EXPERIMENT NO: 01

TO IMPLEMENT A SERVER CALCULATOR USING RPC CONCEPT. (MAKE USE OF DATAGRAM)

EXPERIMENT NO: 01

Q] To implement a Server calculator using RPC concept. (Make use of datagram)

Code:

RPCClient.java

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

class RPCClient {
   public RPCClient() {
      DatagramSocket sendSocket = null; // Declare outside try-catch
      DatagramSocket receiveSocket = null; // Declare outside try-catch

      try {
         InetAddress ia = InetAddress.getLocalHost();

         // Client sends from an arbitrary ephemeral port, receives on port 1300
         sendSocket = new DatagramSocket(); // Initialize inside try
         receiveSocket = new DatagramSocket(1300); // Initialize inside try

         System.out.println("\nRPC Client - Calculator\n");
         System.out.println("Choose an operation:");
         System.out.println("1. Addition (add)");
         System.out.println("2. Subtraction (sub)");
         System.out.println("3. Multiplication (mul)");
         System.out.println("4. Division (div)");
         System.out.println("Enter 'q' to quit\n");

         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

         while (true) {
```

```
            System.out.print("Enter your choice (1-4 or q): ");
            String choice = br.readLine();

            if (choice.equalsIgnoreCase("q")) {
               // Send "q" to the server to signal shutdown
               byte[] quitBytes = "q".getBytes();
               DatagramPacket quitPacket = new DatagramPacket(quitBytes, quitBytes.length,
ia, 1200);
               sendSocket.send(quitPacket);
               System.out.println("Client shutting down.");
               break;
            }

            String methodName = "";
            switch (choice) {
               case "1":
                  methodName = "add";
                  break;
               case "2":
                  methodName = "sub";
                  break;
               case "3":
                  methodName = "mul";
                  break;
               case "4":
                  methodName = "div";
                  break;
               default:
                  System.out.println("Invalid choice. Please enter 1, 2, 3, 4, or q.");
                  continue; // Go back to the beginning of the loop
            }
```

```
            System.out.print("Enter first number: ");
            String num1Str = br.readLine();
            int val1;
            try {
               val1 = Integer.parseInt(num1Str);
            } catch (NumberFormatException e) {
               System.out.println("Invalid number. Please enter an integer.");
               continue;
            }

            System.out.print("Enter second number: ");
            String num2Str = br.readLine();
            int val2;
            try {
               val2 = Integer.parseInt(num2Str);
            } catch (NumberFormatException e) {
               System.out.println("Invalid number. Please enter an integer.");
               continue;
            }

            // Construct the message string for the server
            String request = methodName + " " + val1 + " " + val2;
            byte[] b = request.getBytes();

            // Send the request to the server
            DatagramPacket dpSend = new DatagramPacket(b, b.length, ia, 1200);
            sendSocket.send(dpSend);

            // Prepare to receive the response
            byte[] receiveBuffer = new byte[4096]; // Buffer for incoming data
            DatagramPacket dpReceive = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
```

```
            receiveSocket.receive(dpReceive);

            String s = new String(dpReceive.getData(), 0, dpReceive.getLength());
            System.out.println("\nResult = " + s + "\n");
         }
      } catch (IOException e) {
         System.err.println("I/O error: " + e.getMessage());
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         // Now these variables are in scope
         if (sendSocket != null) {
            sendSocket.close();
         }
         if (receiveSocket != null) {
            receiveSocket.close();
         }
      }
   }

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

RPCServer.java

```
import java.util.*;
import java.net.*;
import java.io.*; // <-- Add this import

class RPCServer {
   private DatagramSocket ds; // Socket for receiving client requests
```

```
// Define an enum for calculator operations
private enum Operation {
    ADD, SUB, MUL, DIV, UNKNOWN
}

public RPCServer() {
    try {
        ds = new DatagramSocket(1200); // Server listens on port 1200
        byte[] b = new byte[4096]; // Buffer for incoming data

        System.out.println("RPC Server started on port 1200...");
        System.out.println("Waiting for client requests...\n");

        while (true) {
            DatagramPacket dp = new DatagramPacket(b, b.length);
            ds.receive(dp); // Receive client request

            String str = new String(dp.getData(), 0, dp.getLength()); // e.g., "add 10 20"

            if (str.equalsIgnoreCase("q")) {
                System.out.println("Client requested shutdown. Server is terminating.");
                break; // Exit the loop and terminate the server
            }

            String methodNameStr;
            Operation operation;
            int val1, val2;
            String result = "Error: Invalid request format"; // Default error message

            try {
                StringTokenizer st = new StringTokenizer(str, " ");
```

```
            methodNameStr = st.nextToken().toUpperCase(); // Convert to uppercase for
enum matching

            // Convert string to Operation enum
            try {
               operation = Operation.valueOf(methodNameStr);
            } catch (IllegalArgumentException e) {
               operation = Operation.UNKNOWN;
            }

            val1 = Integer.parseInt(st.nextToken());
            val2 = Integer.parseInt(st.nextToken());

            System.out.println("Received request: " + methodNameStr + " " + val1 + " " +
val2);

            switch (operation) {
               case ADD:
                  result = String.valueOf(add(val1, val2));
                  break;
               case SUB:
                  result = String.valueOf(sub(val1, val2));
                  break;
               case MUL:
                  result = String.valueOf(mul(val1, val2));
                  break;
               case DIV:
                  if (val2 == 0) {
                     result = "Error: Division by zero";
                  } else {
                     result = String.valueOf(div(val1, val2));
                  }
                  break;
```

```
                case UNKNOWN:
                default: // Should be caught by UNKNOWN, but good practice
                    result = "Error: Unknown method " + methodNameStr;
                    break;
                }
            } catch (NoSuchElementException e) {
                result = "Error: Malformed request. Expected 'method val1 val2'.";
                System.err.println("Error parsing request: " + e.getMessage());
            } catch (NumberFormatException e) {
                result = "Error: Invalid number format. Parameters must be integers.";
                System.err.println("Error converting numbers: " + e.getMessage());
            }

            // Send the result back to the client
            byte[] b1 = result.getBytes();
            // Use the address and port from the incoming packet to send the reply back
            DatagramPacket dp1 = new DatagramPacket(b1, b1.length, dp.getAddress(),
dp.getPort());
            ds.send(dp1);

            System.out.println("Sent result: " + result + "\n");
        }
    } catch (IOException e) { // This line now compiles
        System.err.println("I/O error in server: " + e.getMessage());
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (ds != null) {
            ds.close(); // Close the server socket
        }
    }
}
```

```
    public int add(int val1, int val2) { return val1 + val2; }
    public int sub(int val3, int val4) { return val3 - val4; }
    public int mul(int val3, int val4) { return val3 * val4; }
    public int div(int val3, int val4) { return val3 / val4; }

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

Output:


C:\Windows\System32\cmd.e
Enter 'q' to quit
Enter your choice (1-4 or q): 1
Enter first number: 12
Enter second number: 5
Result = 17
Enter your choice (1-4 or q): 2
Enter first number: 6
Enter second number: 9
Result = -3
Enter your choice (1-4 or q): 3
Enter first number: 10
Enter second number: 3
Result = 30
Enter your choice (1-4 or q): 4
Enter first number: 13
Enter second number: 6
Result = 2
Enter your choice (1-4 or q): q
Client shutting down.
C:\125\Prac 01>


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
C:\125\Prac 01>java RPCServer.java
RPC Server started on port 1200...
Waiting for client requests...
Received request: ADD 12 5
Sent result: 17
Received request: SUB 6 9
Sent result: -3
Received request: MUL 10 3
Sent result: 30
Received request: DIV 13 6
Sent result: 2
Client requested shutdown. Server is terminating.
C:\125\Prac 01>

EXPERIMENT NO: 02

TO IMPLEMENT A DATE TIME SERVER USING RPC CONCEPT. (MAKE USE OF DATAGRAM)

EXPERIMENT NO: 02

Q] To implement a Date Time Server using RPC concept. (Make use of datagram)

Code:

RPCClient1.java

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

class RPCClient1 {
   public RPCClient1() {
      DatagramSocket clientSocket = null; // Use a single socket for sending and receiving

      try {
         InetAddress ia = InetAddress.getLocalHost();

         // Bind the client's single socket to a known port (1300) for both sending and
receiving
         clientSocket = new DatagramSocket(1300);

         System.out.println("\nRPC Client - Date/Time Service\n");
         System.out.println("Available commands:");
         System.out.println("  'date' - To get the current date from the server");
         System.out.println("  'time' - To get the current time from the server");
         System.out.println("  'q'    - To quit the client and signal server shutdown\n");

         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

         while (true) {
            System.out.print("Enter command (date, time, or q): ");
            String command = br.readLine();

            // Send the command to the server
```

```
            byte[] sendBuffer = command.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
ia, 1200);
            clientSocket.send(sendPacket);

            if (command.equalsIgnoreCase("q")) {
               System.out.println("Client shutting down.");
               break; // Exit loop if 'q' is entered
            }

            // Prepare to receive the response
            byte[] receiveBuffer = new byte[4096]; // Increased buffer size for safety
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);

            // Receive the response on the same socket
            clientSocket.receive(receivePacket);

            String result = new String(receivePacket.getData(), 0, receivePacket.getLength());
            System.out.println("\nServer response: " + result + "\n");
         }
      } catch (SocketException e) {
         System.err.println("Socket error: " + e.getMessage());
      } catch (IOException e) {
         System.err.println("I/O error: " + e.getMessage());
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         if (clientSocket != null) {
            clientSocket.close(); // Ensure the socket is closed
         }
      }
   }
```

```
    public static void main(String[] args) {
        new RPCClient1();
    }
}
```

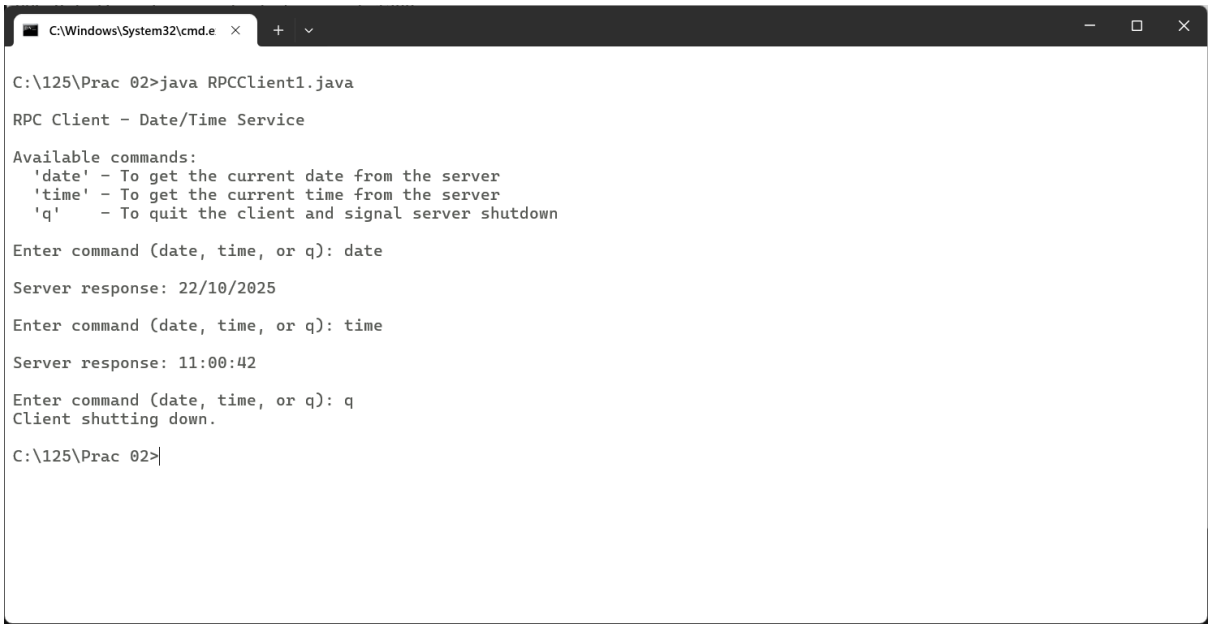
RPCServer1.java

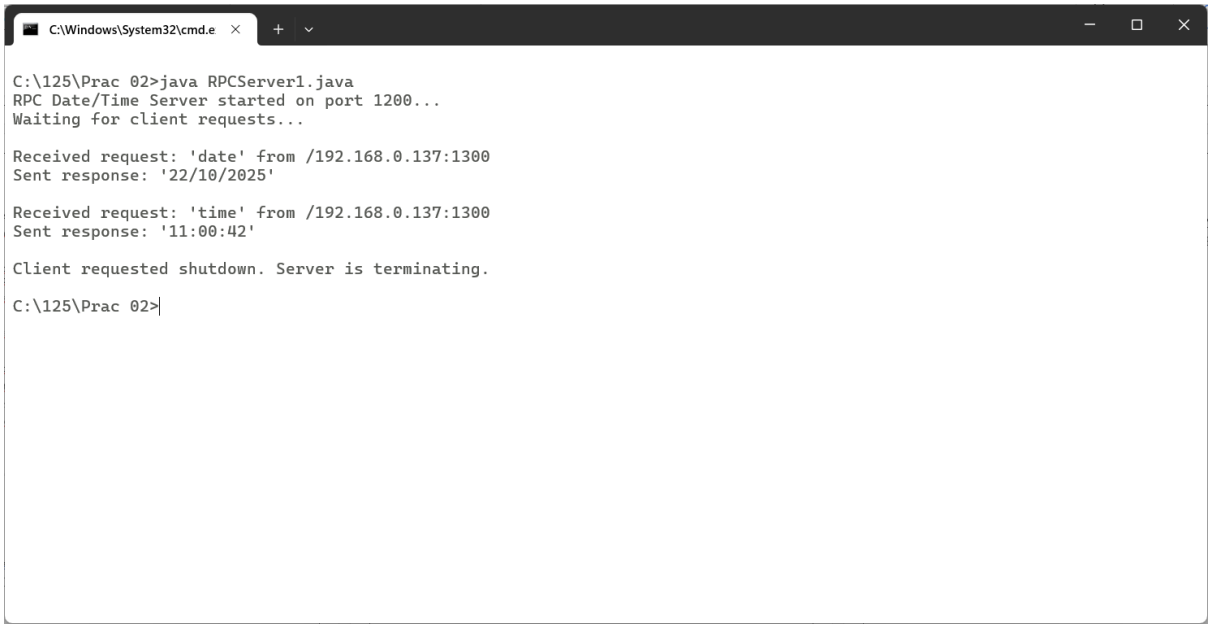
```
import java.io.*; // Added for IOException
import java.util.*;
import java.net.*;
import java.text.SimpleDateFormat;

class RPCServer1 {
    private DatagramSocket ds; // Socket for receiving client requests

    public RPCServer1() {
        try {
            ds = new DatagramSocket(1200); // Server listens on port 1200
            byte[] b = new byte[4096]; // Buffer for incoming data

            System.out.println("RPC Date/Time Server started on port 1200...");
            System.out.println("Waiting for client requests...\n");

            while (true) {
                DatagramPacket dp = new DatagramPacket(b, b.length);
                ds.receive(dp); // Receive client request

                String clientRequest = new String(dp.getData(), 0, dp.getLength()).trim(); // Trim
whitespace

                if (clientRequest.equalsIgnoreCase("q")) {
                    System.out.println("Client requested shutdown. Server is terminating.");
                    // No need to send a reply for 'q' for a clean exit, as client exits locally.
```

```
                // If you want to acknowledge 'q', you could send a "Server shutting down"
message
                // before breaking. For this practical, exiting is fine.
                break;
            }

            String methodName;
            String result = "Error: Unknown command"; // Default error message

            try {
                // In this specific RPC, the request is just the method name ("date" or "time")
                methodName = clientRequest.toLowerCase();

                Calendar c = Calendar.getInstance();
                Date d = c.getTime(); // Get current date and time

                System.out.println("Received request: '" + methodName + "' from " +
dp.getAddress() + ":" + dp.getPort());

                if (methodName.equals("date")) {
                    SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
                    result = dateFormat.format(d);
                } else if (methodName.equals("time")) {
                    // Using SimpleDateFormat for time as well for better formatting options
                    SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm:ss"); // 24-
hour format
                    result = timeFormat.format(d);
                } else {
                    // Handled by the default "Error: Unknown command"
                }
            } catch (Exception e) {
                // Catch any unexpected parsing or date/time formatting errors
                result = "Server error processing request: " + e.getMessage();
```

```
                System.err.println("Server internal error: " + e.getMessage());
            }

            // Send the result back to the client
            byte[] responseBytes = result.getBytes();
            // Send reply to the client's address and the port it sent from (which is now 1300)
            DatagramPacket replyPacket = new DatagramPacket(responseBytes,
responseBytes.length, dp.getAddress(), dp.getPort());
            ds.send(replyPacket);

            System.out.println("Sent response: '" + result + "'\n");
        }
      } catch (IOException e) { // Catch more specific IOException
        System.err.println("I/O error in server: " + e.getMessage());
      } catch (Exception e) { // Catch any other unexpected exceptions
        e.printStackTrace();
      } finally {
        if (ds != null) {
            ds.close(); // Close the server socket
        }
      }
   }

   public static void main(String[] args) {
      new RPCServer1();
   }
}
```

