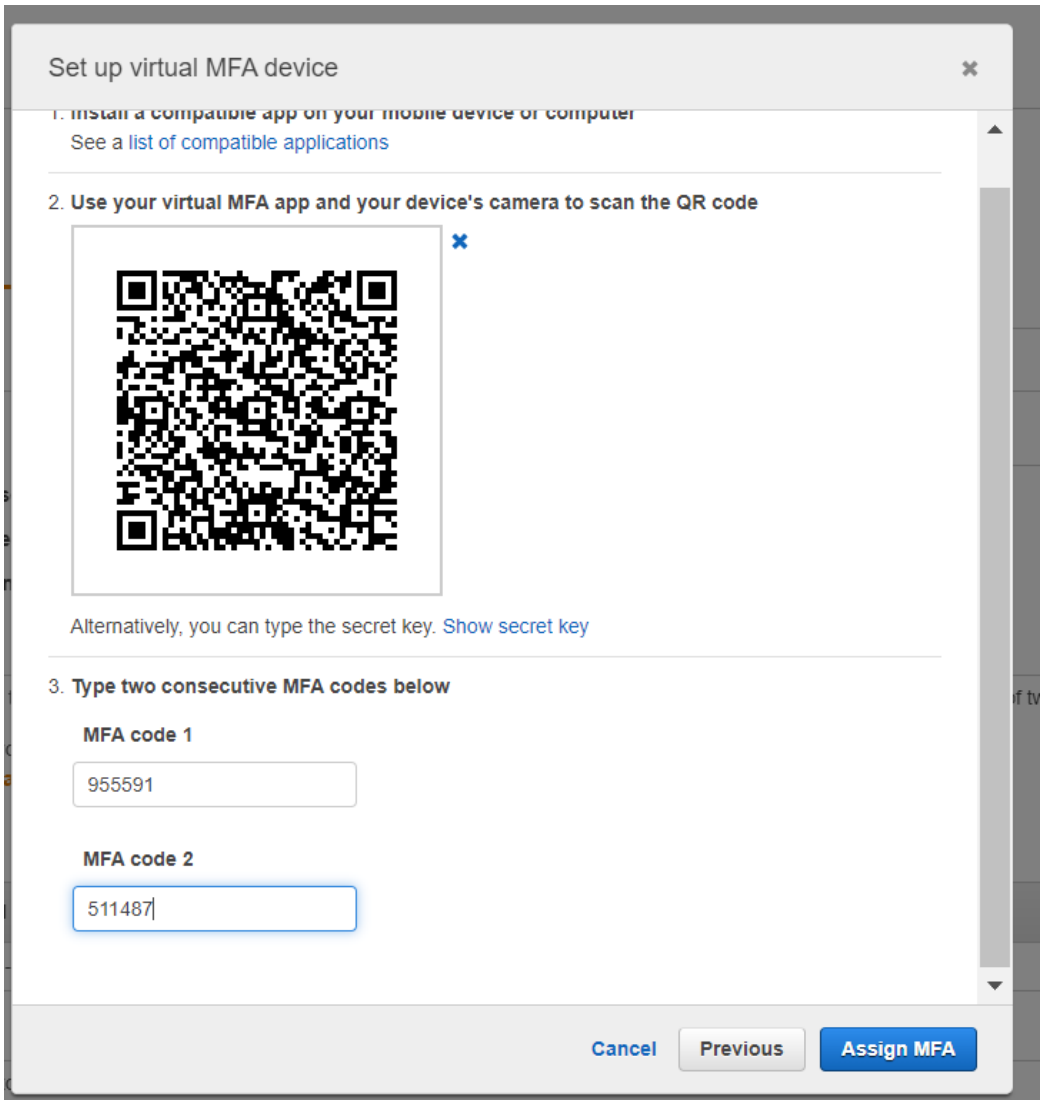


Click on the manage:

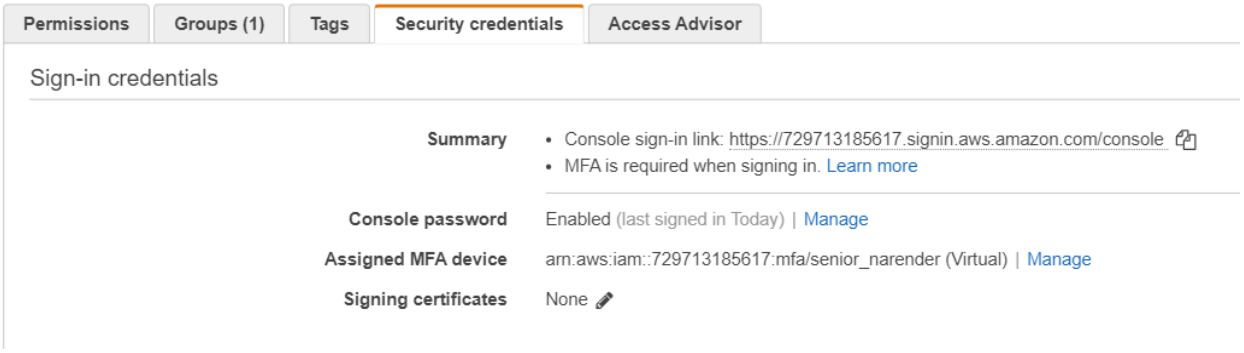
Select virtual MFA device:



Scan this QR Code in Google Authenticator & paste the MFA Code



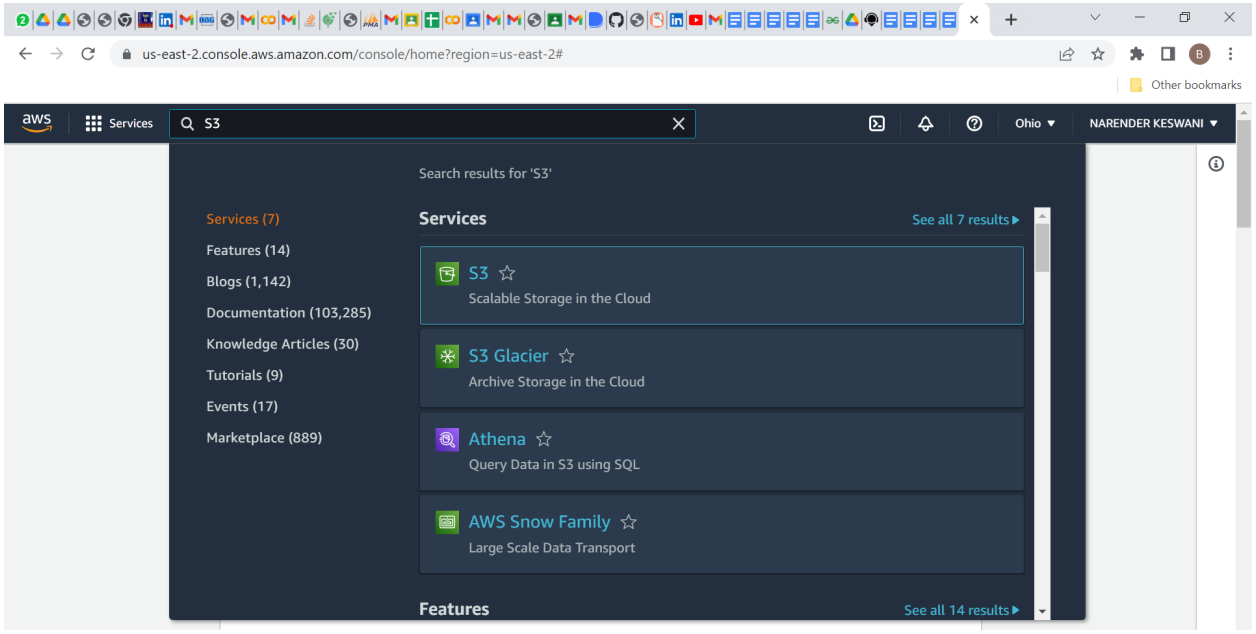
Click on the Assign MFA:

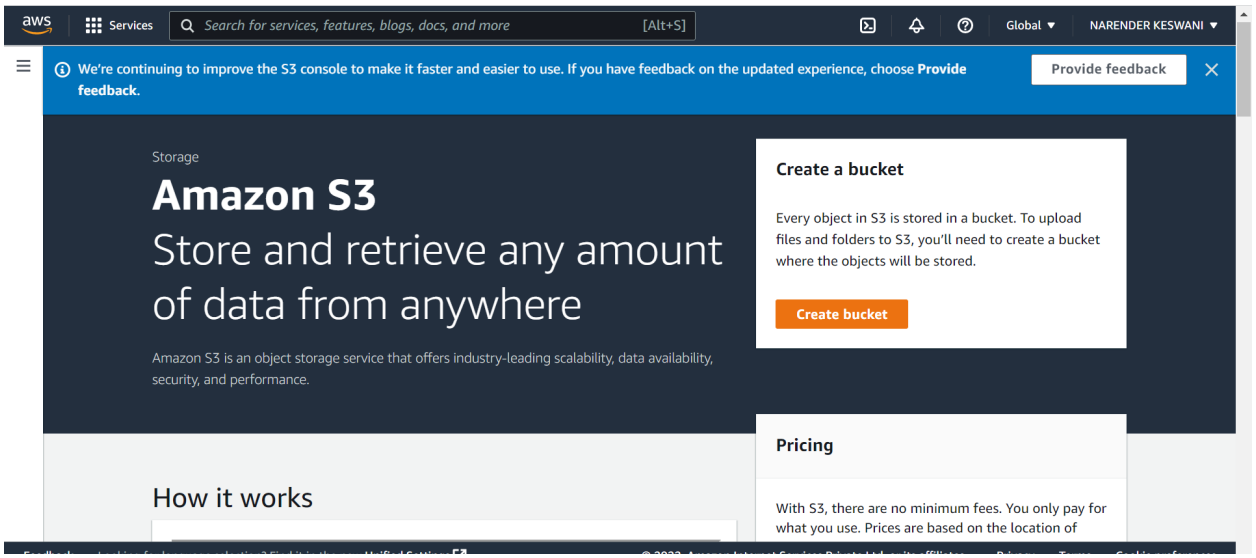


Verify whether MFA is assigned or not to senior_narender:

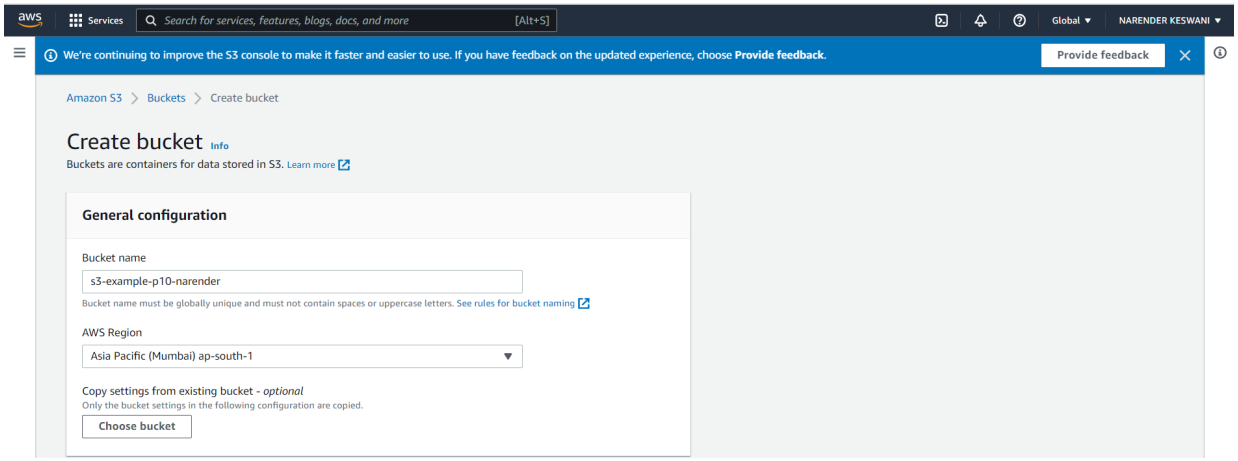
User name	Groups	Last activity	MFA
junior_narender	limited_access_read_only	3 hours ago	None
senior_narender	allow_all_access	3 hours ago	Virtual

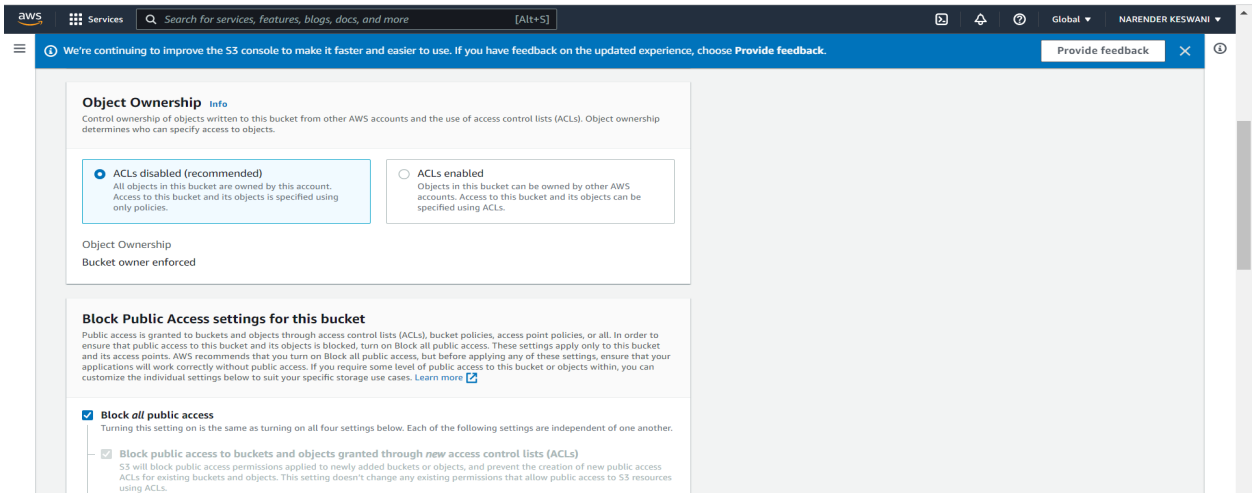
In search bar of service section, search “S3”:
And click on S3:

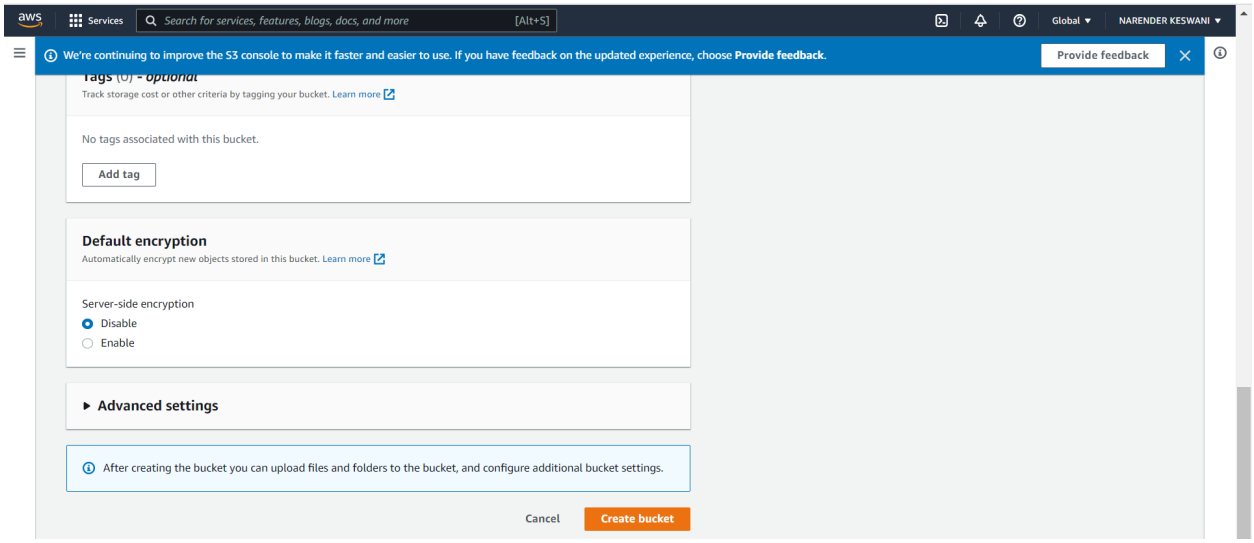




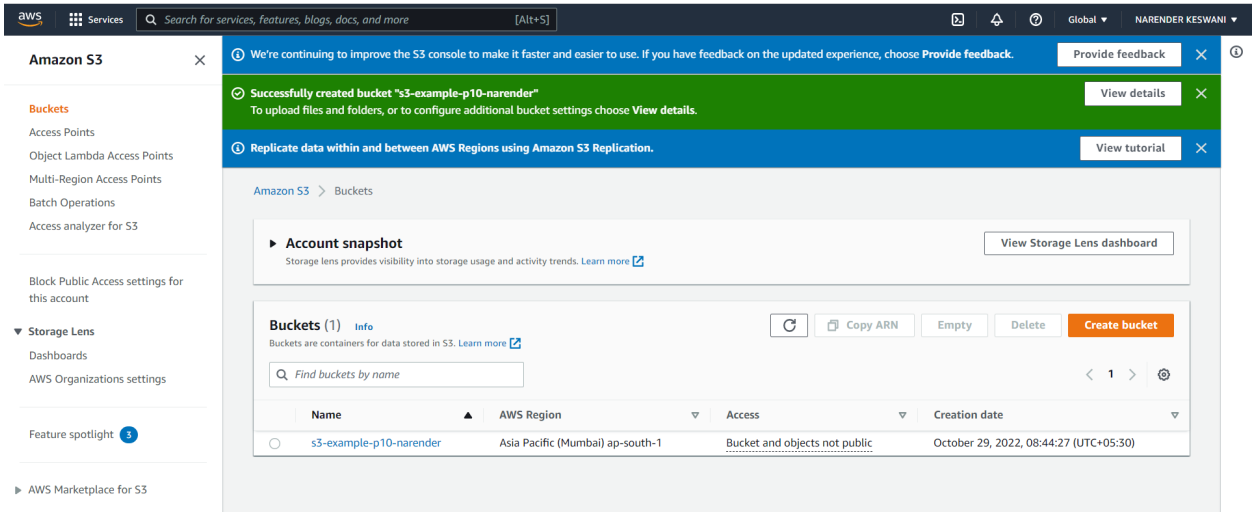
Click on the create a bucket:
Give a meaningful name to the bucket [s3-example-p10-narender]
Select any AWS Region [preferred is mumbai]