Output:


C:\Windows\System32\cmd.e
C:\125\Prac 02>java RPCClient1.java
RPC Client - Date/Time Service
Available commands:
'date' - To get the current date from the server
'time' - To get the current time from the server
'q' - To quit the client and signal server shutdown
Enter command (date, time, or q): date
Server response: 22/10/2025
Enter command (date, time, or q): time
Server response: 11:00:42
Enter command (date, time, or q): q
Client shutting down.
C:\125\Prac 02>


C:\Windows\System32\cmd.e
C:\125\Prac 02>java RPCServer1.java
RPC Date/Time Server started on port 1200...
Waiting for client requests...
Received request: 'date' from /192.168.0.137:1300
Sent response: '22/10/2025'
Received request: 'time' from /192.168.0.137:1300
Sent response: '11:00:42'
Client requested shutdown. Server is terminating.
C:\125\Prac 02>

EXPERIMENT NO: 03

TO IMPLEMENT A SERVER CALCULATOR USING RPC CONCEPT. (MAKE USE OF SERVER SOCKET)

EXPERIMENT NO: 03

Q] To implement a Server calculator using RPC concept. (Make use of Server Socket)

Code:

Client.java

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

public class Client {
   private static final String HOST = "localhost";
   private static final int PORT = 5000;

   public static void main(String[] args) {
      try (
         Socket socket = new Socket(HOST, PORT); // Establish connection
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true); // Auto-flush
output
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         Scanner scanner = new Scanner(System.in)
      ) {
         System.out.println("Connected to Calculator Server at " + HOST + ":" + PORT);
         System.out.println("\n--- Calculator Operations ---");
         System.out.println("1. Addition (add)");
         System.out.println("2. Subtraction (sub)");
         System.out.println("3. Multiplication (mul)");
         System.out.println("4. Division (div)");
         System.out.println("5. Modulo (mod)"); // Assuming 'mod' was added in server
         System.out.println("-----------------------------");
         System.out.println("Type 'quit' to exit.\n");
```

```
String inputLine;
while (true) {
   System.out.print("Enter your choice (1-5 or quit): ");
   String choice = scanner.nextLine();

   if (choice.equalsIgnoreCase("quit")) {
      out.println("quit"); // Signal server to disconnect
      System.out.println("Exiting calculator client.");
      break; // Exit client loop
   }

   String methodName;
   switch (choice) {
      case "1":
         methodName = "add";
         break;
      case "2":
         methodName = "sub";
         break;
      case "3":
         methodName = "mul";
         break;
      case "4":
         methodName = "div";
         break;
      case "5":
         methodName = "mod";
         break;
      default:
         System.out.println("Invalid choice. Please enter 1-5 or 'quit'.");
         continue; // Ask for choice again
   }
```

```
            double num1, num2;

            System.out.print("Enter first number: ");
            try {
                num1 = Double.parseDouble(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid number.");
                continue; // Ask for choice again
            }

            System.out.print("Enter second number: ");
            try {
                num2 = Double.parseDouble(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid number.");
                continue; // Ask for choice again
            }

            // Construct the request string in the format expected by the server
            String request = methodName + " " + num1 + " " + num2;
            out.println(request); // Send request to server

            String response = in.readLine(); // Read response from server
            System.out.println("Result: " + response);
            System.out.println(); // Add a blank line for readability
        }

    } catch (ConnectException e) {
        System.err.println("Connection refused: Make sure the server is running on " + HOST
+ ":" + PORT);
    } catch (UnknownHostException e) {
```

```
            System.err.println("Unknown host: " + HOST);
        } catch (IOException e) {
            System.err.println("I/O error with server: " + e.getMessage());
            // This can happen if the server unexpectedly closes the connection (e.g., server
shutdown)
            if (e.getMessage() != null && e.getMessage().contains("Connection reset")) {
                System.err.println("Server connection was reset. It might have shut down.");
            }
            e.printStackTrace();
        }
    }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors; // For thread pool

public class Server {
    private static final int PORT = 5000;
    // Use a thread pool to handle multiple client connections concurrently
    private static final int THREAD_POOL_SIZE = 10;
    private static final ExecutorService executor =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Calculator Server is running on port " + PORT + " (TCP/IP
Stream Socket)");
            System.out.println("Waiting for client connections...");

            while (true) {
                // Accept incoming client connections
```

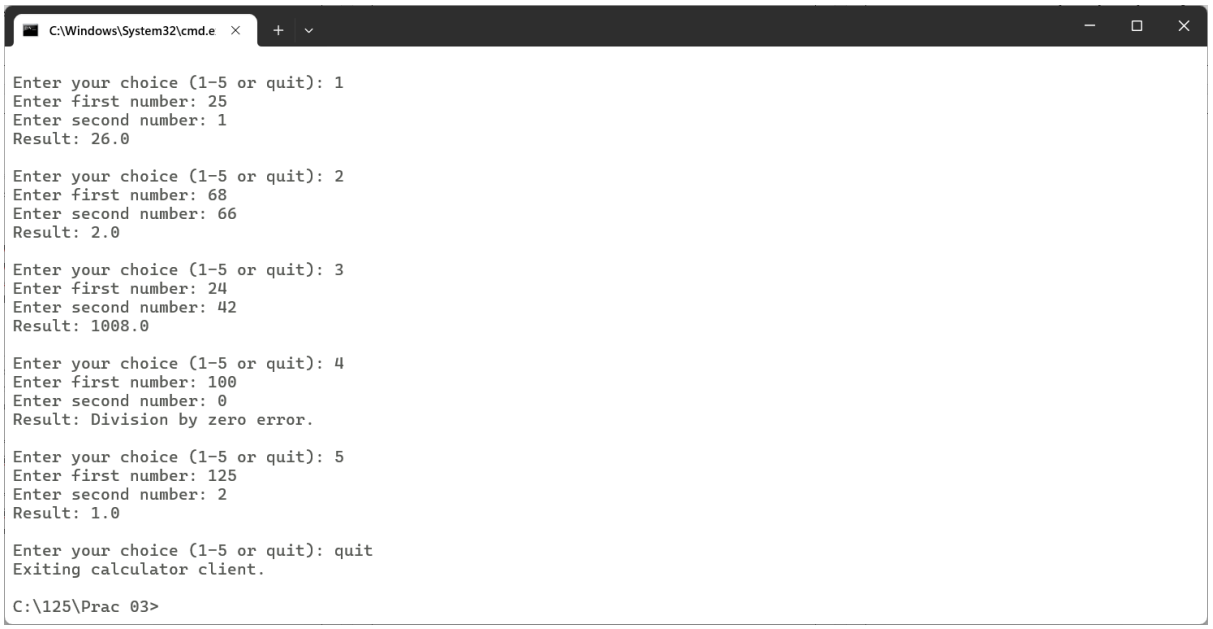
```
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected from: " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());

            // Hand over client handling to a new thread from the pool
            executor.submit(new ClientHandler(clientSocket));
        }
    } catch (IOException e) {
        System.err.println("Server error: " + e.getMessage());
        e.printStackTrace();
    } finally {
        // Shut down the thread pool when the server exits (e.g., on error)
        executor.shutdown();
    }
}

// Inner class to handle each client connection in a separate thread
private static class ClientHandler implements Runnable {
    private Socket clientSocket;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }

    @Override
    public void run() {
        try (
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true); // 'true'
for auto-flush
        ) {
            String request;
```

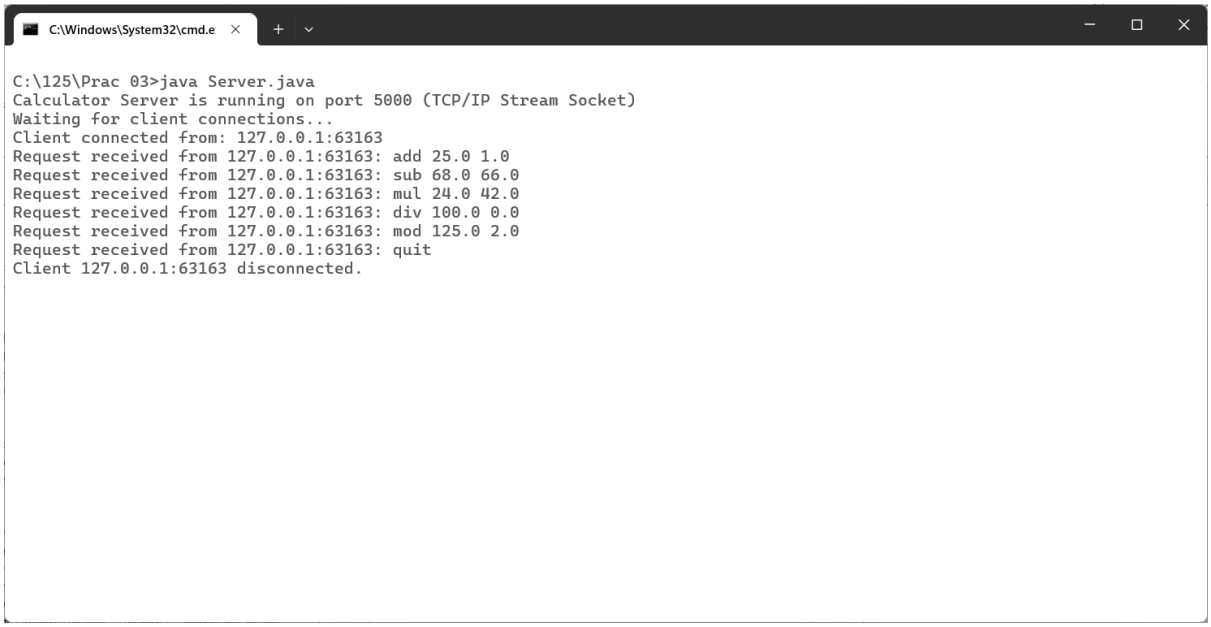
```
            // Keep reading requests from this client until they send "quit"
            while ((request = in.readLine()) != null) {
                System.out.println("Request received from " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort() + ": " +
request);


                if (request.equalsIgnoreCase("quit")) {
                    System.out.println("Client " + clientSocket.getInetAddress().getHostAddress()
+ ":" + clientSocket.getPort() + " disconnected.");
                    break; // Exit loop, closing streams and socket
                }


                String response = handleRequest(request);
                out.println(response);
            }
        } catch (IOException e) {
            // Log client-specific connection issues without crashing the server
            System.err.println("Error handling client " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort() + ": " +
e.getMessage());
        } finally {
            try {
                clientSocket.close(); // Ensure client socket is closed
            } catch (IOException e) {
                System.err.println("Error closing client socket: " + e.getMessage());
            }
        }
    }


    private String handleRequest(String request) {
        String[] tokens = request.split(" ");
        if (tokens.length != 3) {
            return "Invalid format. Use: operation operand1 operand2 (e.g., add 10.5 2.3)";
        }
```

```
        String operation = tokens[0].toLowerCase();
        double num1, num2;

        try {
            num1 = Double.parseDouble(tokens[1]);
            num2 = Double.parseDouble(tokens[2]);
        } catch (NumberFormatException e) {
            return "Invalid numbers. Operands must be numeric.";
        }

        switch (operation) {
            case "add": return String.valueOf(num1 + num2);
            case "sub": return String.valueOf(num1 - num2);
            case "mul": return String.valueOf(num1 * num2);
            case "div":
                if (num2 == 0) return "Division by zero error.";
                return String.valueOf(num1 / num2);
            case "mod": // Added modulo operation
                if (num2 == 0) return "Modulo by zero error.";
                return String.valueOf(num1 % num2);
            default: return "Unsupported operation: " + operation + ". Supported: add, sub, mul,
div, mod.";
        }
    }
  }
}
```

Output:


C:\Windows\System32\cmd.e
C:\125\Prac 03>java Client.java
Connected to Calculator Server at localhost:5000
--- Calculator Operations ---
1. Addition (add)
2. Subtraction (sub)
3. Multiplication (mul)
4. Division (div)
5. Modulo (mod)
Type 'quit' to exit.
Enter your choice (1-5 or quit): 1
Enter first number: 25
Enter second number: 1
Result: 26.0
Enter your choice (1-5 or quit): 2
Enter first number: 68
Enter second number: 66
Result: 2.0
Enter your choice (1-5 or quit): 3
Enter first number: 24
Enter second number: 42
Result: 1008.0
Enter your choice (1-5 or quit): 4
Enter first number: 100


C:\Windows\System32\cmd.e
Enter your choice (1-5 or quit): 1
Enter first number: 25
Enter second number: 1
Result: 26.0
Enter your choice (1-5 or quit): 2
Enter first number: 68
Enter second number: 66
Result: 2.0
Enter your choice (1-5 or quit): 3
Enter first number: 24
Enter second number: 42
Result: 1008.0
Enter your choice (1-5 or quit): 4
Enter first number: 100
Enter second number: 0
Result: Division by zero error.
Enter your choice (1-5 or quit): 5
Enter first number: 125
Enter second number: 2
Result: 1.0
Enter your choice (1-5 or quit): quit
Exiting calculator client.
C:\125\Prac 03>


C:\Windows\System32\cmd.e
C:\125\Prac 03>java Server.java
Calculator Server is running on port 5000 (TCP/IP Stream Socket)
Waiting for client connections...
Client connected from: 127.0.0.1:63163
Request received from 127.0.0.1:63163: add 25.0 1.0
Request received from 127.0.0.1:63163: sub 68.0 66.0
Request received from 127.0.0.1:63163: mul 24.0 42.0
Request received from 127.0.0.1:63163: div 100.0 0.0
Request received from 127.0.0.1:63163: mod 125.0 2.0
Request received from 127.0.0.1:63163: quit
Client 127.0.0.1:63163 disconnected.

EXPERIMENT NO: 04

TO IMPLEMENT A DATE TIME SERVER USING RPC CONCEPT. (MAKE USE OF SERVER SOCKET)

EXPERIMENT NO: 04

Q] To implement a Date Time Server using RPC concept. (Make use of Server Socket)

Code:

DateTimeClient.java

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

public class DateTimeClient {
   private static final String HOST = "localhost";
   private static final int PORT = 5001; // Must match server's port

   public static void main(String[] args) {
      try (
         Socket socket = new Socket(HOST, PORT); // Establish connection
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true); // Auto-flush
output
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         Scanner scanner = new Scanner(System.in)
      ) {
         System.out.println("Connected to Date/Time Server at " + HOST + ":" + PORT);
         System.out.println("\n--- Date/Time Service Operations ---");
         System.out.println("1. Get Current Date (date)");
         System.out.println("2. Get Current Time (time)");
         System.out.println("3. Get Current Date and Time (datetime)");
         System.out.println("-----------------------------------");
         System.out.println("Type 'quit' to exit.\n");

         String inputLine;
         while (true) {
```

```
System.out.print("Enter your choice (1-3 or quit): ");
String choice = scanner.nextLine();

String commandToSend;

switch (choice.toLowerCase()) { // Client still uses switch for user input parsing
   case "1":
      commandToSend = "date";
      break;
   case "2":
      commandToSend = "time";
      break;
   case "3":
      commandToSend = "datetime";
      break;
   case "quit":
      commandToSend = "quit";
      break;
   default:
      System.out.println("Invalid choice. Please enter 1, 2, 3, or 'quit'.");
      System.out.println(); // Add blank line for readability
      continue; // Ask for choice again
}

out.println(commandToSend); // Send request to server

if (commandToSend.equalsIgnoreCase("quit")) {
   System.out.println("Exiting Date/Time client.");
   break; // Exit client loop
}

String response = in.readLine(); // Read response from server
```

```
            System.out.println("Server response: " + response);
            System.out.println(); // Add a blank line for readability
        }

    } catch (ConnectException e) {
        System.err.println("Connection refused: Make sure the server is running on " + HOST
+ ":" + PORT);
    } catch (UnknownHostException e) {
        System.err.println("Unknown host: " + HOST);
    } catch (IOException e) {
        System.err.println("I/O error with server: " + e.getMessage());
        if (e.getMessage() != null && e.getMessage().contains("Connection reset")) {
            System.err.println("Server connection was reset. It might have shut down.");
        }
        e.printStackTrace();
    }
  }
}
```

DateTimeServer.java

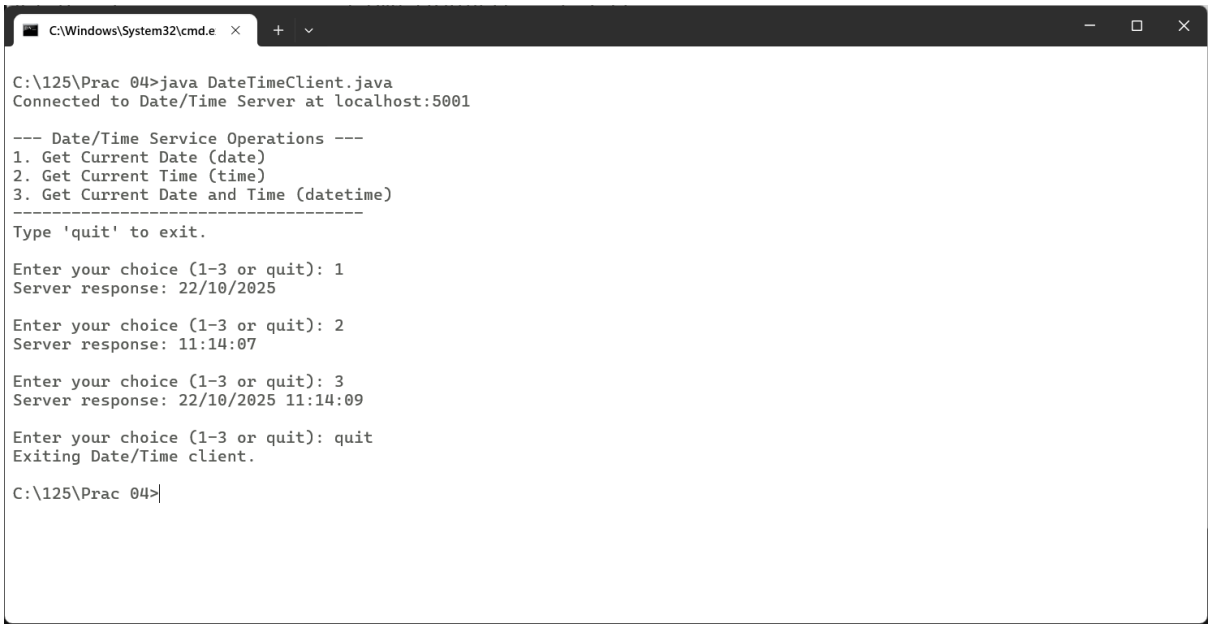
```
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class DateTimeServer {
  private static final int PORT = 5001; // Using a different port than the calculator
  private static final int THREAD_POOL_SIZE = 5;
  private static final ExecutorService executor =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);

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

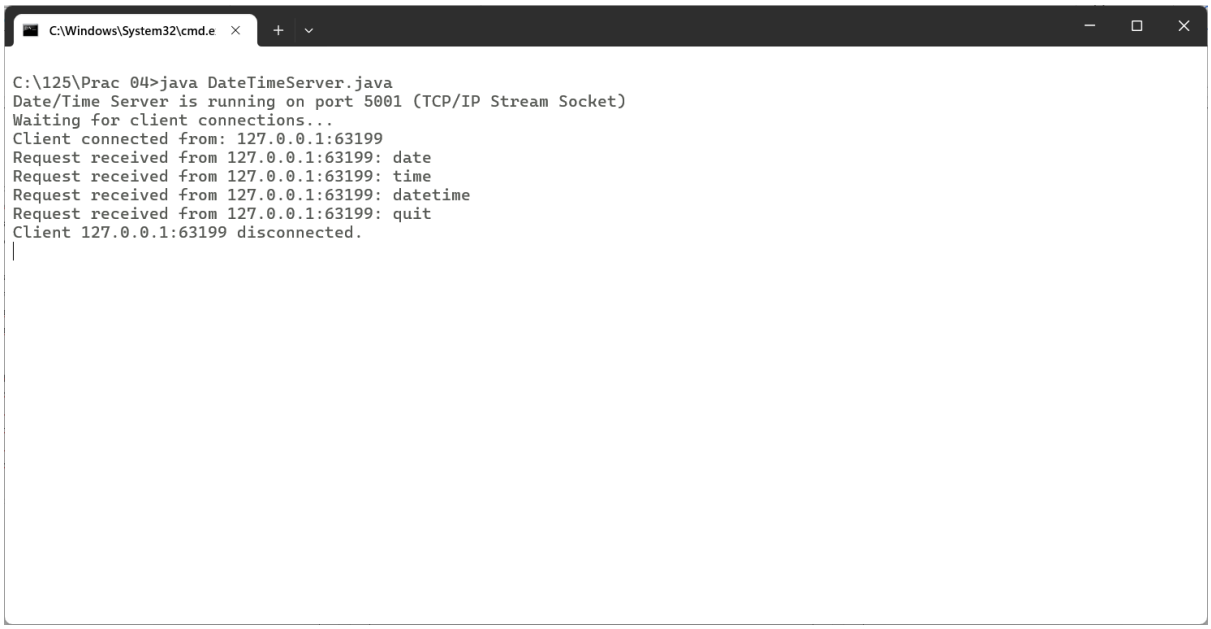
```
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Date/Time Server is running on port " + PORT + " (TCP/IP
Stream Socket)");
            System.out.println("Waiting for client connections...");


            while (true) {
                // Accept incoming client connections
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected from: " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());


                // Hand over client handling to a new thread from the pool
                executor.submit(new ClientHandler(clientSocket));
            }
        } catch (IOException e) {
            System.err.println("Server error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // Shut down the thread pool when the server exits (e.g., on error or manual
termination)
            executor.shutdown();
        }
    }


    // Inner class to handle each client connection in a separate thread
    private static class ClientHandler implements Runnable {
        private Socket clientSocket;


        public ClientHandler(Socket socket) {
            this.clientSocket = socket;
        }


        @Override
```

```
    public void run() {
        try (
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true); // 'true'
for auto-flush
        ) {
            String request;
            // Keep reading requests from this client until they send "quit"
            while ((request = in.readLine()) != null) {
                System.out.println("Request received from " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort() + ": " +
request);


                if (request.equalsIgnoreCase("quit")) {
                    System.out.println("Client " + clientSocket.getInetAddress().getHostAddress()
+ ":" + clientSocket.getPort() + " disconnected.");
                    break; // Exit loop, closing streams and socket
                }


                String response = handleRequest(request);
                out.println(response);
            }
        } catch (IOException e) {
            // Log client-specific connection issues without crashing the server
            System.err.println("Error handling client " +
clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort() + ": " +
e.getMessage());
        } finally {
            try {
                clientSocket.close(); // Ensure client socket is closed
            } catch (IOException e) {
                System.err.println("Error closing client socket: " + e.getMessage());
            }
```

```
            }
        }

        // This method acts as the "remote procedure" dispatcher
        private String handleRequest(String request) {
            // Trim whitespace from the request
            String command = request.trim().toLowerCase();

            // Get current date and time
            Date now = new Date();
            SimpleDateFormat dateFormatter = new SimpleDateFormat("dd/MM/yyyy");
            SimpleDateFormat timeFormatter = new SimpleDateFormat("HH:mm:ss"); // 24-hour
format

            if (command.equals("date")) {
                return dateFormatter.format(now);
            } else if (command.equals("time")) {
                return timeFormatter.format(now);
            } else if (command.equals("datetime")) {
                return dateFormatter.format(now) + " " + timeFormatter.format(now);
            } else {
                return "Error: Unknown command. Supported commands: date, time, datetime.";
            }
        }
    }
}
```

Output:


C:\Windows\System32\cmd.e
C:\125\Prac 04>java DateTimeClient.java
Connected to Date/Time Server at localhost:5001
--- Date/Time Service Operations ---
1. Get Current Date (date)
2. Get Current Time (time)
3. Get Current Date and Time (datetime)
Type 'quit' to exit.
Enter your choice (1-3 or quit): 1
Server response: 22/10/2025
Enter your choice (1-3 or quit): 2
Server response: 11:14:07
Enter your choice (1-3 or quit): 3
Server response: 22/10/2025 11:14:09
Enter your choice (1-3 or quit): quit
Exiting Date/Time client.
C:\125\Prac 04>


C:\Windows\System32\cmd.e
C:\125\Prac 04>java DateTimeServer.java
Date/Time Server is running on port 5001 (TCP/IP Stream Socket)
Waiting for client connections...
Client connected from: 127.0.0.1:63199
Request received from 127.0.0.1:63199: date
Request received from 127.0.0.1:63199: time
Request received from 127.0.0.1:63199: datetime
Request received from 127.0.0.1:63199: quit
Client 127.0.0.1:63199 disconnected.

EXPERIMENT NO: 05

TO RETRIEVE DAY, TIME AND DATE FUNCTION FROM SERVER TO CLIENT. THIS PROGRAM SHOULD DISPLAY SERVER DAY, TIME AND DATE. (USE CONCEPT OF JDBC AND RMI FOR ACCESSING MULTIPLE DATA ACCESS OBJECTS)

EXPERIMENT NO: 05

Q] To retrieve day, time and date function from server to client. This program should display server day, time and date. (Use Concept of JDBC and RMI for accessing multiple data access objects)

Code:

IDate.java

```
import java.rmi.*;
public interface IDate extends Remote
 {
    String getDate() throws RemoteException;
 }
```

DateImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

public class DateImpl extends UnicastRemoteObject implements IDate
{
 public DateImpl() throws RemoteException
 {}
 public String getDate()
 {
  Date d=new Date();
  return(d.toString());
 }
}
```

DateServer.java

```
import java.rmi.*;

public class DateServer
 {
```

```
 public static void main(String[] args)
  {
    try
     {
      DateImpl di=new DateImpl();
      Naming.rebind("DServer",di);

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

DateClient.java

```
import java.rmi.*;
public class DateClient
 {
  public static void main(String[] args)
   {
    try
     {
       String url="rmi://127.0.0.1/DServer";
       IDate intf=(IDate)Naming.lookup(url);
       // IDate intf=(IDate)Naming.lookup("rmi://127.0.0.1/DateServer");

       System.out.println("The Date On Server is: "+intf.getDate());
     }
    catch(Exception e)
     {
       e.printStackTrace();
```

```
    }
  }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
C:\125\Prac 05>rmiregistry


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
C:\125\Prac 05>javac DateServer.java
C:\125\Prac 05>java DateServer
Date Server is Ready


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
C:\125\Prac 05>javac DateClient.java
C:\125\Prac 05>java DateClient
The Date On Server is: Wed Oct 22 11:21:08 IST 2025
C:\125\Prac 05>

EXPERIMENT NO: 06

TO IMPLEMENT EQUATION SOLVER USING DATAGRAM. THE CLIENT SHOULD PROVIDE AN EQUATION TO THE SERVER THROUGH AN INTERFACE. THE SERVER WILL SOLVE THE EXPRESSION GIVEN BY THE CLIENT. (A-B)2 = A2 –2AB + B2; IF A = 5 AND B = 2 THEN RETURN VALUE = 52 – 2.5.2 + 22 = 9.

EXPERIMENT NO: 06

Q] To implement Equation solver using Datagram. The client should provide an equation to the Server through an interface. The server will solve the expression given by the client. (a-b)2 = a2 –2ab + b2; If a = 5 and b = 2 then return value = 52 – 2.5.2 + 22 = 9.

Code:

intfEqSolve.java

```
import java.rmi.*;

public interface intfEqSolve extends Remote {
   public int solveEq1(int a, int b) throws RemoteException;
   public int solveEq2(int a, int b) throws RemoteException;
   public int solveEq3(int a, int b) throws RemoteException;
   public int solveEq4(int a, int b) throws RemoteException;

   // Added a method to get the equation's string representation
   public String getEquation(int choice) throws RemoteException;
}
```

implEqSolve.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.stream.Stream;

public class implEqSolve extends UnicastRemoteObject implements intfEqSolve {
   public implEqSolve() throws RemoteException { }

   @Override
   public int solveEq1(int a, int b) throws RemoteException {
      return (a * a) - (2 * a * b) + (b * b);
   }
```

```
    @Override
    public int solveEq2(int a, int b) throws RemoteException {
        return (a * a) + (2 * a * b) + (b * b);
    }

    @Override
    public int solveEq3(int a, int b) throws RemoteException {
        return (a * a * a) - (3 * a * a * b) + (3 * a * b * b) - (b * b * b);
    }

    @Override
    public int solveEq4(int a, int b) throws RemoteException {
        return (a * a * a) + (3 * a * a * b) + (3 * a * b * b) + (b * b * b);
    }

    @Override
    public String getEquation(int choice) throws RemoteException {
        // Use a switch expression for a clean way to return the equation string
        return switch (choice) {
            case 1 -> "(a-b)2 = a2 – 2ab + b2";
            case 2 -> "(a+b)2 = a2 + 2ab + b2";
            case 3 -> "(a-b)3 = a3 – 3a2b + 3ab2 – b3";
            case 4 -> "(a+b)3 = a3 + 3a2b + 3ab2 + b3";
            default -> "Invalid equation choice.";
        };
    }
}
```

serverEqSolve.java

```
import java.io.*;
import java.rmi.*;

public class serverEqSolve {
```

```
    public static void main(String[] args) {
        try {
            // Set the hostname for the RMI registry lookup.
            // This is crucial if the client and server are on different machines.
            // System.setProperty("java.rmi.server.hostname", "127.0.0.1");

            implEqSolve obj = new implEqSolve();
            Naming.rebind("EquationSolver", obj);

            System.out.println("RMI Server for Equation Solving is ready.");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

clientEqSolve.java

```
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.util.Scanner;

public class clientEqSolve {
    public static void main(String[] args) {
        Scanner scanner = null;
        try {
            // A more descriptive name for the RMI object lookup
            intfEqSolve object = (intfEqSolve)
Naming.lookup("rmi://127.0.0.1/EquationSolver");

            scanner = new Scanner(System.in);
            System.out.println("RMI Equation Solver Client is running.");
```

```
while (true) {
    System.out.println("\nEquations:");
    System.out.println("1. (a-b)2");
    System.out.println("2. (a+b)2");
    System.out.println("3. (a-b)3");
    System.out.println("4. (a+b)3");
    System.out.println("Type 'quit' to exit.");

    System.out.print("Choose the equation (1-4): ");
    String choiceStr = scanner.nextLine();

    if (choiceStr.equalsIgnoreCase("quit")) {
        System.out.println("Exiting client.");
        break;
    }

    int choice;
    try {
        choice = Integer.parseInt(choiceStr);
        if (choice < 1 || choice > 4) {
            System.out.println("Invalid option. Please choose a number from 1 to 4.");
            continue;
        }
    } catch (NumberFormatException e) {
        System.out.println("Invalid input. Please enter a number.");
        continue;
    }

    String equationString = object.getEquation(choice);
    System.out.println("You chose: " + equationString);
```

```
            System.out.print("Enter the value of 'a': ");
            int num1;
            try {
                num1 = Integer.parseInt(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input for 'a'. Please enter an integer.");
                continue;
            }

            System.out.print("Enter the value of 'b': ");
            int num2;
            try {
                num2 = Integer.parseInt(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input for 'b'. Please enter an integer.");
                continue;
            }

            int res = 0;
            // Use a switch expression to make the code more concise
            res = switch (choice) {
                case 1 -> object.solveEq1(num1, num2);
                case 2 -> object.solveEq2(num1, num2);
                case 3 -> object.solveEq3(num1, num2);
                case 4 -> object.solveEq4(num1, num2);
                default -> 0; // This case is unreachable due to the validation above
            };

            System.out.println("The answer is: " + res);
        }
    } catch (Exception e) {
        System.err.println("Client exception: " + e.toString());
```

```
            e.printStackTrace();
        } finally {
            if (scanner != null) {
                scanner.close();
            }
        }
    }
}
```

Output:


C:\Windows\System32\cmd.e
C:\125\Prac 06>javac clientEqSolve.java
C:\125\Prac 06>java clientEqSolve
RMI Equation Solver Client is running.
Equations:
1. (a-b)2
2. (a+b)2
3. (a-b)3
4. (a+b)3
Type 'quit' to exit.
Choose the equation (1-4): 2
You chose: (a+b)2 = a2 + 2ab + b2
Enter the value of 'a': 12
Enter the value of 'b': 21
The answer is: 1089
Equations:
1. (a-b)2
2. (a+b)2
3. (a-b)3
4. (a+b)3
Type 'quit' to exit.
Choose the equation (1-4): 3
You chose: (a-b)3 = a3 ? 3a2b + 3ab2 ? b3
Enter the value of 'a': 12
Enter the value of 'b': 21
The answer is: -729


C:\Windows\System32\cmd.e
Equations:
1. (a-b)2
2. (a+b)2
3. (a-b)3
4. (a+b)3
Type 'quit' to exit.
Choose the equation (1-4): 4
You chose: (a+b)3 = a3 + 3a2b + 3ab2 + b3
Enter the value of 'a': 12
Enter the value of 'b': 21
The answer is: 35937
Equations:
1. (a-b)2
2. (a+b)2
3. (a-b)3
4. (a+b)3
Type 'quit' to exit.
Choose the equation (1-4): quit
Exiting client.
C:\125\Prac 06>

EXPERIMENT NO: 07

USING MYSQL CREATE LIBRARY DATABASE. CREATE TABLE BOOK (BOOK_ID, BOOK_NAME, BOOK_AUTHOR AND RETRIEVE THE BOOK INFORMATION FROM LIBRARY DATABASE USING REMOTE OBJECT COMMUNICATION CONCEPT.