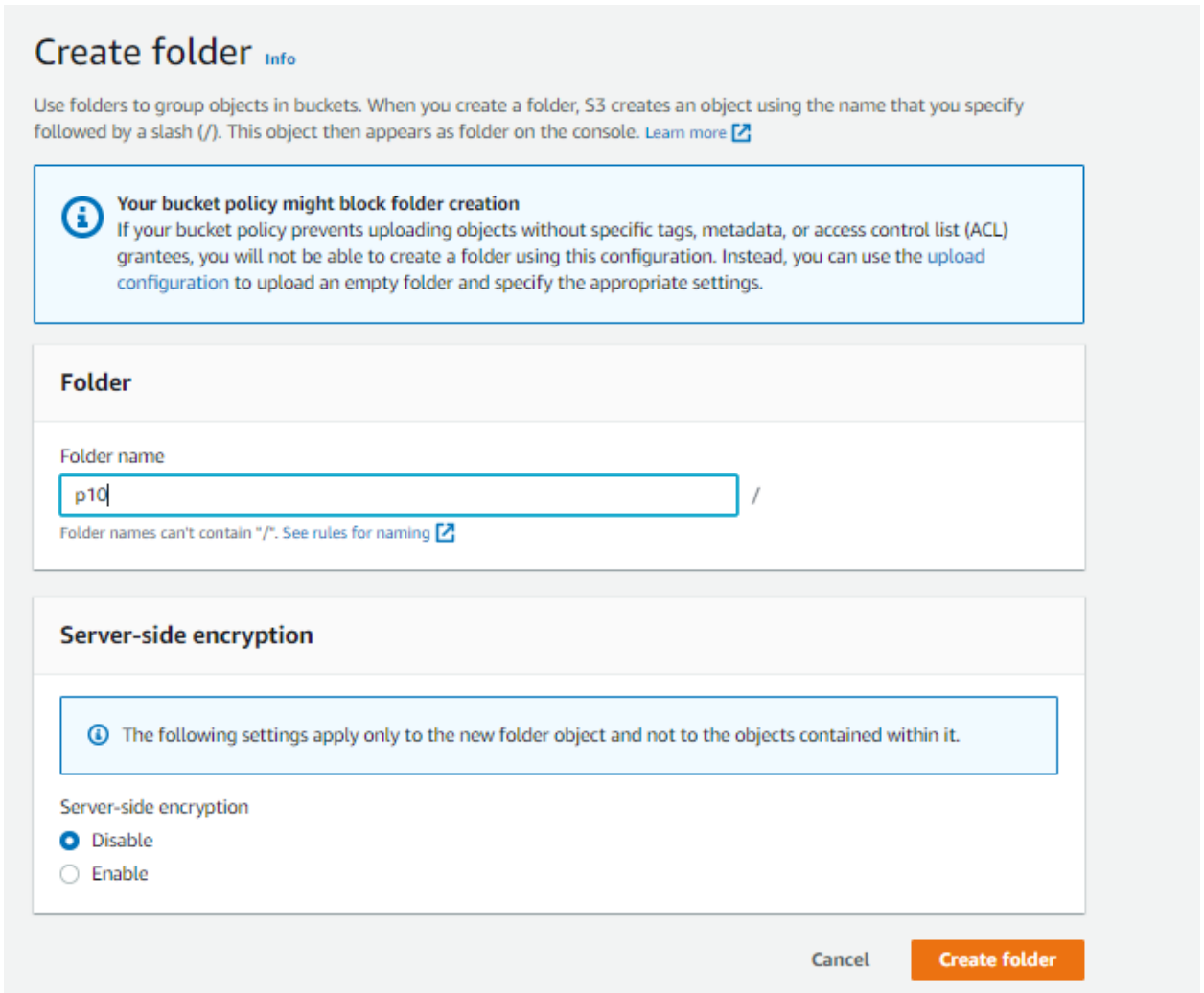


Click on create bucket:

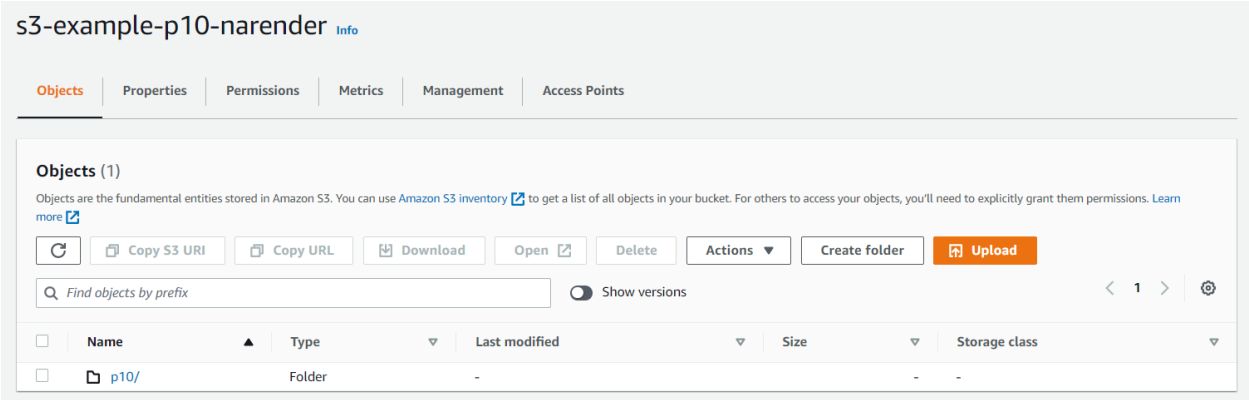


Click on the s3-example-p10-narender:

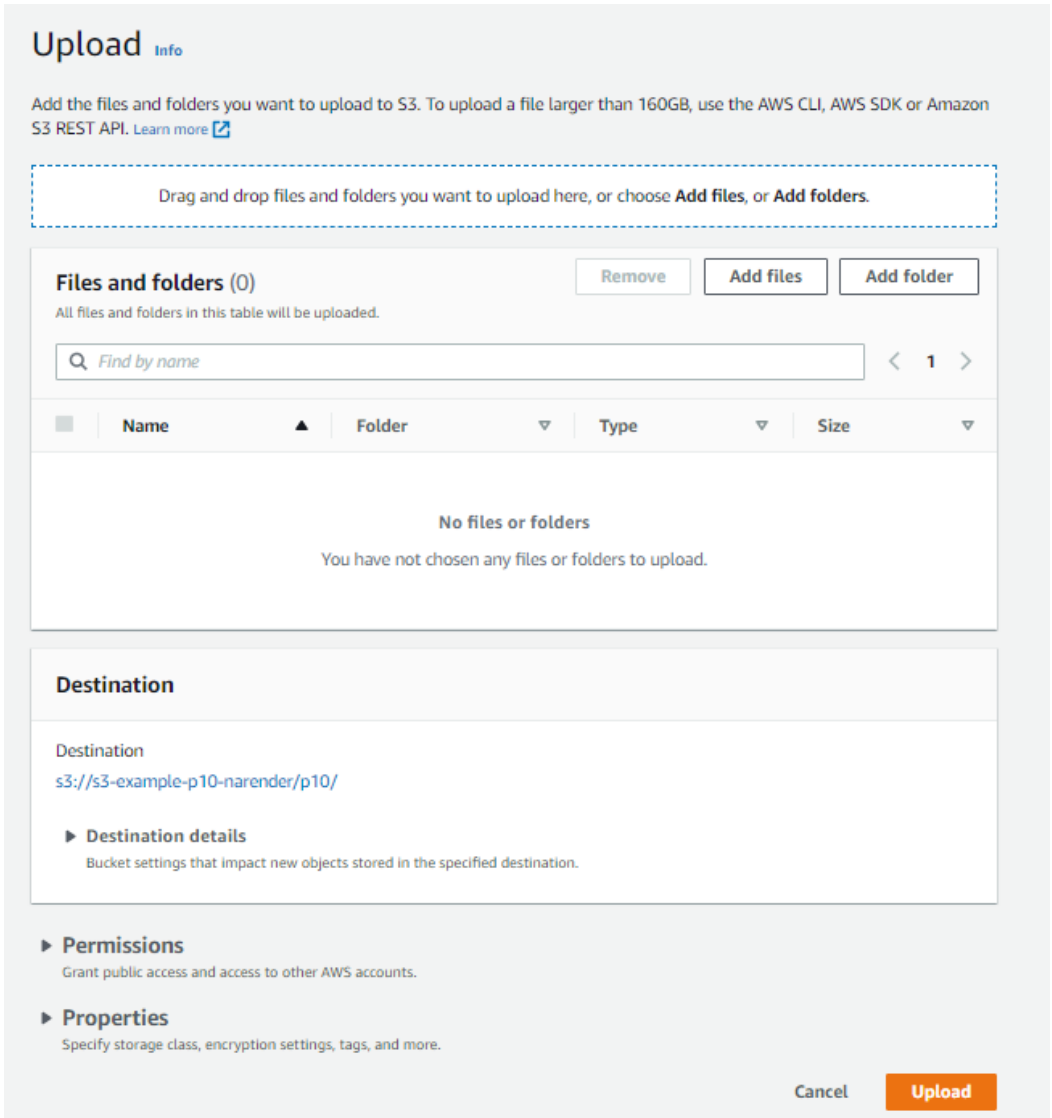
Create folder: [p10]



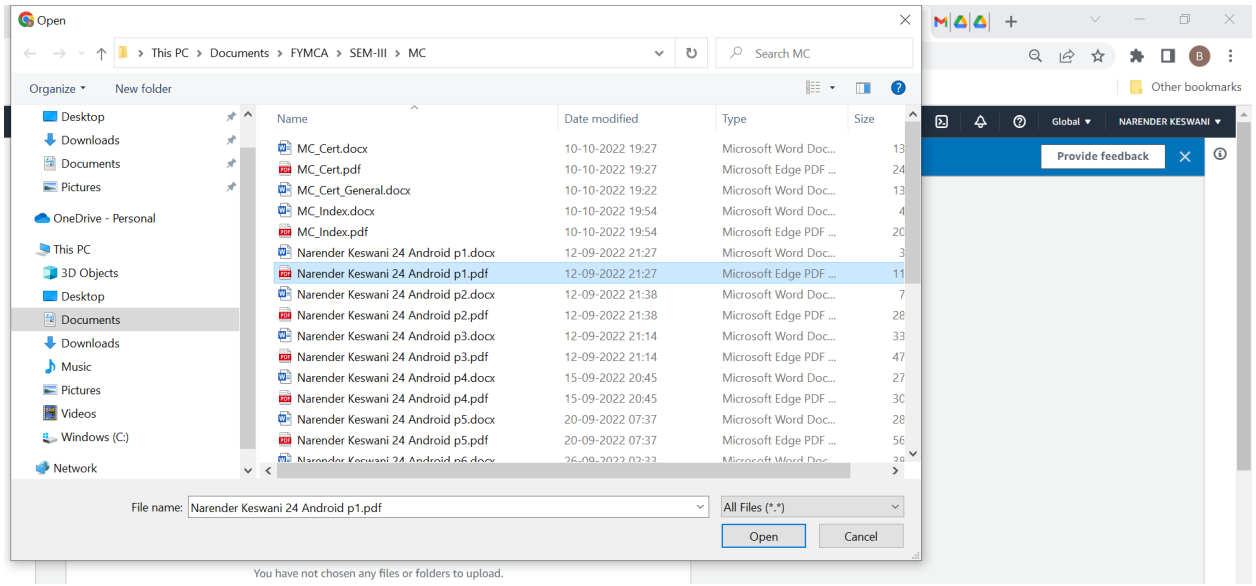
Click on the create folder:

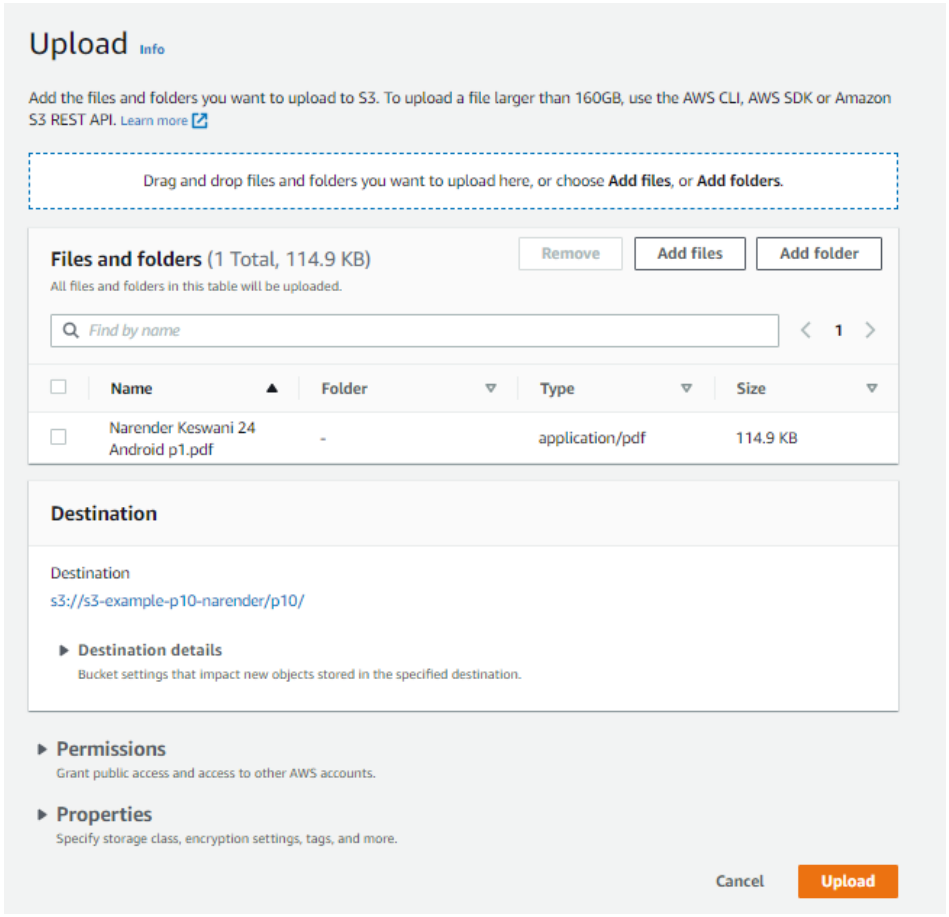


Click on upload:



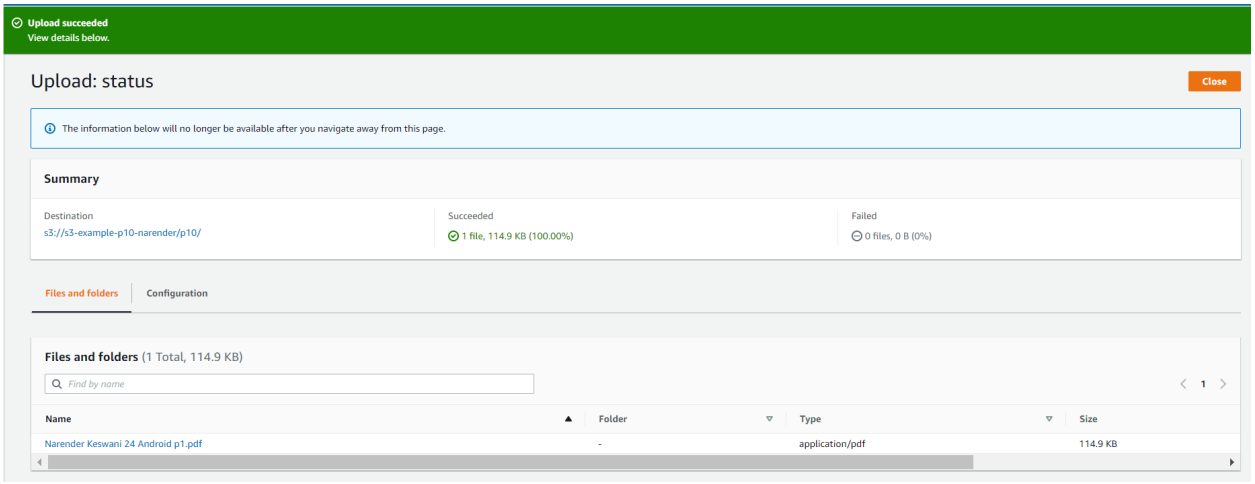
Upload files by clicking add files:





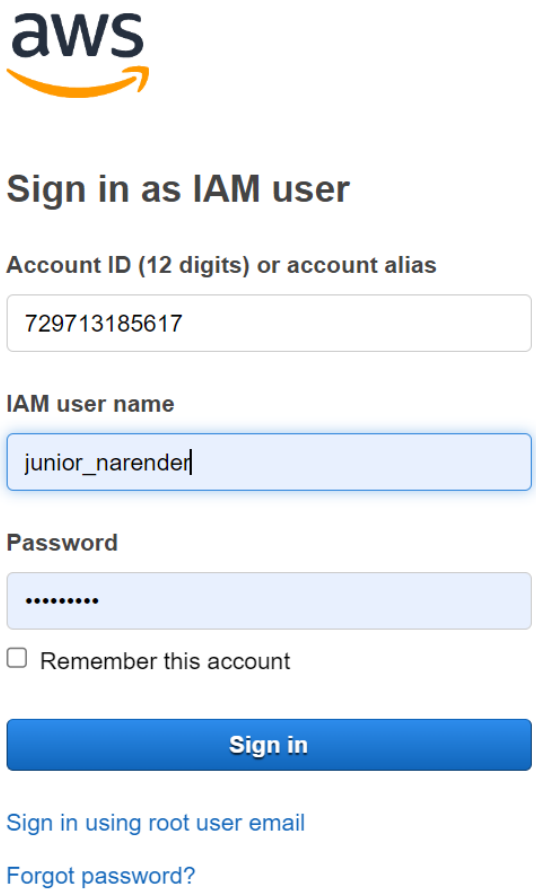
Click on upload button

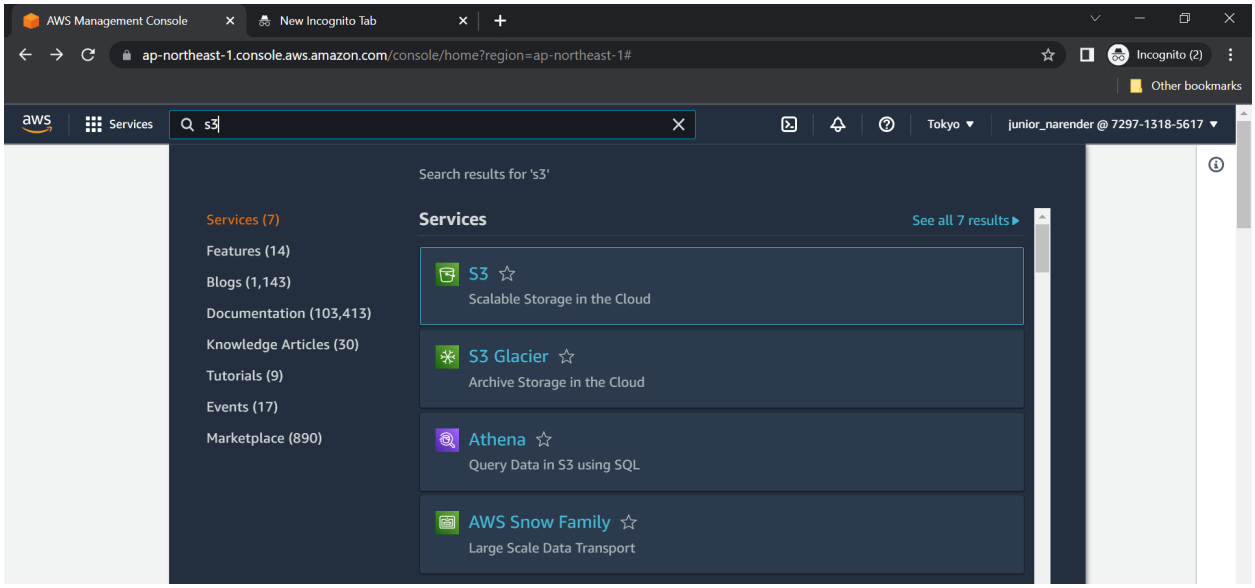
Check the status whether the file is uploaded or not:



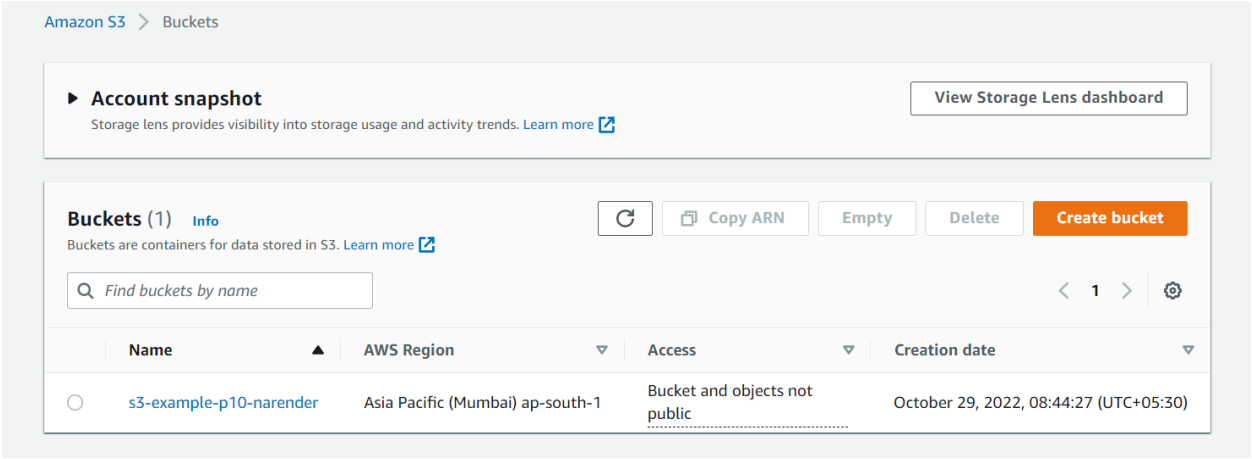
Now, check identity management on S3 bucket by logging using junior_narender & senior_narender:

A) Junior_narender login:

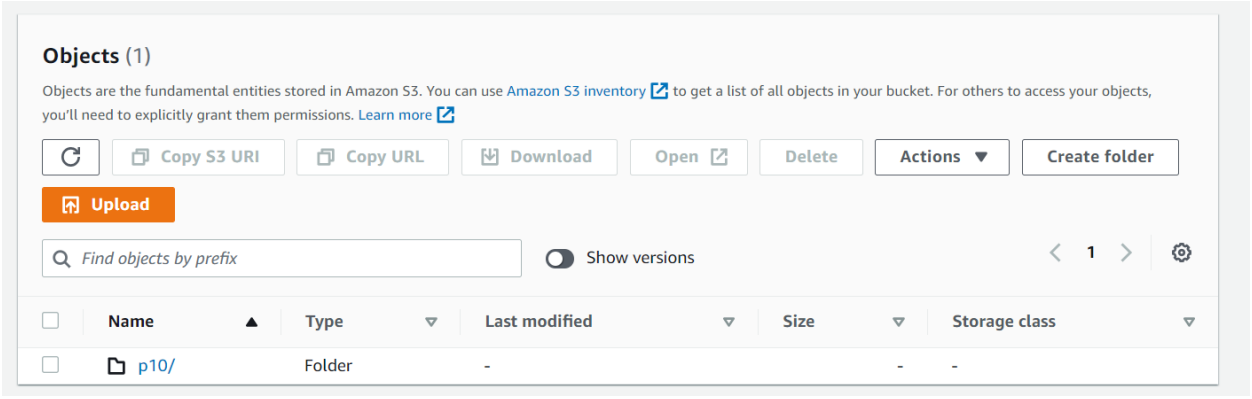




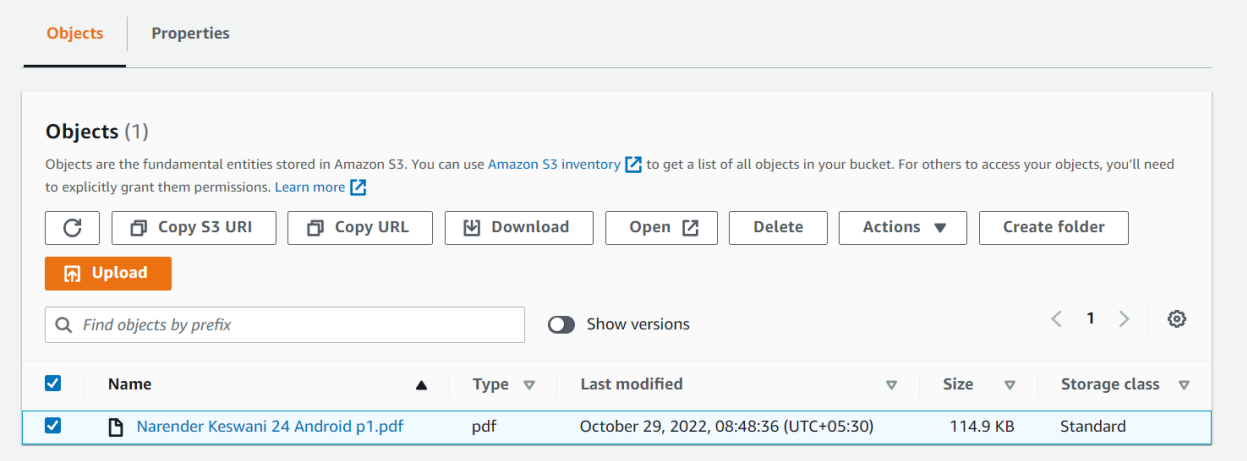
As we specified read only permission for junior_narender:
Now we try to delete s3 bucket/objects/files:



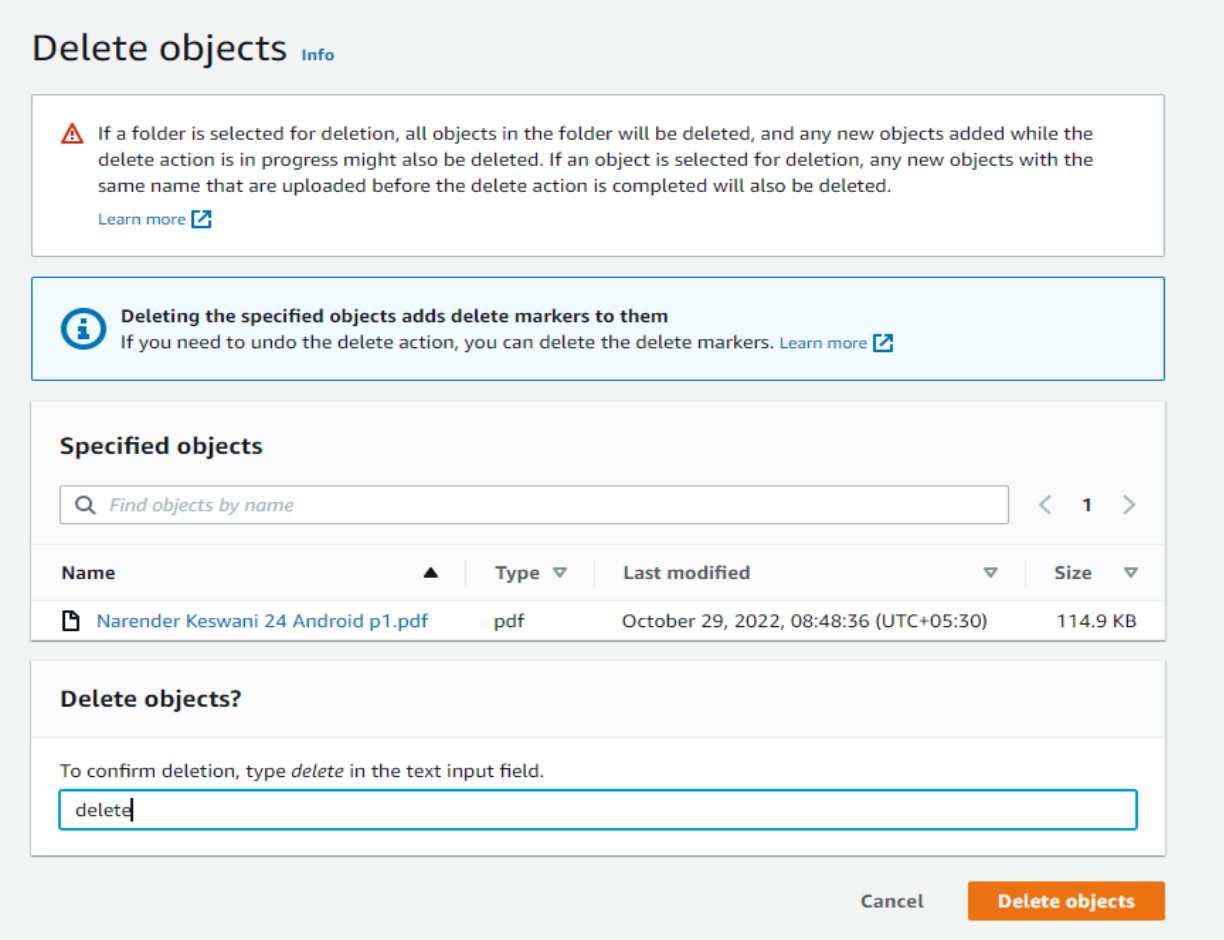
Click on s3-example-p10-narender:



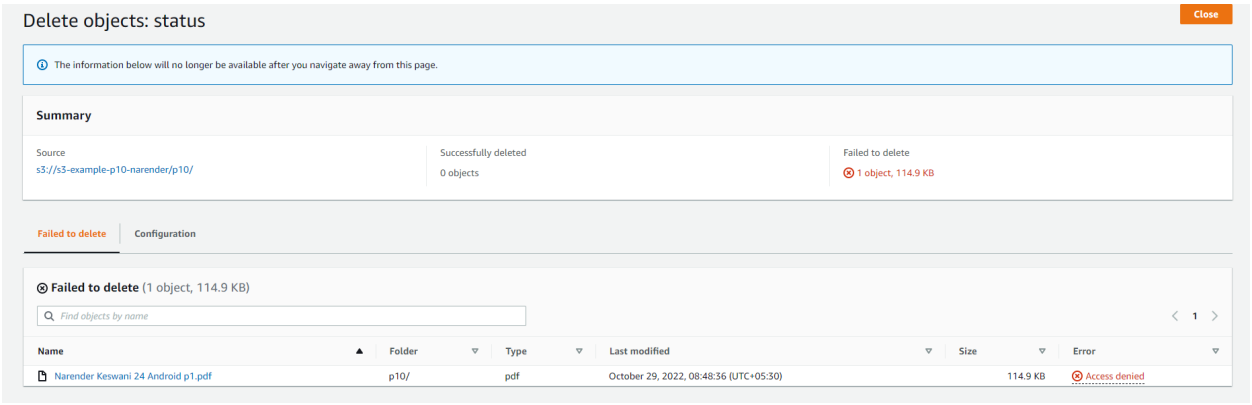
Click on p10:



Select “Narender Keswani 24 Android p1.pdf”:
Now click on the “Delete”:

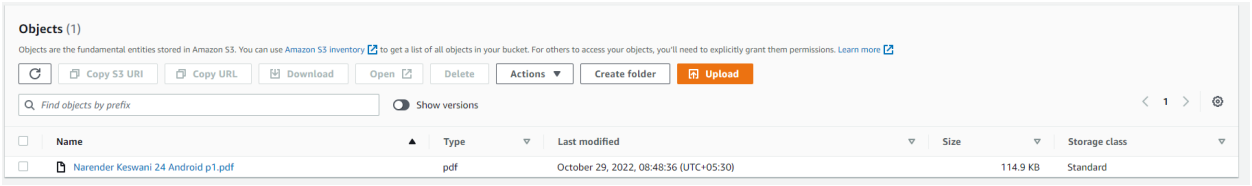


Type “delete” in textbox for confirmation:
Click on the delete objects:

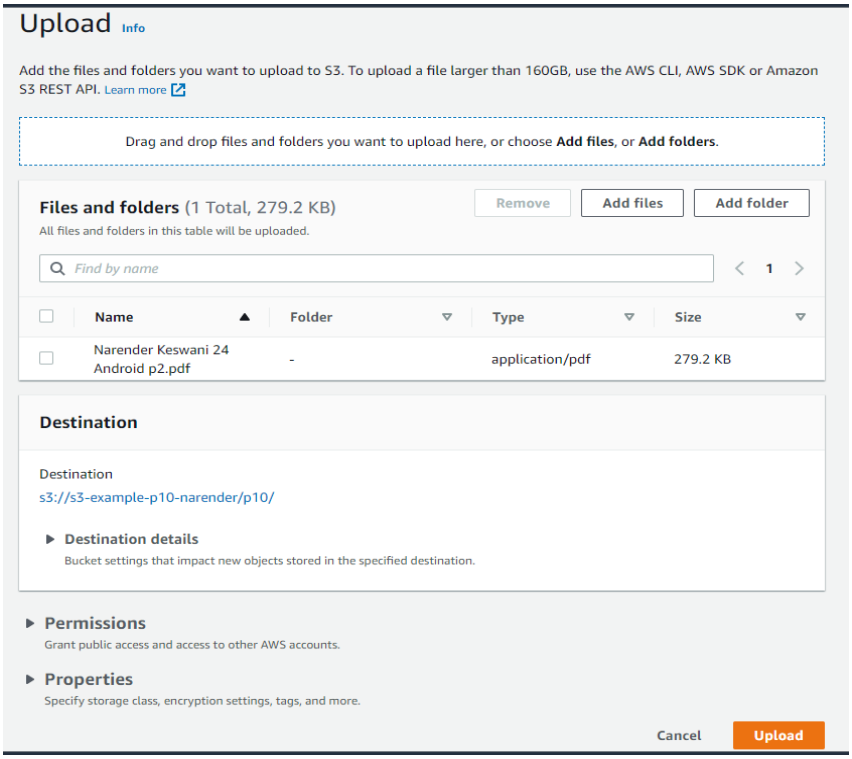


The object can’t be deleted because junior_narender does not have the access/permission