EXPERIMENT NO: 07

Q] Using MySQL create Library database. Create table Book (Book_id, Book_name, Book_author) and retrieve the Book information from Library database using Remote Object Communication concept.

Code:

DBServer.java

```
import java.rmi.*;
public class DBServer
 {
  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();
    }
  }
}
```

DBClient.java

```
import java.rmi.*;
import java.io.*;

public class DBClient_P6 {
  public static void main(String[] args) {
    String sql = "", res = "";
    try
```

```
    {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("\n*** Books Table ***");
      sql = "select * from Book";
      String url = "rmi://127.0.0.1/DBServer";
      IDb id = (IDb) Naming.lookup(url);
      res = id.getData(sql, "LibraryDB");
      System.out.println("\n-----------------------------------------------------------");
      System.out.print(res);
      System.out.println("-----------------------------------------------------------");
    } // end of try
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

DBImpl.java

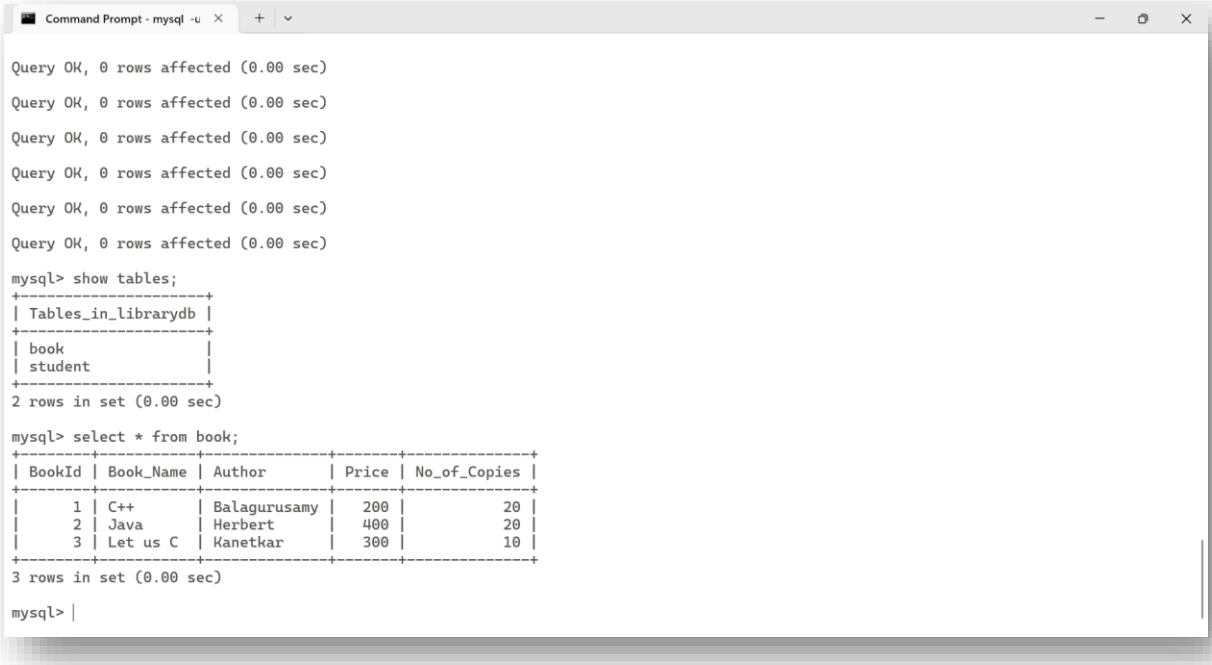
```
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;

public class DBImpl extends UnicastRemoteObject implements IDb
  {
    String str,str1;
    public DBImpl() throws RemoteException
    {}

    public String getData(String sql,String dsn)
    {
      String URL="jdbc:odbc:"+ dsn;
      try
```

```
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection(URL);
      Statement s=con.createStatement();
      ResultSet rs=s.executeQuery(sql);
      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);
 }
}
```

IDb.java

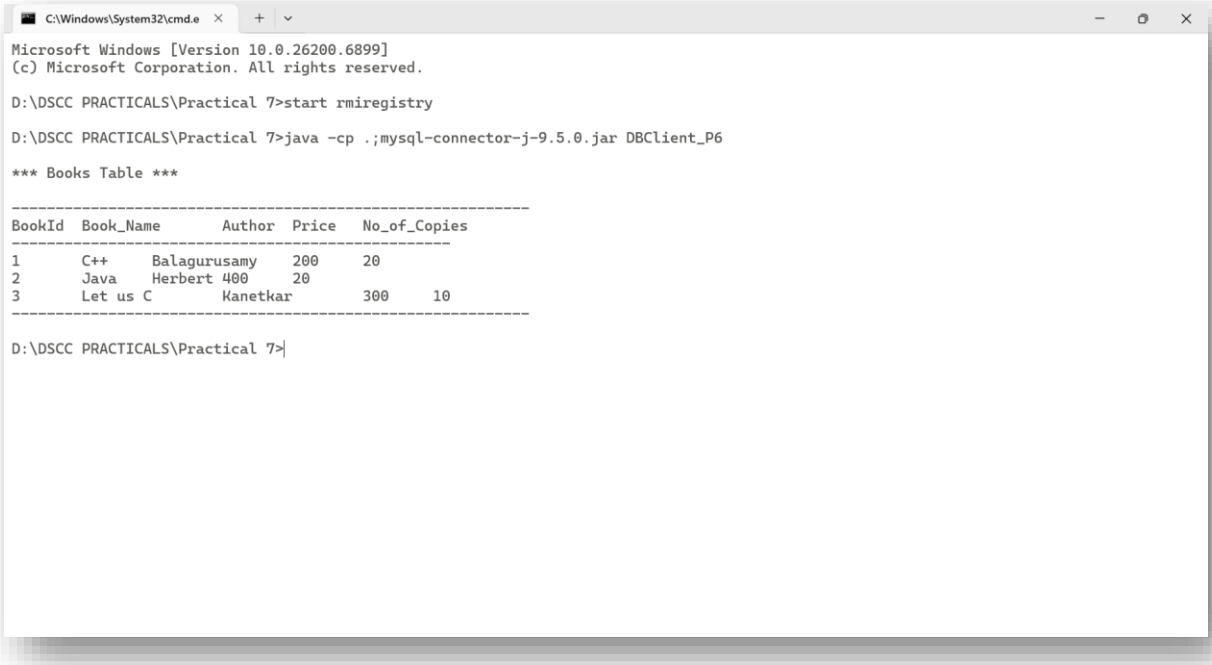
```
import java.rmi.*;
public interface IDb extends Remote
 {
   public String getData(String s,String db) throws RemoteException;
 }
```

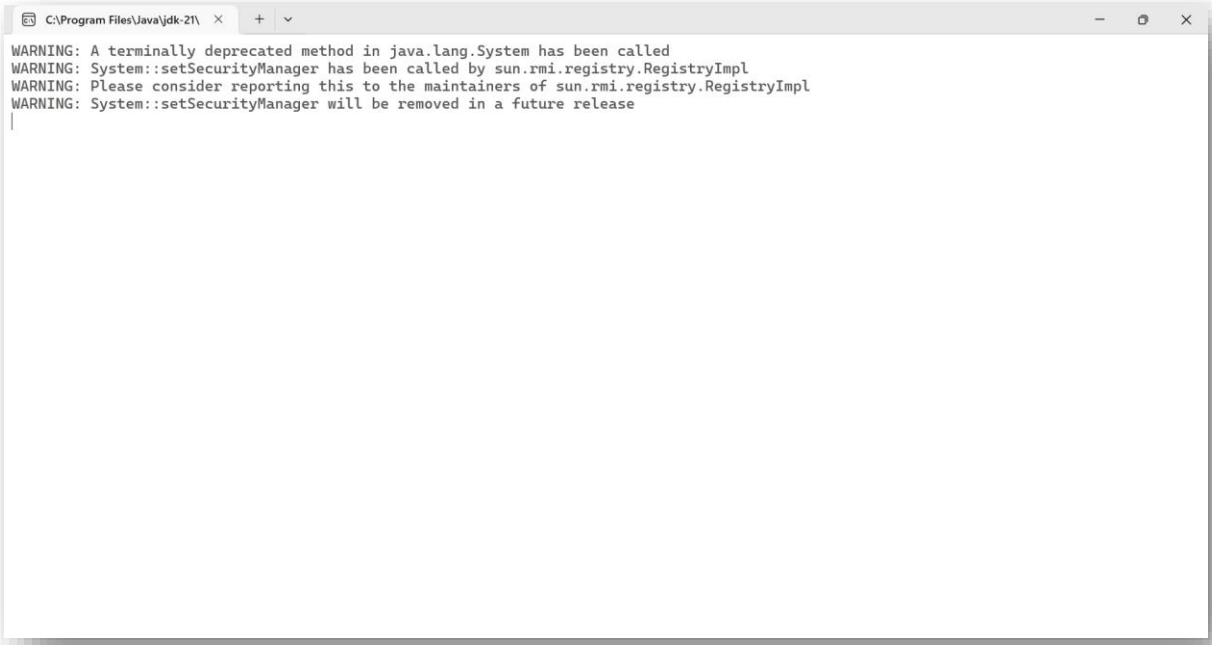
Output:



```
Command Prompt - mysql -u

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+---------------------+
| Tables_in_librarydb |
+---------------------+
| book                |
| student             |
+---------------------+
2 rows in set (0.00 sec)

mysql> select * from book;
+--------+-----------+--------------+-------+--------------+
| BookId | Book_Name | Author       | Price | No_of_Copies |
+--------+-----------+--------------+-------+--------------+
|      1 | C++       | Balagurusamy |   200 |           20 |
|      2 | Java      | Herbert      |   400 |           20 |
|      3 | Let us C  | Kanetkar     |   300 |           10 |
+--------+-----------+--------------+-------+--------------+
3 rows in set (0.00 sec)

mysql>
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.

D:\DSCC PRACTICALS\Practical 7>javac -source 21 -target 21 -cp .;mysql-connector-j-9.5.0.jar *.java
warning: [options] location of system modules is not set in conjunction with -source 21
  not setting the location of system modules may lead to class files that cannot run on JDK 21
    --release 21 is recommended instead of -source 21 -target 21 because it sets the location of system modules automatically
1 warning

D:\DSCC PRACTICALS\Practical 7>java -cp .;mysql-connector-j-9.5.0.jar DBServer
Server Registered.
```


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
D:\DSCC PRACTICALS\Practical 7>start rmiregistry
D:\DSCC PRACTICALS\Practical 7>java -cp .;mysql-connector-j-9.5.0.jar DBClient_P6
*** Books Table ***
BookId Book_Name Author Price No_of_Copies
1 C++ Balagurusamy 200 20
2 Java Herbert 400 20
3 Let us C Kanetkar 300 10
D:\DSCC PRACTICALS\Practical 7>


C:\Program Files\Java\jdk-21\
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release

EXPERIMENT NO: 08

USING MYSQL CREATE THE ELECRTIC_BILL DATABASE. CREATE TABLE BILL (CONSUMER_NAME, BILL_DUE_DATE, BILL_AMOUNT) AND RETRIEVE THE BILL INFORMATION FROM THE ELECRTIC_BILL DATABASE USING REMOTE OBJECT COMMUNICATION CONCEPT.

EXPERIMENT NO: 08

Q] Using MySQL create the Elecrtic_Bill database. Create table Bill (consumer_name, bill_due_date, bill_amount) and retrieve the Bill information from the Elecrtic_Bill database using Remote Object Communication concept.

Code:

DBClient.java

```
import java.rmi.*;
import java.io.*;

public class DBClient_P7 {
  public static void main(String[] args) {
    String sql = "", res = "";
    try
    {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("\n*** Bill Table ***");
      sql = "select * from Bill";
      String url = "rmi://127.0.0.1/DBServer";
      IDb id = (IDb) Naming.lookup(url);
      res = id.getData(sql, "MtnlDB");
      System.out.println("\n-----------------------------------------------------------");
      System.out.print(res);
      System.out.println("-----------------------------------------------------------");
    } // end of try
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

DBServer.java

```
import java.rmi.*;
public class DBServer
 {
  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();
    }
  }
}
```

DBImpl.java

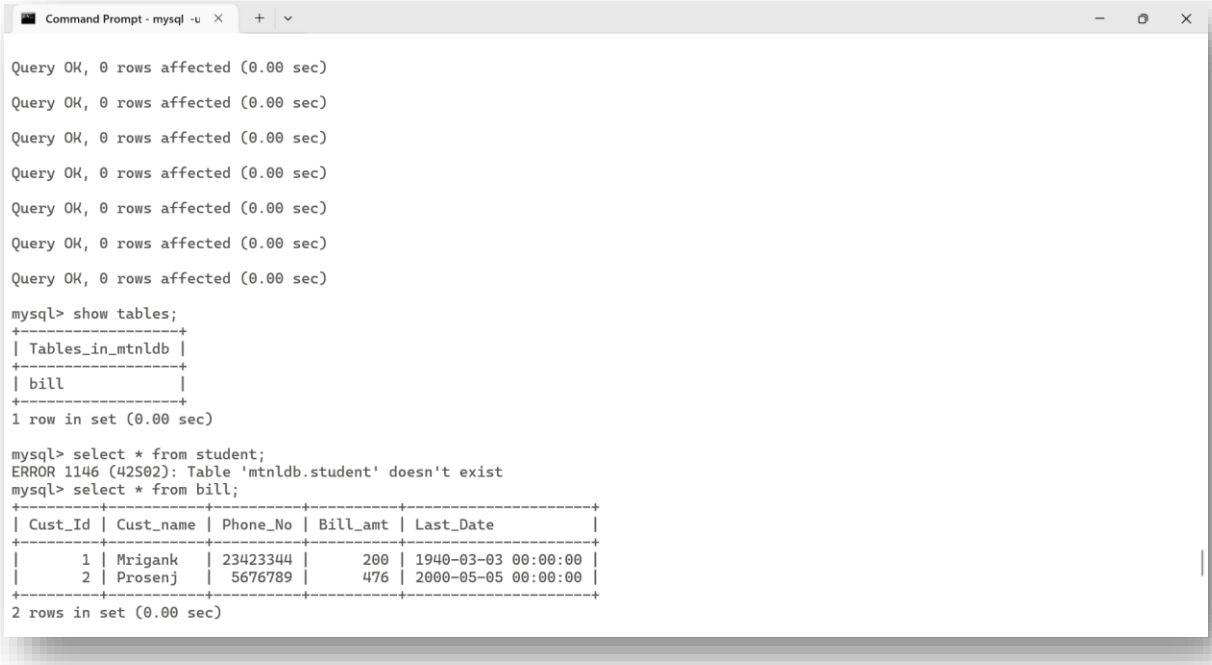
```
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;

public class DBImpl extends UnicastRemoteObject implements IDb
  {
    String str,str1;
    public DBImpl() throws RemoteException
    {}

    public String getData(String sql,String dsn)
    {
      String URL="jdbc:odbc:"+ dsn;
```

```
    try
    {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     Connection con=DriverManager.getConnection(URL);
     Statement s=con.createStatement();
     ResultSet rs=s.executeQuery(sql);
     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);
 }
}
```