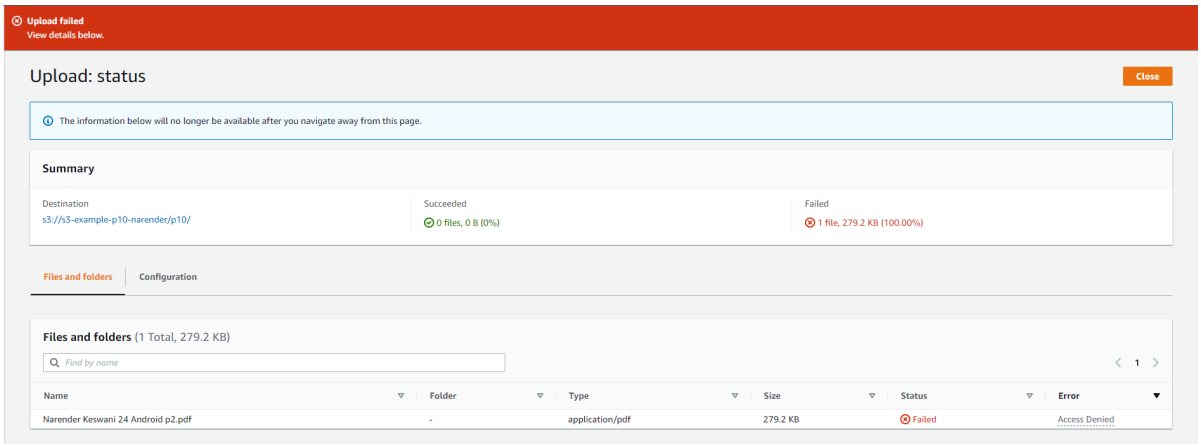
Now we will try to upload files in junior_narender:



Click on the upload button:

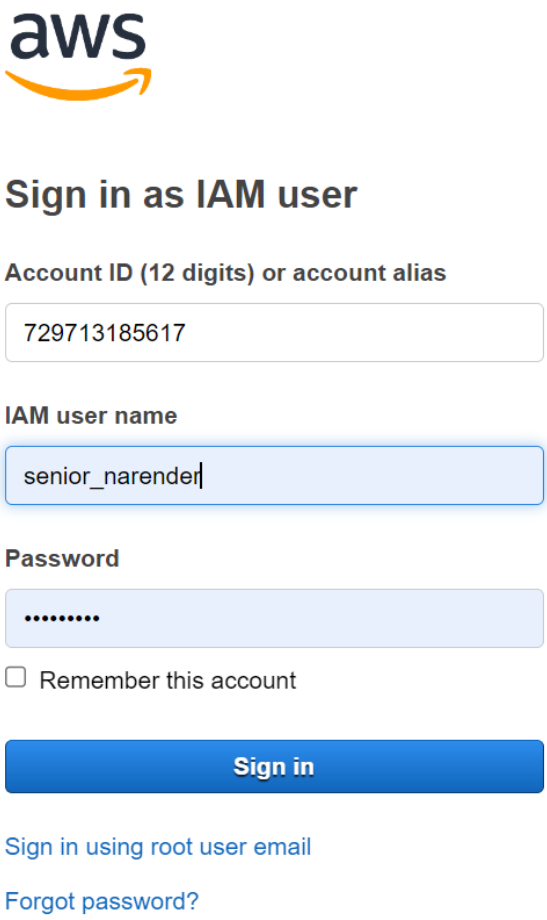


Add files & click on the upload:

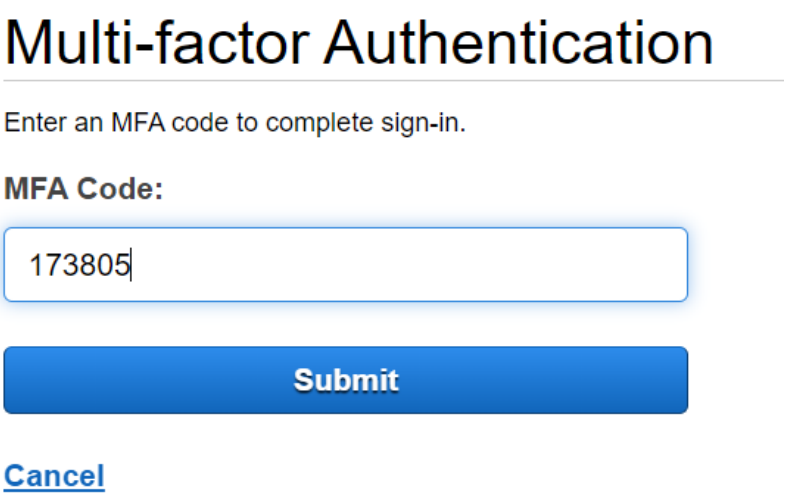


As the junior_narender does not have the permission to upload, the file upload failed.

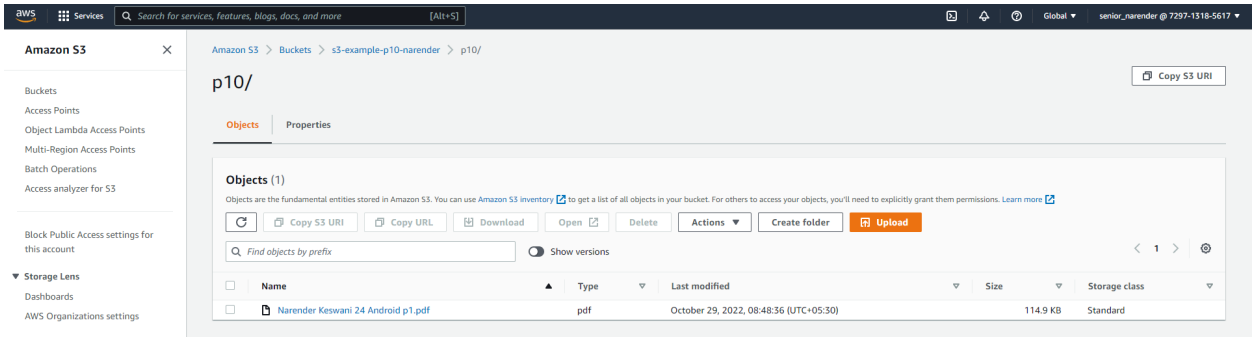
B) Senior_narender login:



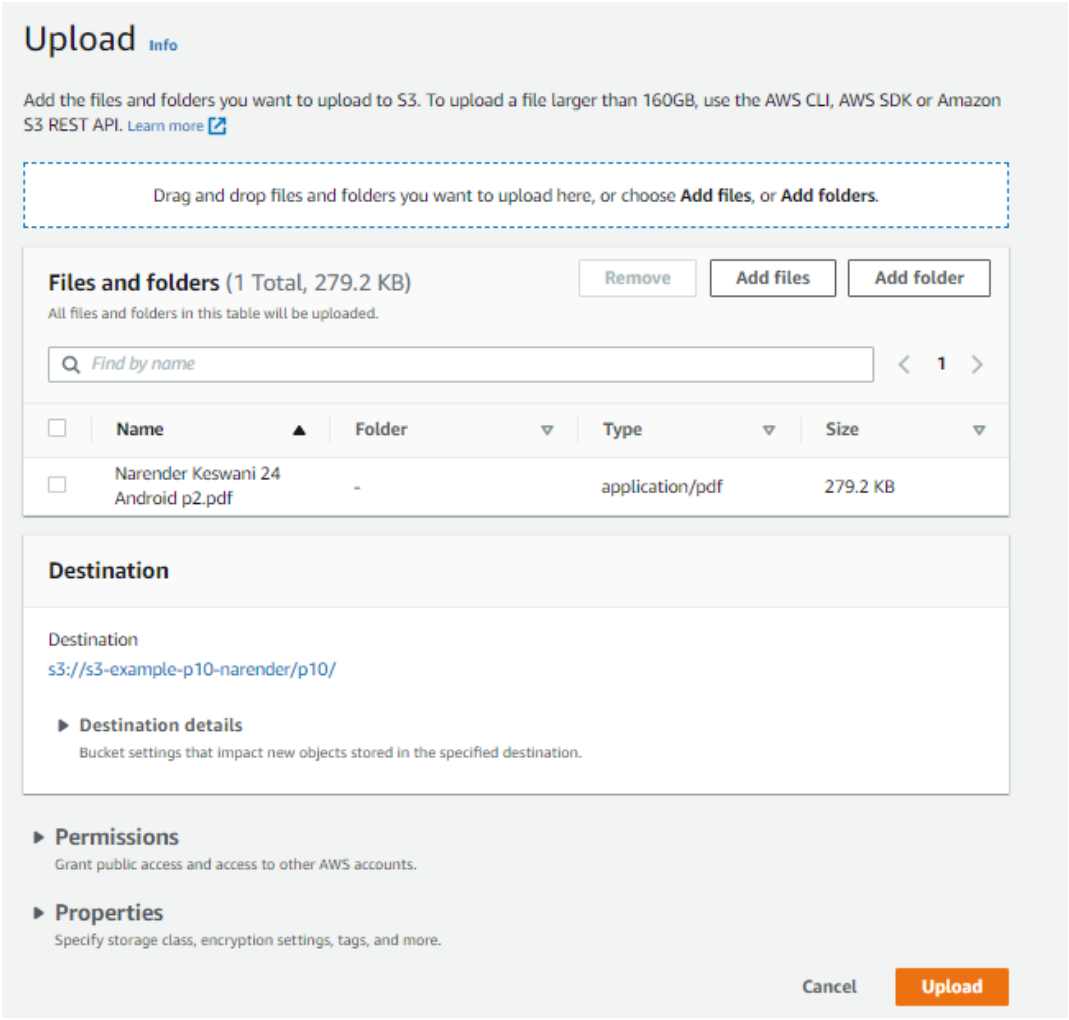
Get MFA Code from google authenticator app



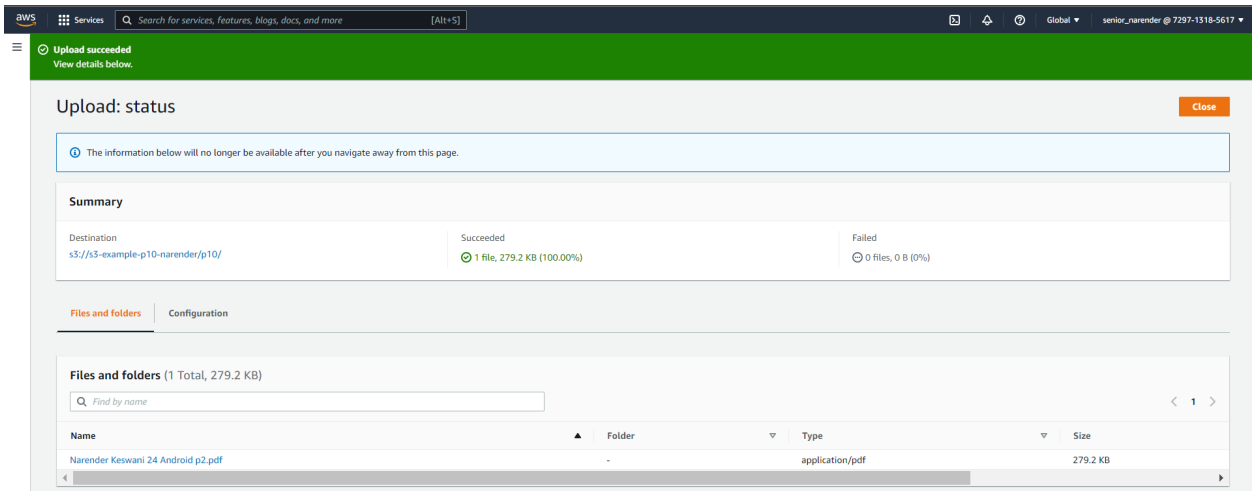
Switch to S3 section [search in service section of search bar]
Navigate to s3-example-p10-narender/p10/



Now, try to upload more files:
Click on the upload button:
Add files:



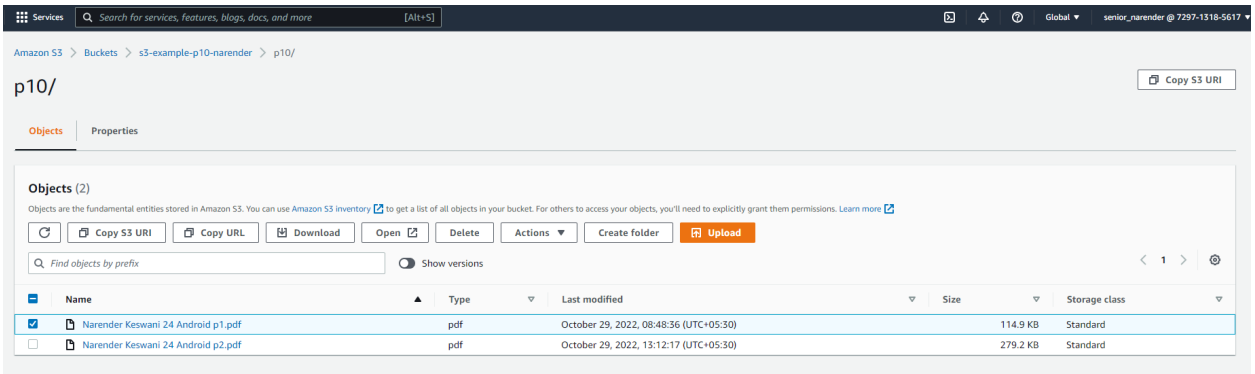
Click on the upload button:



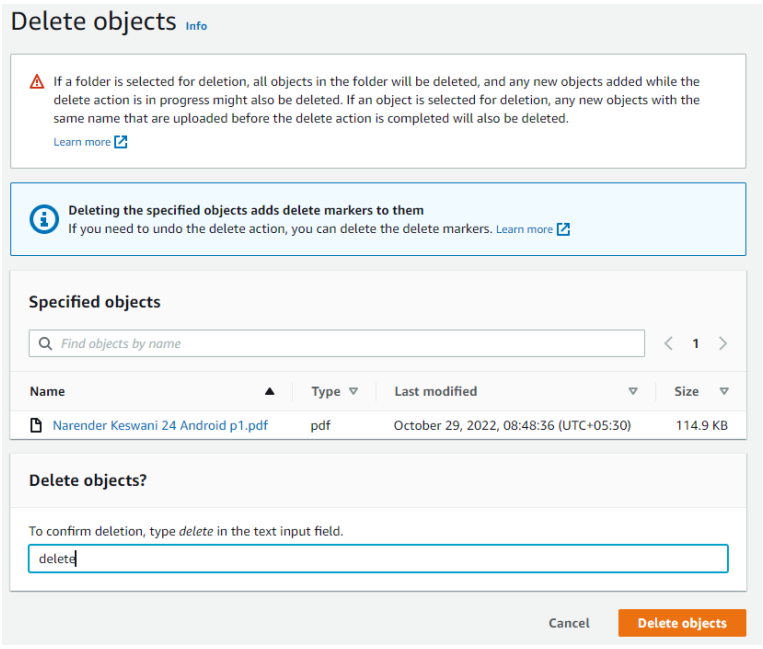
Upload was successful, because the senior_narender has the all the access to s3 bucket services of AWS [defined in user group role]

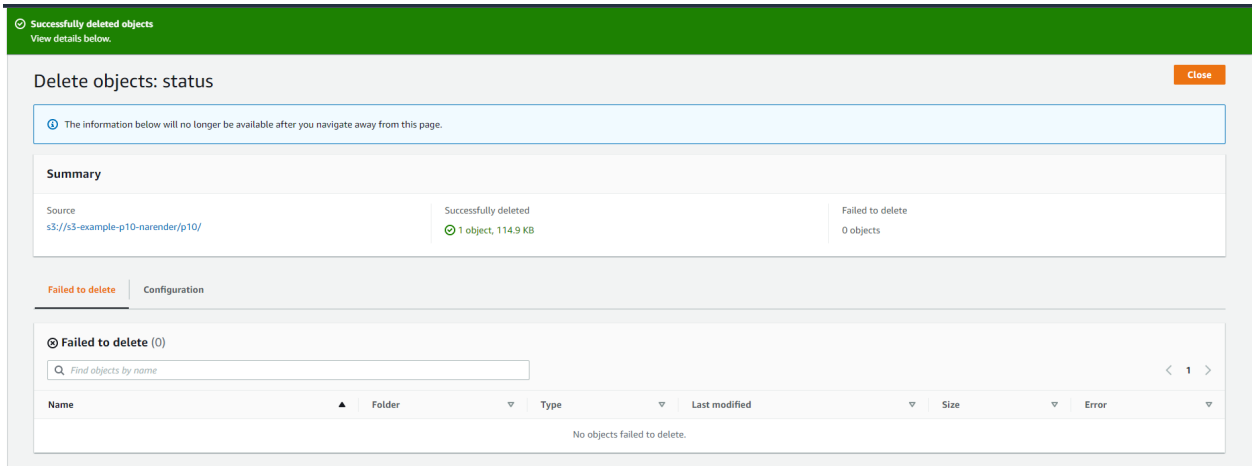
Now , we will try to delete file:

Select the file which you want to delete:



Click on the delete button:
Type delete in textbox:
Click on the delete objects:





File delete was successful, because the senior_narender has the all the access to s3 bucket services of AWS [defined in user group role]

CONCLUSION:

From this practical, I have learned & implemented Identity and Access Management in AWS.