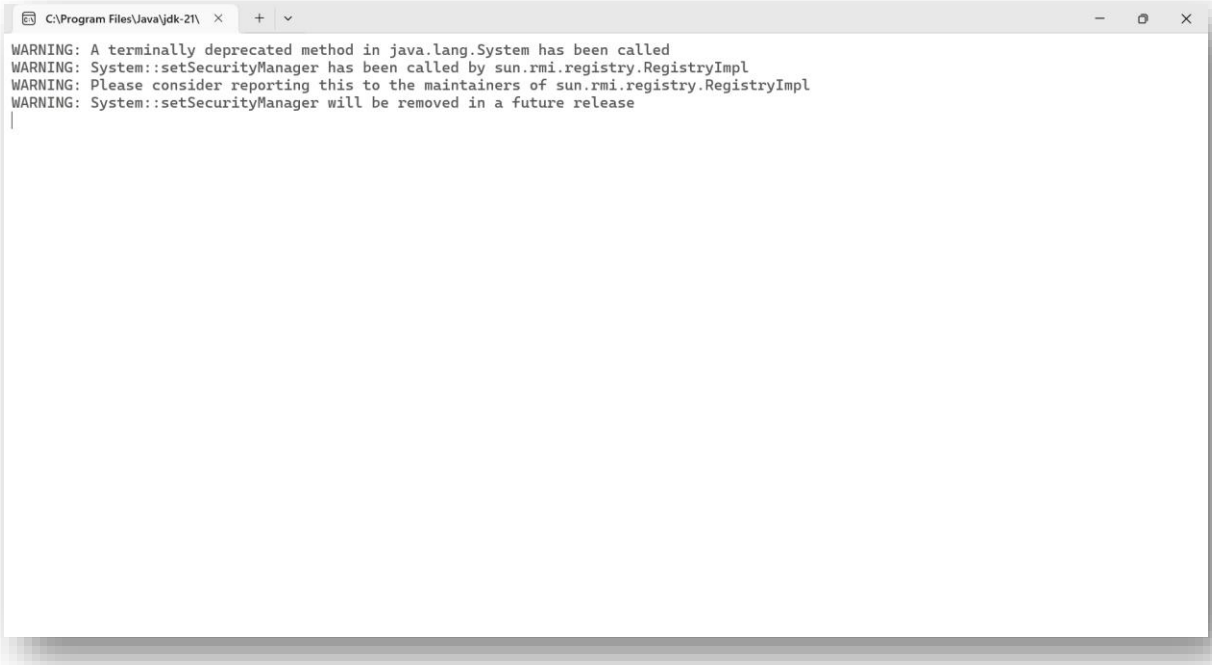
IDb.java

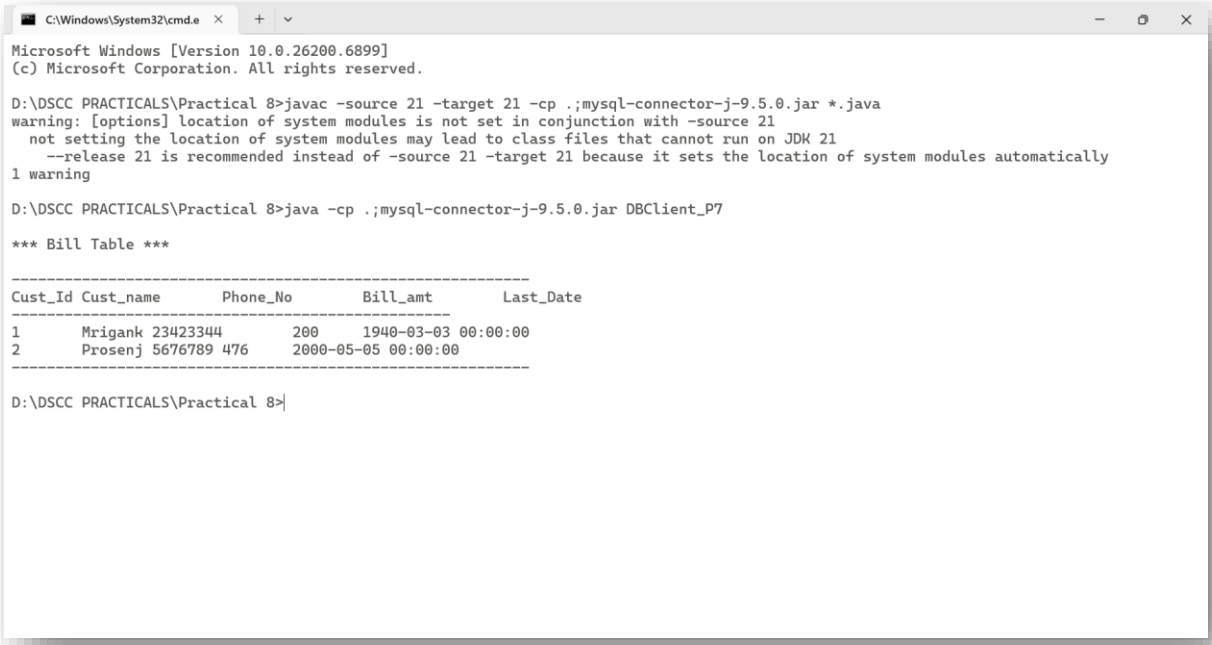
```
import java.rmi.*;
public interface IDb extends Remote
 {
   public String getData(String s,String db) throws RemoteException;
 }
```

Output:


Command Prompt - mysql -u
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
mysql> show tables;
+------------------+
| Tables_in_mtnldb |
+------------------+
| bill |
+------------------+
1 row in set (0.00 sec)
mysql> select * from student;
ERROR 1146 (42S02): Table 'mtnldb.student' doesn't exist
mysql> select * from bill;
+---------+-----------+----------+----------+---------------------+
| Cust_Id | Cust_name | Phone_No | Bill_amt | Last_Date |
+---------+-----------+----------+----------+---------------------+
| 1 | Mrigank | 23423344 | 200 | 1940-03-03 00:00:00 |
| 2 | Prosenj | 5676789 | 476 | 2000-05-05 00:00:00 |
+---------+-----------+----------+----------+---------------------+
2 rows in set (0.00 sec)


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
D:\DSCC PRACTICALS\Practical 8>start rmiregistry
D:\DSCC PRACTICALS\Practical 8>java -cp .;mysql-connector-j-9.5.0.jar DBServer
Server Registered.


C:\Program Files\Java\jdk-21\
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
D:\DSCC PRACTICALS\Practical 8>javac -source 21 -target 21 -cp .;mysql-connector-j-9.5.0.jar *.java
warning: [options] location of system modules is not set in conjunction with -source 21
not setting the location of system modules may lead to class files that cannot run on JDK 21
--release 21 is recommended instead of -source 21 -target 21 because it sets the location of system modules automatically
1 warning
D:\DSCC PRACTICALS\Practical 8>java -cp .;mysql-connector-j-9.5.0.jar DBClient_P7
*** Bill Table ***
Cust_Id Cust_name Phone_No Bill_amt Last_Date
1 Mrigank 23423344 200 1940-03-03 00:00:00
2 Prosenj 5676789 476 2000-05-05 00:00:00
D:\DSCC PRACTICALS\Practical 8>

EXPERIMENT NO: 09

IMPLEMENTATION OF MUTUAL EXCLUSION USING TOKEN RING ALGORITHM.

EXPERIMENT NO: 09

Q] Implementation of mutual exclusion using Token ring algorithm.

Code:

TokenServer.java

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

class TokenServer {
   public static void main(String[] args) {
      DatagramSocket ds = null;
      try {
         ds = new DatagramSocket(1000); // Server listens on port 1000
         System.out.println("TokenServer (Logger) running on port 1000. Waiting for data...");

         while (true) {
            byte buff[] = new byte[1024];
            DatagramPacket dp = new DatagramPacket(buff, buff.length);
            ds.receive(dp);

            String str = new String(dp.getData(), 0, dp.getLength()).trim();

            // Display the received data
            System.out.println("----------------------------------------");
            System.out.println("SHARED RESOURCE ACCESS: " + str);
            System.out.println("----------------------------------------");
         }
      } catch (SocketException e) {
         System.err.println("Error creating socket: " + e.getMessage());
      } catch (IOException e) {
         System.err.println("I/O error: " + e.getMessage());
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (ds != null) {
                ds.close();
            }
        }
    }
}
```

TokenClient1.java

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

class TokenClient1 {
    // Port 100 is for Client 1
    private static final int CLIENT_PORT = 100;
    private static final int NEXT_CLIENT_PORT = 200; // Client 2
    private static final int SERVER_PORT = 1000;
    private static final String TOKEN_MSG = "Token";

    public static void main(String[] args) throws Exception {
        DatagramSocket ds = null;
        BufferedReader br = null;
        boolean hasToken = false; // Start Client1 without the token, let the server or a dedicated starter send it.
                                  // For a simple demo, we'll start Client2 with the token (see below).

        try {
            ds = new DatagramSocket(CLIENT_PORT);
            br = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("TokenClient1 (Port 100) running.");
            System.out.println("Waiting for token...");
```

```
            // --- Initial Token Handoff (Necessary for a ring to start) ---
            // To start the ring, one client must initially have the token.
            // We'll let Client2 start with it for now (see TokenClient2 changes).

        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        while (true) {
            if (hasToken) {
                System.out.println("\n----------------------------------------");
                System.out.println("TOKEN RECEIVED. Do you want to enter data? (yes/no):");
                String ans = br.readLine().trim();
                System.out.println("----------------------------------------");

                if (ans.equalsIgnoreCase("yes")) {
                    System.out.print("Enter data to send to server: ");
                    String data = "Client-1===> " + br.readLine();

                    // 1. Send Data to Server (Shared Resource)
                    byte[] dataBuff = data.getBytes();
                    ds.send(new DatagramPacket(dataBuff, dataBuff.length,
InetAddress.getLocalHost(), SERVER_PORT));
                    System.out.println("Data sent to server (Port " + SERVER_PORT + ").");

                    // 2. Client still has the token, can send again or pass it.
                    // We'll immediately ask if they want to pass the token after sending data.
                    // (This ensures token passing happens without blocking on receive.)
                }
```

```
            // --- Token Passing Logic ---
            System.out.println("Passing token to Client 2 (Port " + NEXT_CLIENT_PORT +
").");
            byte[] tokenBuff = TOKEN_MSG.getBytes();

            // Send Token to Client 2
            ds.send(new DatagramPacket(tokenBuff, tokenBuff.length,
InetAddress.getLocalHost(), NEXT_CLIENT_PORT));
            hasToken = false;

        } else {
            // Client does not have the token, so it must be in receiving mode.
            System.out.println("\nWaiting/Receiving mode... (Port " + CLIENT_PORT + ")");

            byte[] bf = new byte[1024];
            DatagramPacket dp = new DatagramPacket(bf, bf.length);
            ds.receive(dp); // This is a blocking call, waiting for the token or data.

            String incomingMsg = new String(dp.getData(), 0, dp.getLength()).trim();
            System.out.println("Received message: " + incomingMsg);

            if (incomingMsg.equals(TOKEN_MSG)) {
                hasToken = true;
                System.out.println("!!! TOKEN ACQUIRED. !!!");
            } else {
                // This handles unexpected data sent directly between clients (not part of simple
token ring,
                // but necessary if the ring logic allows it).
                System.out.println("Received unexpected data: " + incomingMsg);
            }
        }
    }
}
```

```
}
```

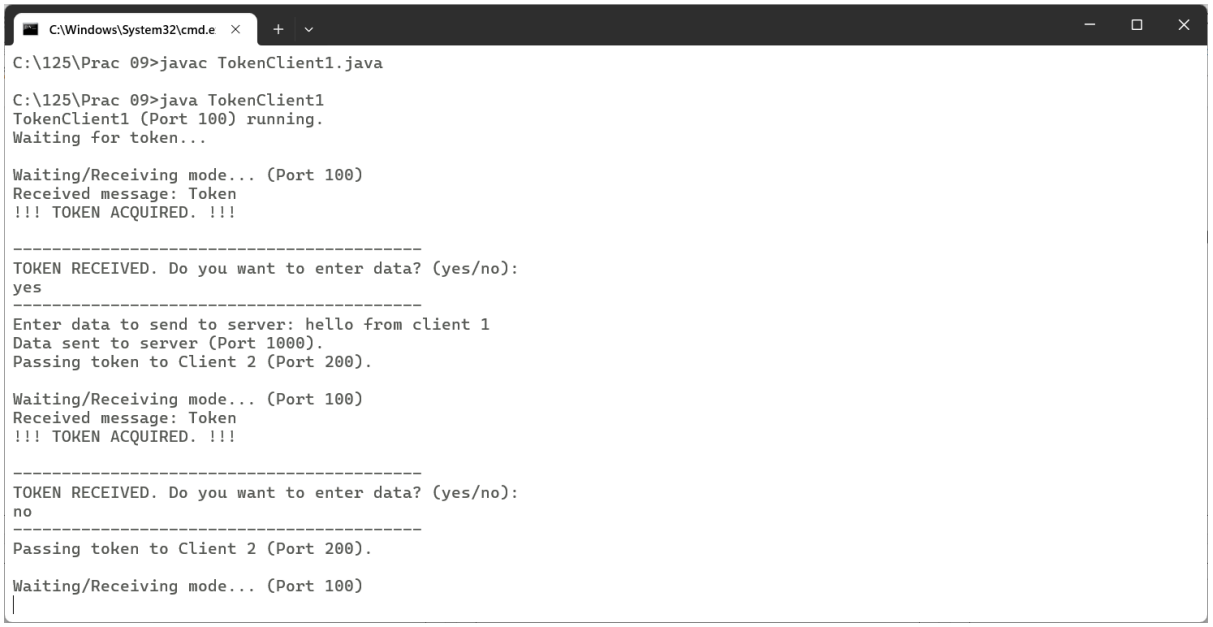
TokenClient2.java

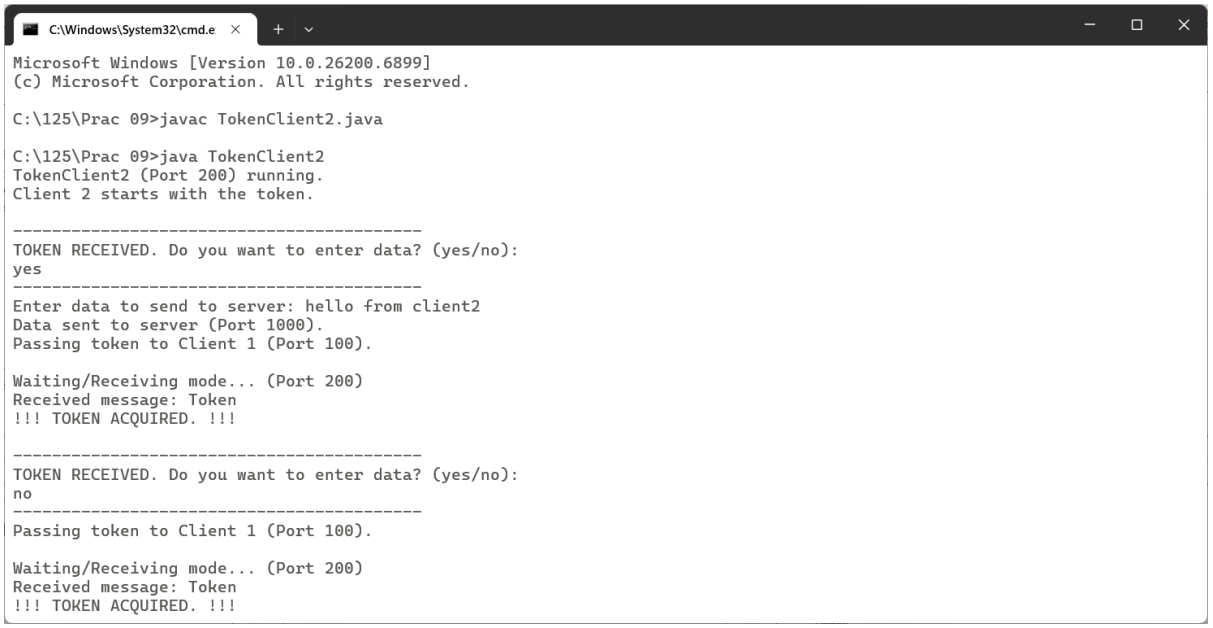
```
import java.net.*;
import java.io.*;

class TokenClient2 {
   // Port 200 is for Client 2
   private static final int CLIENT_PORT = 200;
   private static final int NEXT_CLIENT_PORT = 100; // Client 1
   private static final int SERVER_PORT = 1000;
   private static final String TOKEN_MSG = "Token";

   public static void main(String[] args) throws Exception {
      DatagramSocket ds = null;
      BufferedReader br = null;
      boolean hasToken = true; // Start Client2 with the token to initiate the ring.

      try {
         ds = new DatagramSocket(CLIENT_PORT);
         br = new BufferedReader(new InputStreamReader(System.in));
         System.out.println("TokenClient2 (Port 200) running.");
         System.out.println("Client 2 starts with the token.");

      } catch (Exception e) {
         e.printStackTrace();
         return;
      }

      while (true) {
         if (hasToken) {
            System.out.println("\n----------------------------------------");
            System.out.println("TOKEN RECEIVED. Do you want to enter data? (yes/no):");
```

```
            String ans = br.readLine().trim();
            System.out.println("----------------------------------------");

            if (ans.equalsIgnoreCase("yes")) {
                System.out.print("Enter data to send to server: ");
                String data = "Client-2===> " + br.readLine();

                // 1. Send Data to Server (Shared Resource)
                byte[] dataBuff = data.getBytes();
                ds.send(new DatagramPacket(dataBuff, dataBuff.length,
InetAddress.getLocalHost(), SERVER_PORT));
                System.out.println("Data sent to server (Port " + SERVER_PORT + ").");

                // 2. Client still has the token, can send again or pass it.
                // We'll immediately ask if they want to pass the token after sending data.
            }

            // --- Token Passing Logic ---
            System.out.println("Passing token to Client 1 (Port " + NEXT_CLIENT_PORT +
").");
            byte[] tokenBuff = TOKEN_MSG.getBytes();

            // Send Token to Client 1
            ds.send(new DatagramPacket(tokenBuff, tokenBuff.length,
InetAddress.getLocalHost(), NEXT_CLIENT_PORT));
            hasToken = false;

        } else {
            // Client does not have the token, so it must be in receiving mode.
            System.out.println("\nWaiting/Receiving mode... (Port " + CLIENT_PORT + ")");

            byte[] bf = new byte[1024];
            DatagramPacket dp = new DatagramPacket(bf, bf.length);
```

```
            ds.receive(dp); // This is a blocking call, waiting for the token or data.

            String incomingMsg = new String(dp.getData(), 0, dp.getLength()).trim();
            System.out.println("Received message: " + incomingMsg);

            if (incomingMsg.equals(TOKEN_MSG)) {
               hasToken = true;
               System.out.println("!!! TOKEN ACQUIRED. !!!");
            } else {
               // This handles unexpected data sent directly between clients
               System.out.println("Received unexpected data: " + incomingMsg);
            }
         }
      }
   }
}
```

Output:


C:\Windows\System32\cmd.e
C:\125\Prac 09>javac TokenClient1.java
C:\125\Prac 09>java TokenClient1
TokenClient1 (Port 100) running.
Waiting for token...
Waiting/Receiving mode... (Port 100)
Received message: Token
!!! TOKEN ACQUIRED. !!!
TOKEN RECEIVED. Do you want to enter data? (yes/no):
yes
Enter data to send to server: hello from client 1
Data sent to server (Port 1000).
Passing token to Client 2 (Port 200).
Waiting/Receiving mode... (Port 100)
Received message: Token
!!! TOKEN ACQUIRED. !!!
TOKEN RECEIVED. Do you want to enter data? (yes/no):
no
Passing token to Client 2 (Port 200).
Waiting/Receiving mode... (Port 100)


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.
C:\125\Prac 09>javac TokenClient2.java
C:\125\Prac 09>java TokenClient2
TokenClient2 (Port 200) running.
Client 2 starts with the token.
TOKEN RECEIVED. Do you want to enter data? (yes/no):
yes
Enter data to send to server: hello from client2
Data sent to server (Port 1000).
Passing token to Client 1 (Port 100).
Waiting/Receiving mode... (Port 200)
Received message: Token
!!! TOKEN ACQUIRED. !!!
TOKEN RECEIVED. Do you want to enter data? (yes/no):
no
Passing token to Client 1 (Port 100).
Waiting/Receiving mode... (Port 200)
Received message: Token
!!! TOKEN ACQUIRED. !!!

EXPERIMENT NO: 10

IMPLEMENTATION OF ELECTION ALGORITHM.

EXPERIMENT NO: 10

Q] Implementation of Election Algorithm.

Code:

RingElection.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class RingElection {

   // Represents a single process in the distributed system
   class Process {
      int id;
      boolean isActive;

      Process(int id) {
         this.id = id;
         this.isActive = true; // All processes start active
      }

      @Override
      public String toString() {
         return "P" + id + (isActive ? " (Active)" : " (Failed)");
      }
   }

   private Process[] processes;
   private int totalProcesses = 5;

   public RingElection() {
```

```
        // Initialization is done in the constructor
        System.out.println("Initializing " + totalProcesses + " processes (0 to " + (totalProcesses - 1) + ")");
        processes = new Process[totalProcesses];
        for (int i = 0; i < totalProcesses; i++) {
            processes[i] = new Process(i);
        }
    }

    /**
     * Finds the index of the highest active process ID.
     * @return The array index of the current coordinator.
     */
    public int findCurrentCoordinatorIndex() {
        int index = -1;
        int maxId = -1;
        for (int i = 0; i < processes.length; i++) {
            if (processes[i].isActive && processes[i].id > maxId) {
                maxId = processes[i].id;
                index = i;
            }
        }
        return index;
    }

    /**
     * Simulates the Ring Election Algorithm.
     * @param initiatorIndex The index of the process that detects the failure and starts the election.
     * @param failedIndex The index of the process that is currently failing (the previous coordinator).
     */
    public void runElection(int initiatorIndex, int failedIndex) {
```

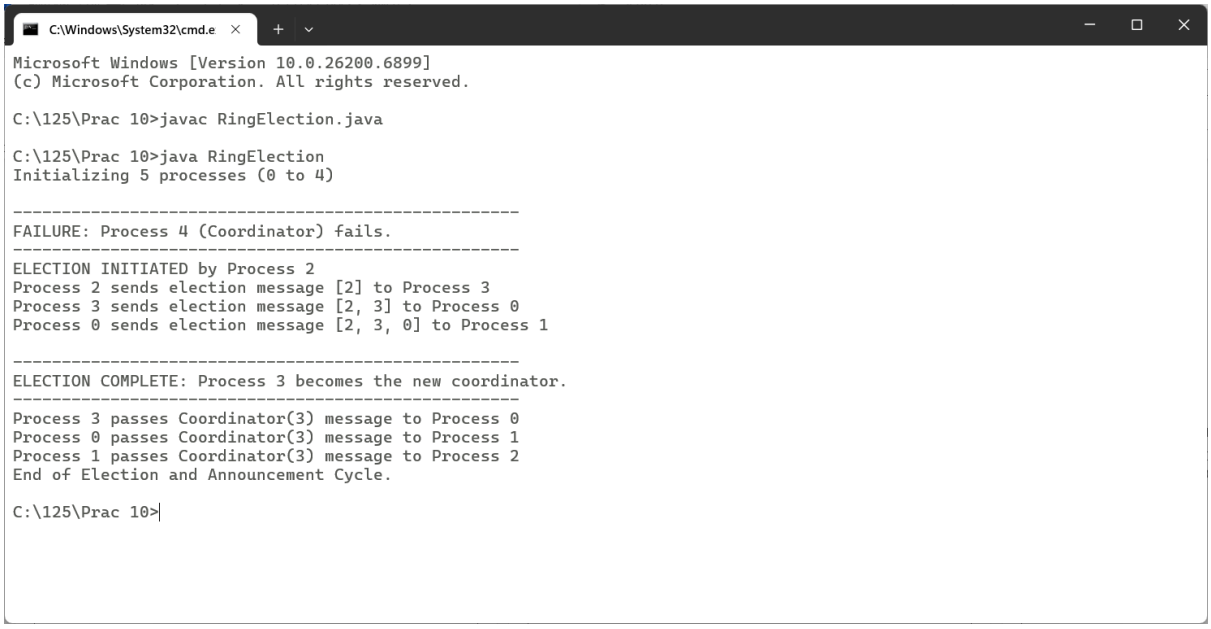
```
        // 1. Failure Simulation
        int oldCoordinatorId = processes[failedIndex].id;
        processes[failedIndex].isActive = false;
        System.out.println("\n-------------------------------------------------");
        System.out.println("FAILURE: Process " + oldCoordinatorId + " (Coordinator) fails.");
        System.out.println("-------------------------------------------------");

        if (!processes[initiatorIndex].isActive) {
            System.err.println("Error: Initiator process " + initiatorIndex + " is not active!");
            return;
        }

        System.out.println("ELECTION INITIATED by Process " +
processes[initiatorIndex].id);

        // The Election Message (list of process IDs involved in the election)
        List<Integer> electionMessage = new ArrayList<>();
        electionMessage.add(processes[initiatorIndex].id);

        int currentProcessIndex = initiatorIndex;

        // 2. Election Phase: Passing the message around the ring
        while (true) {
            int nextProcessIndex = (currentProcessIndex + 1) % totalProcesses;

            // Search for the next active process in the ring
            while (!processes[nextProcessIndex].isActive && nextProcessIndex != initiatorIndex)
{
                nextProcessIndex = (nextProcessIndex + 1) % totalProcesses;
            }

            // Check if the message has completed the loop
```

```
            if (nextProcessIndex == initiatorIndex) {
                // If the message returns, the election phase is complete.
                break;
            }

            Process sender = processes[currentProcessIndex];
            Process receiver = processes[nextProcessIndex];

            System.out.println(
                "Process " + sender.id +
                " sends election message " + electionMessage +
                " to Process " + receiver.id
            );

            // Logic: The receiving process adds its ID to the message if it's not already there
            if (!electionMessage.contains(receiver.id)) {
                electionMessage.add(receiver.id);
            }

            // Move to the receiver for the next hop
            currentProcessIndex = nextProcessIndex;
        }

        // 3. Coordination Phase: Determine and Announce the new coordinator

        // Find the maximum ID in the message list (which represents all participants)
        int newCoordinatorId = Collections.max(electionMessage);

        System.out.println("\n--------------------------------------------------");
        System.out.println("ELECTION COMPLETE: Process " + newCoordinatorId + "
becomes the new coordinator.");
        System.out.println("--------------------------------------------------");
```

```
        // 4. Announcement Phase: Coordinator sends the message to all active processes
        int coordinatorIndex = findCurrentCoordinatorIndex();
        int announcementSenderIndex = coordinatorIndex;

        do {
            int nextProcessIndex = (announcementSenderIndex + 1) % totalProcesses;

            // Search for the next active process in the ring
            while (!processes[nextProcessIndex].isActive && nextProcessIndex !=
coordinatorIndex) {
                nextProcessIndex = (nextProcessIndex + 1) % totalProcesses;
            }

            if (nextProcessIndex == coordinatorIndex) {
                // Check if the announcement completed a full cycle back to the coordinator
                break;
            }

            Process sender = processes[announcementSenderIndex];
            Process receiver = processes[nextProcessIndex];

            System.out.println(
                "Process " + sender.id +
                " passes Coordinator(" + newCoordinatorId + ") message to Process " +
                receiver.id
            );

            announcementSenderIndex = nextProcessIndex;

        } while (announcementSenderIndex != coordinatorIndex);
```

```
        System.out.println("End of Election and Announcement Cycle.");
    }

    public static void main(String[] args) {
        RingElection simulation = new RingElection();

        // Initial Coordinator is P4 (Index 4)
        int initialCoordinatorIndex = simulation.findCurrentCoordinatorIndex();

        // We will simulate P4 failing (Index 4)
        int failedIndex = 4;

        // The election will be initiated by P2 (Index 2)
        int initiatorIndex = 2;

        simulation.runElection(initiatorIndex, failedIndex);
    }
}
```

Output:



EXPERIMENT NO: 11

IMPLEMENTATION OF STORAGE AS A SERVICE USING GOOGLE DOCS

EXPERIMENT NO: 11

Q] Implementation of Storage as a Service using Google Docs

Requirements

- **A computer or laptop with internet connection**
- **Web browser (Google Chrome or Edge)**
- **A valid Google Account**
- **Access to Google Drive (https://drive.google.com/)**

Theory

1. Software as a Service (SaaS)

SaaS is a cloud computing model where users access software applications via the internet using a web browser. The service provider manages the infrastructure, security, and maintenance.
Example: Google Docs, Google Sheets, Microsoft 365.

2. Storage as a Service (STaaS)

STaaS provides users with on-demand cloud storage resources managed by a third-party provider. Users can upload, access, and share files without needing physical storage devices.
Example: Google Drive, Dropbox.

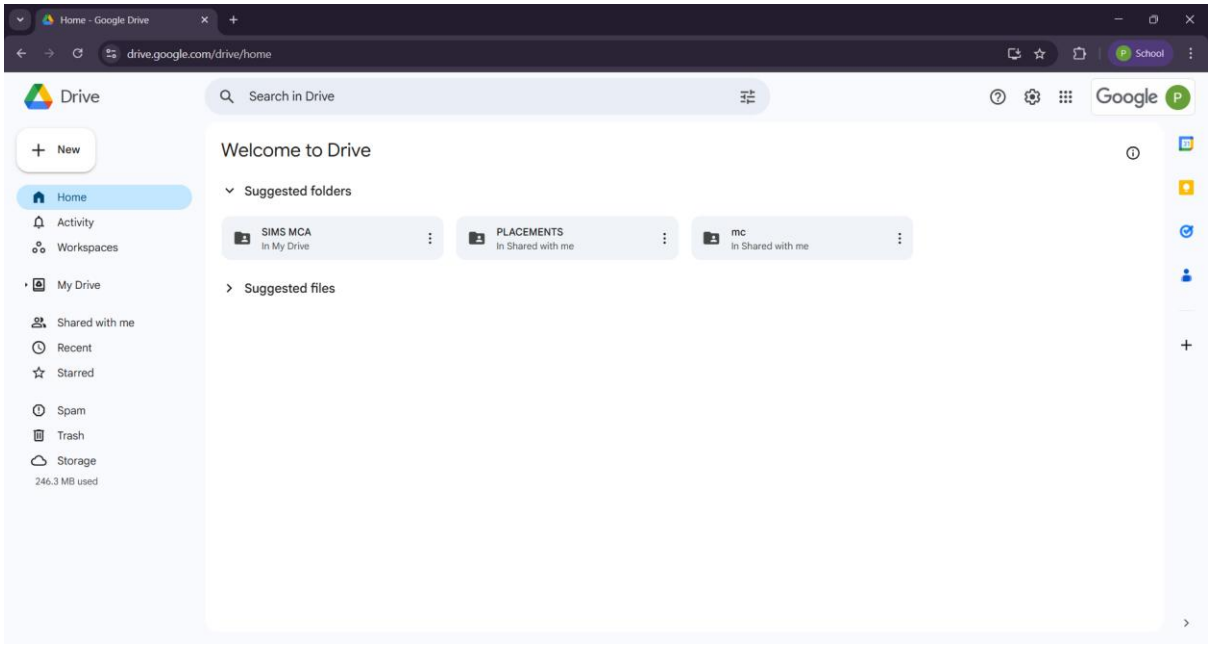
3. Key Features

- **Universal Access:** Files are accessible from any device with internet access.
- **Real-Time Collaboration:** Multiple users can work on the same document simultaneously.
- **Multi-Tenancy:** Multiple users share the same infrastructure securely.
- **Role-Based Access Control (RBAC):** Permissions such as Editor, Viewer, and Commenter manage access.
- **Persistence and Availability:** Files remain available online and offline with automatic synchronization.

Procedure

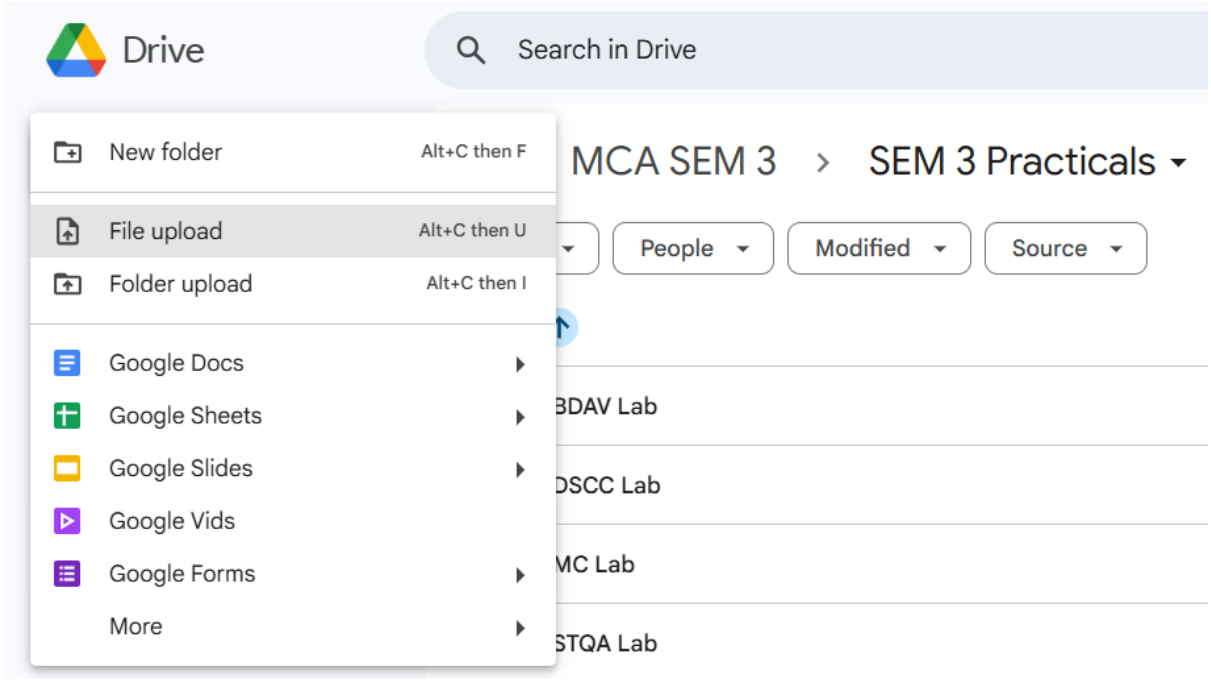
Step 1: Access the Service

1. Open a web browser and navigate to https://drive.google.com/.
2. Log in using a Google account.



Step 2: Demonstrate STaaS (File Upload and Management)

1. Click + New → File upload.
2. Select a file from your computer to upload.
3. Verify that the file appears in your Drive storage.

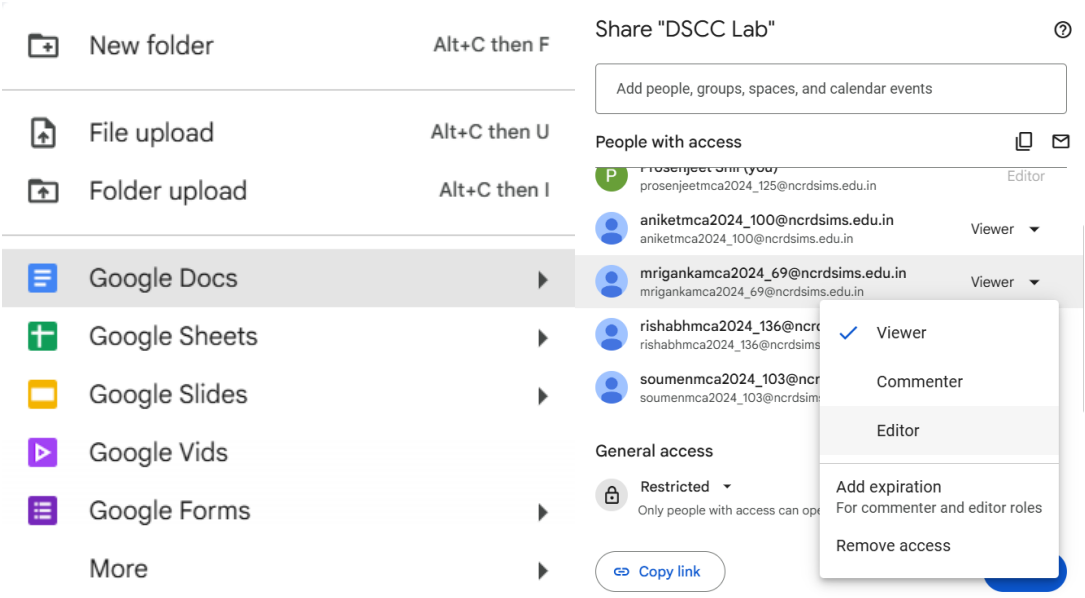


Step 3: Demonstrate SaaS and Collaboration

1. Click + New → Google Docs to create a new cloud document.
2. Click Share → Enter a collaborator's email → Grant Editor access.
3. Ask the collaborator to edit the document.

Step 4: Demonstrate Ownership and Multi-Tenancy

1. Note that the creator is shown as the owner in "My Drive."
2. If using a **Shared Drive**, mention that ownership is assigned to the team.

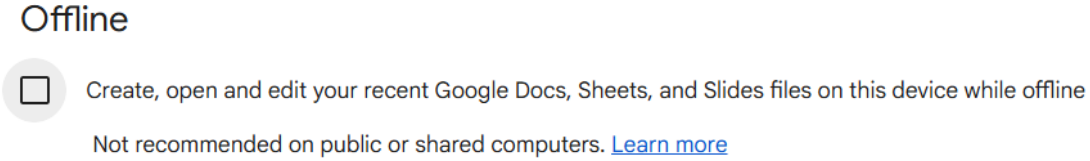


Step 5: Demonstrate Role-Based Access Control (RBAC)

1. Open the sharing dialog again.
2. Change permissions to Viewer or Commenter.
3. Ask the collaborator to try editing.

Step 6: Demonstrate Persistence and Availability

1. Enable Offline Mode in Drive (Settings → Offline).
2. Access the document offline.
3. Once connected again, observe automatic synchronization.




Open with
Download
Rename Ctrl+Alt+E
Make a copy Ctrl+C Ctrl+V
Share
Organize
File information
Make available offline
Remove Delete
Report or block
Name
DSCC Lab
Available Offline

EXPERIMENT NO: 12

IMPLEMENTATION OF IDENTITY MANAGEMENT.

EXPERIMENT NO: 12

Q] Implementation of Identity Management.

Theory

Identity Management is the process of managing individual users' identities and their access to systems, applications, and data. It ensures that only authorized users can access specific resources while maintaining system security.

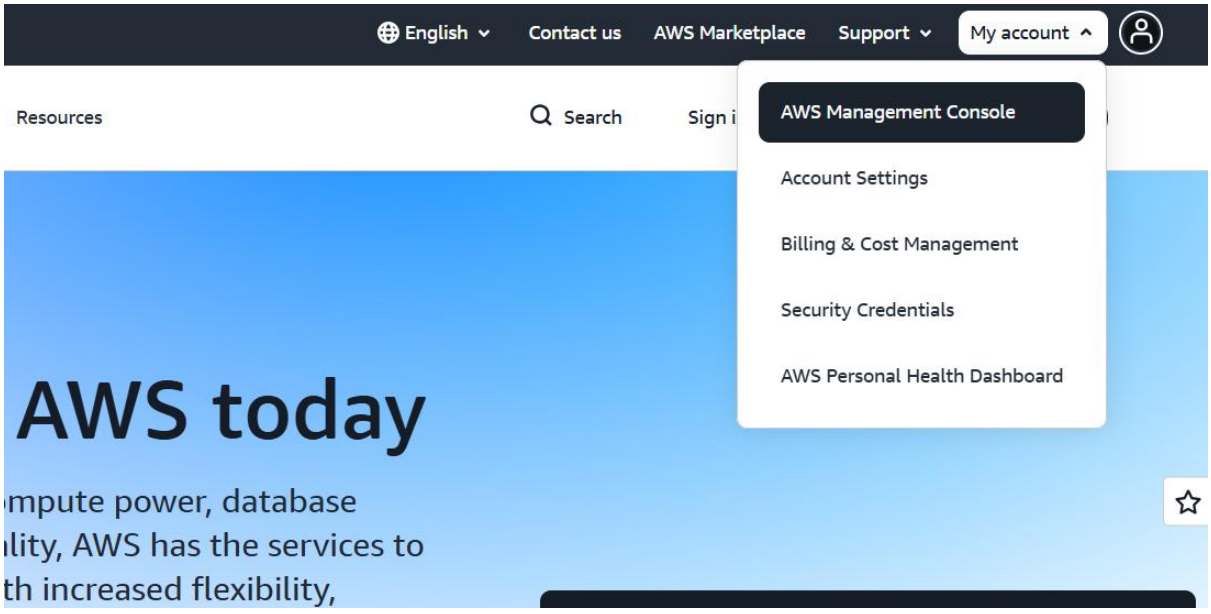
In **cloud computing**, Identity and Access Management (IAM) allows administrators to control who can perform what actions on which resources.
AWS provides **AWS Identity and Access Management (IAM)**, a web service that enables secure control of access to AWS resources.

Key Features of AWS IAM:

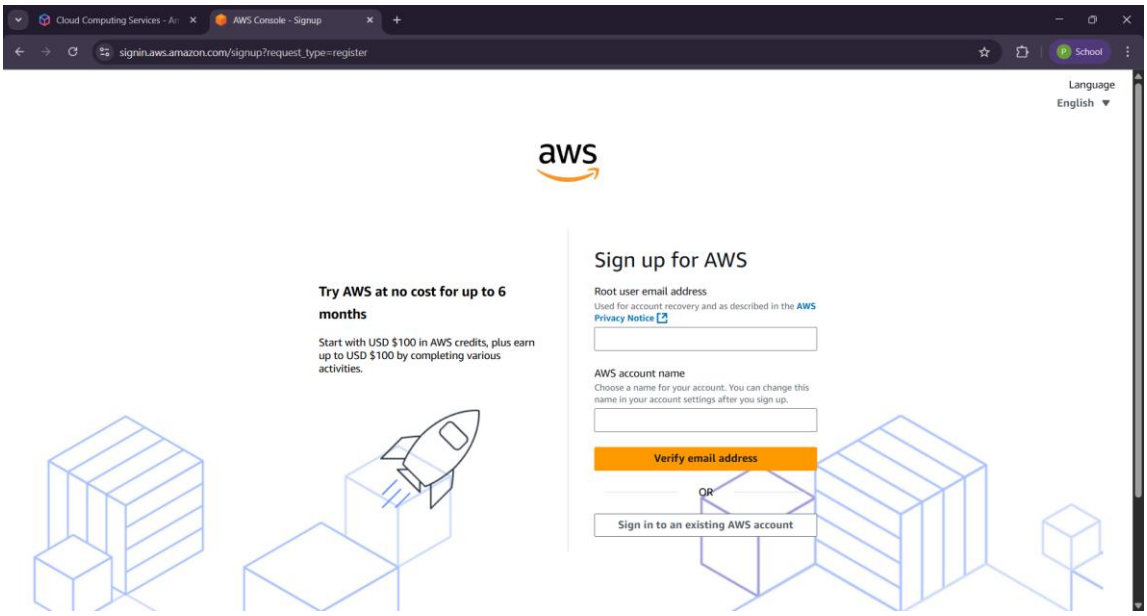
1. **User Management:** Create and manage users within an AWS account.
2. **Group Management:** Group users and assign collective permissions.
3. **Access Control:** Assign permissions using IAM policies.
4. **Multi-Factor Authentication (MFA):** Adds an extra layer of security.
5. **Granular Permissions:** Restrict access based on role or need.

Procedure

Step 1: Open the link https://aws.amazon.com/.
Click on **My Account → AWS Management Console**.



Step 2: Click on **Create a New AWS Account.**



Step 3: Fill in your details (email, password, AWS account name), then click **Continue.**

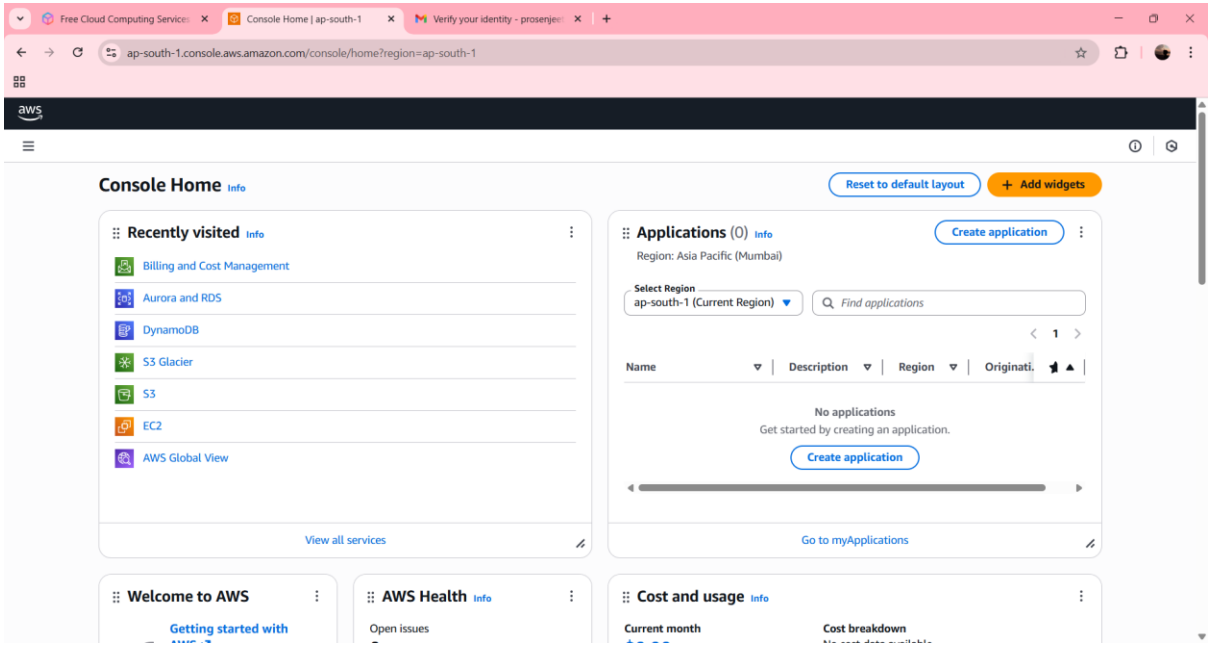
Step 4: Enter your **contact number** and **home address**. Click **Create Account and Continue.**

Step 5: When AWS asks for payment (credit/debit card details), **close the browser**. (This step is skipped for safety in demonstration.)

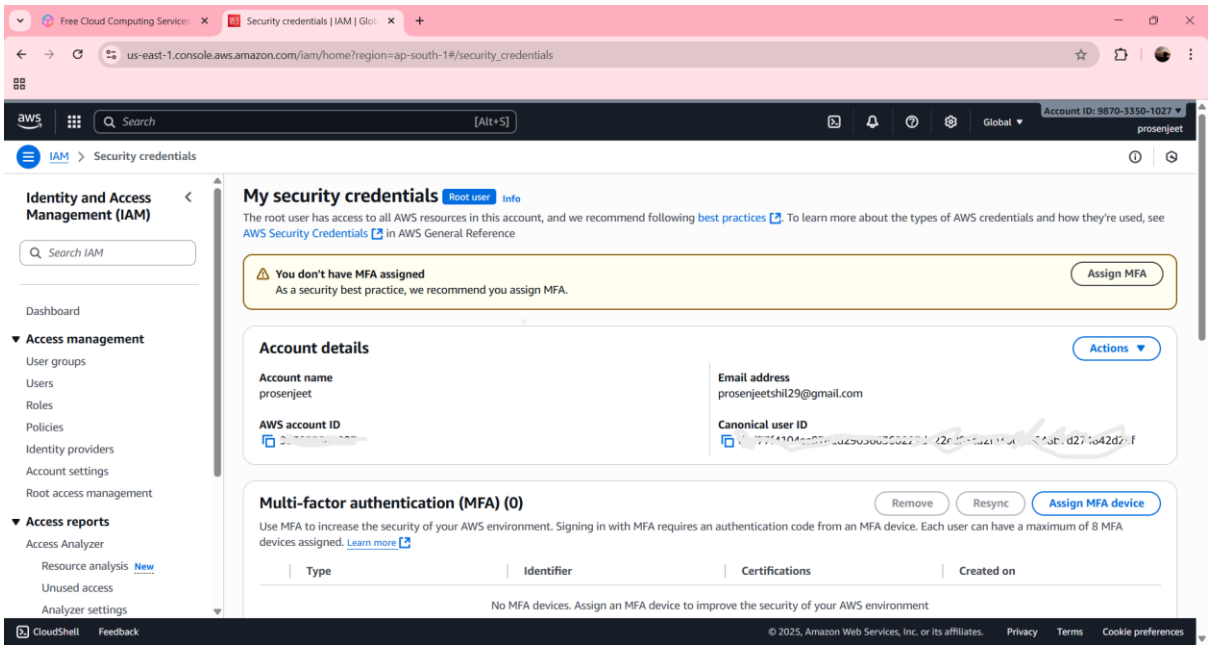
Step 6: Reopen https://aws.amazon.com/ and go to **My Account → AWS Management Console.**

Step 7: Enter your registered email ID and password to sign in.

Step 8: Once logged in, you will reach the **AWS Management Console** dashboard.

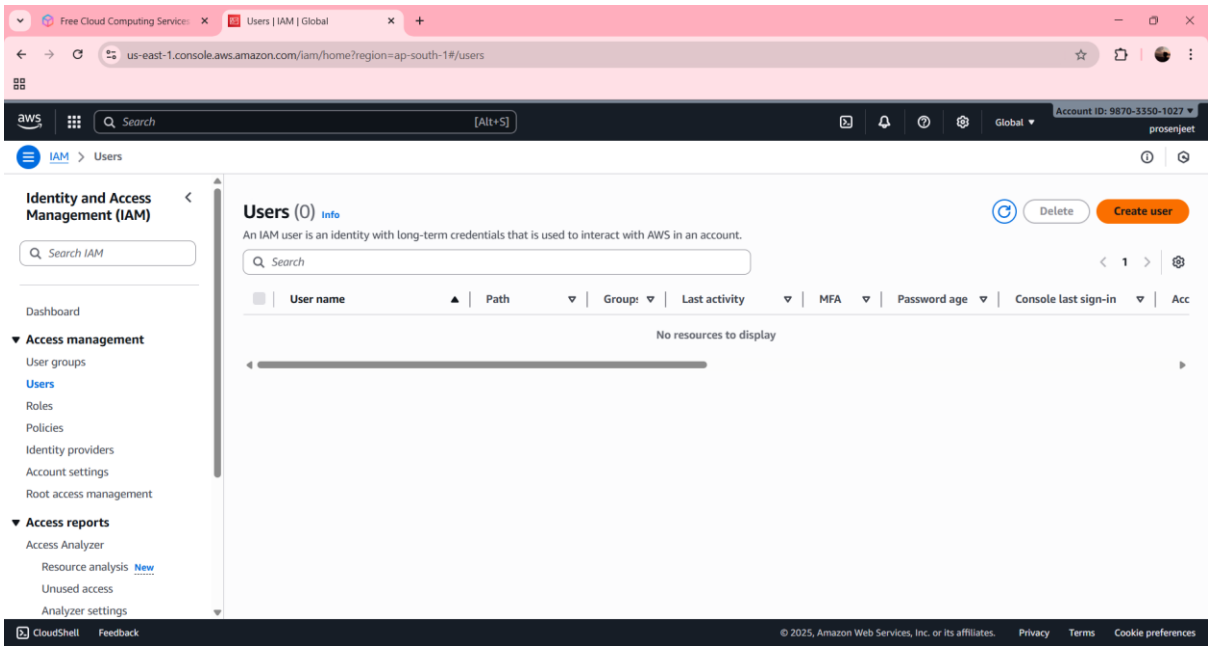


Step 9: Go to **My Security Credentials**.



Step 10: Click on **Users** from the IAM Dashboard.

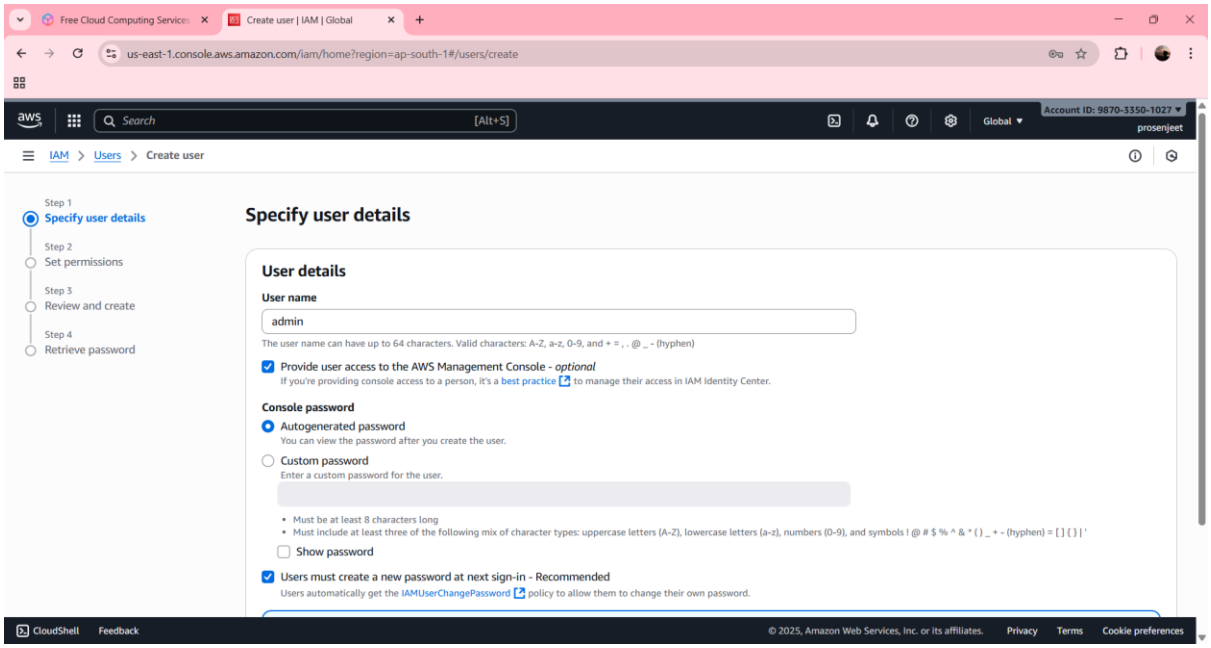
Step 11: Click on **Add User**.

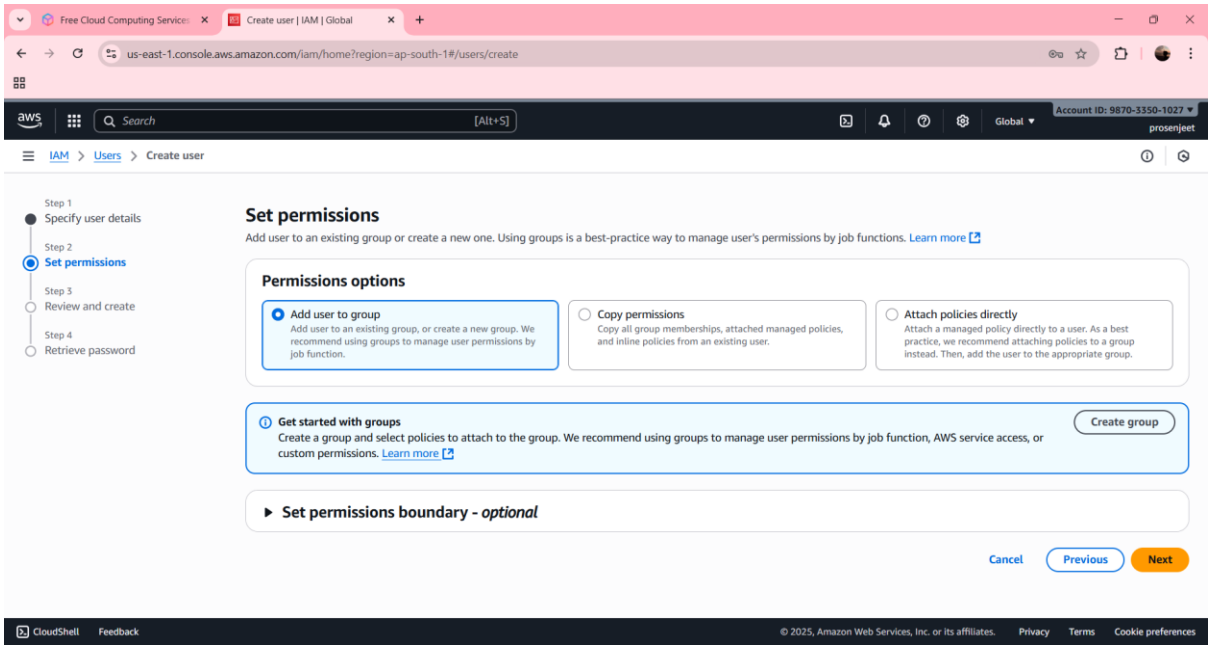


Step 12: Enter a **User Name**, select:

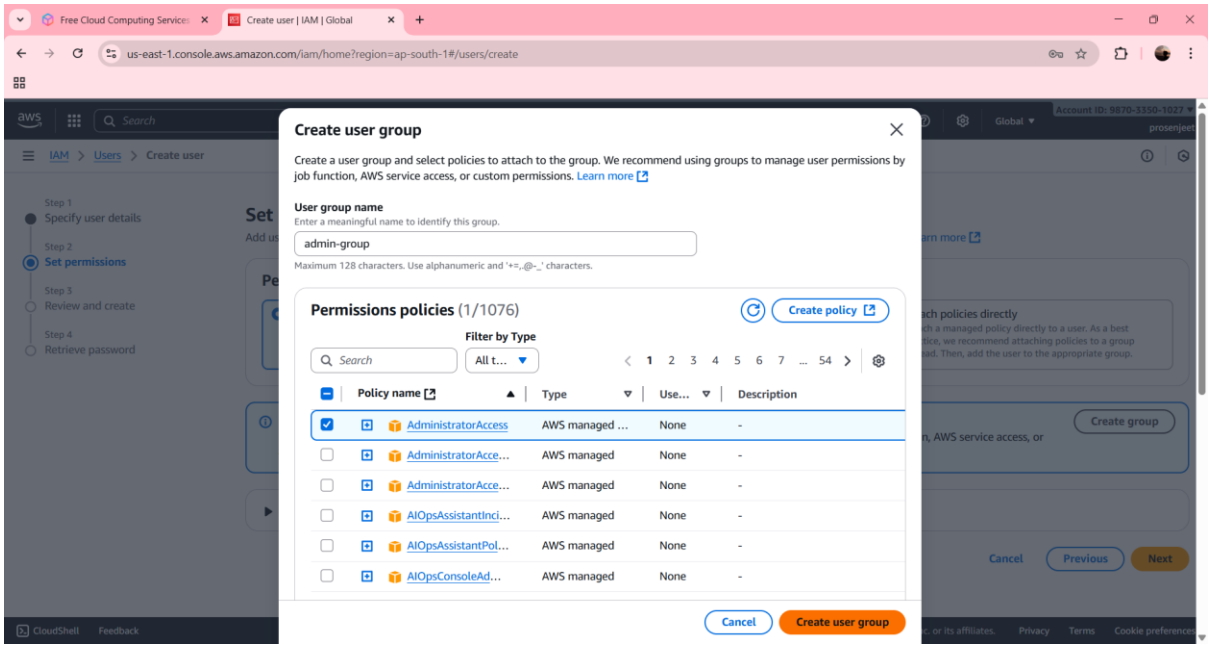
- **Programmatic Access**
- **AWS Management Console Access**

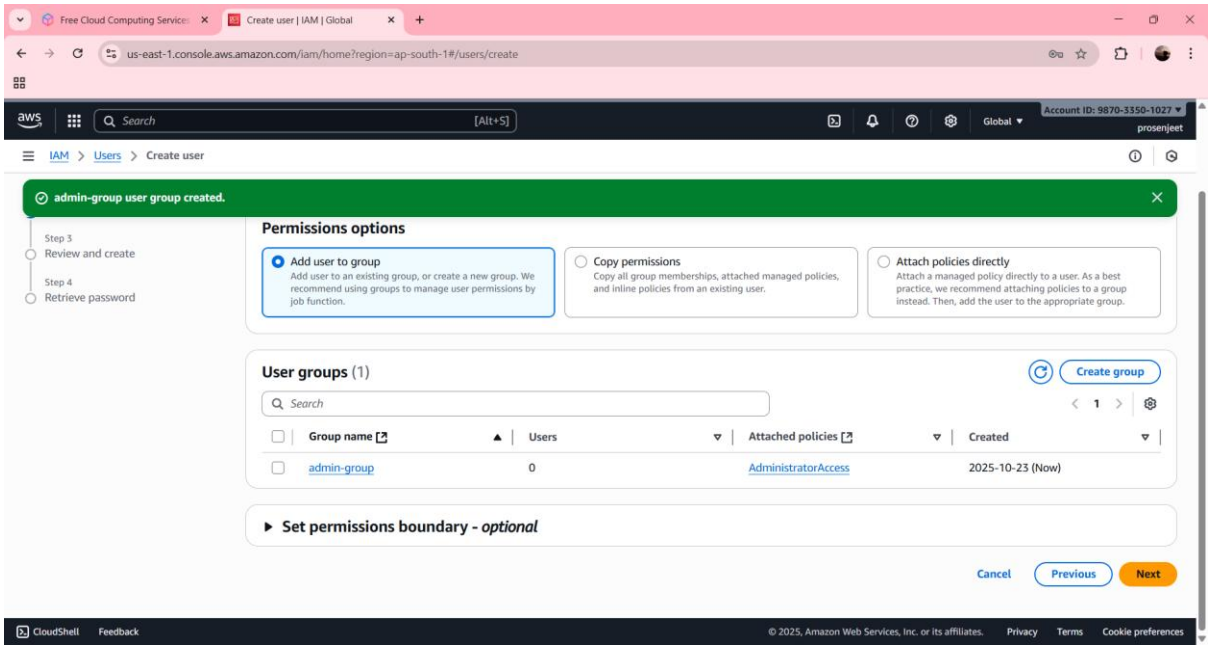
Then, create a **custom password** → Click **Next: Permissions**.





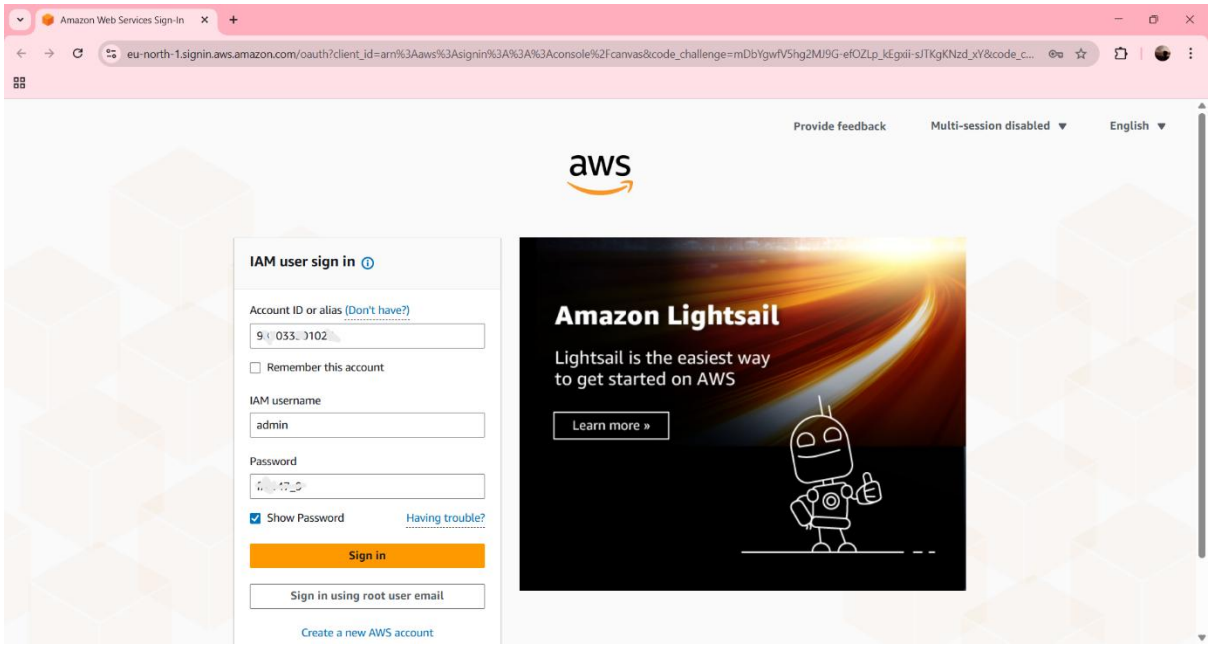
Step 13: Click on **Create Group**, provide a group name, and attach permissions (e.g., AdministratorAccess).
Click **Create Group → Next: Tags** (leave blank) → **Next: Review → Create User**.



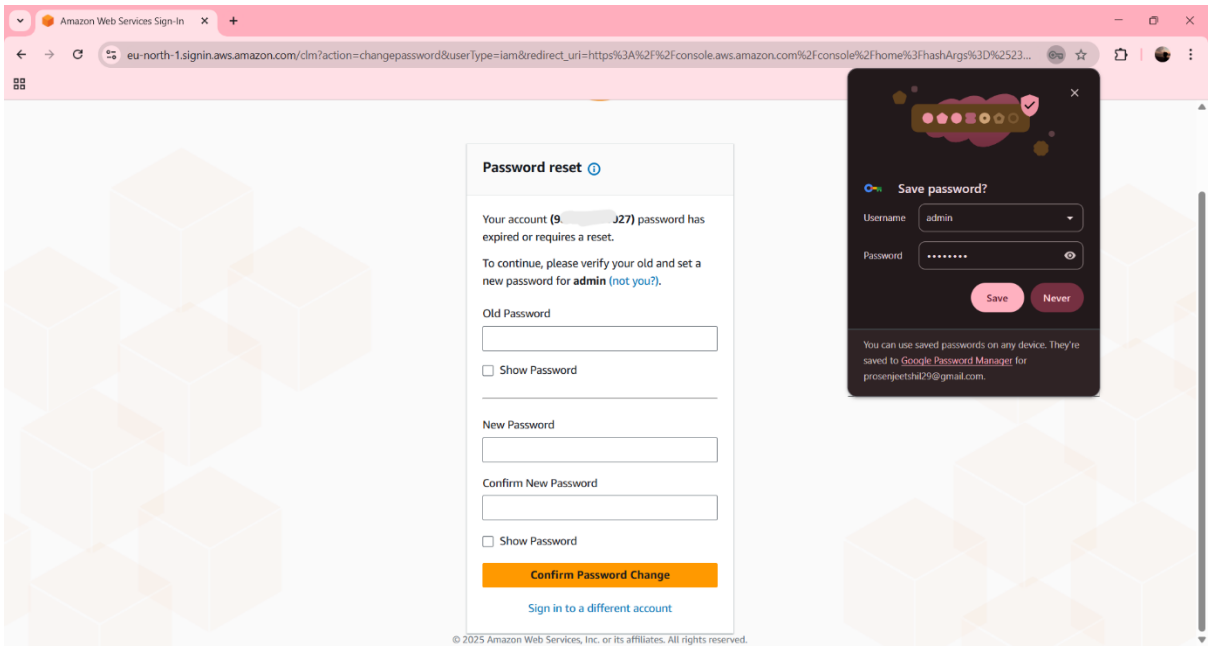


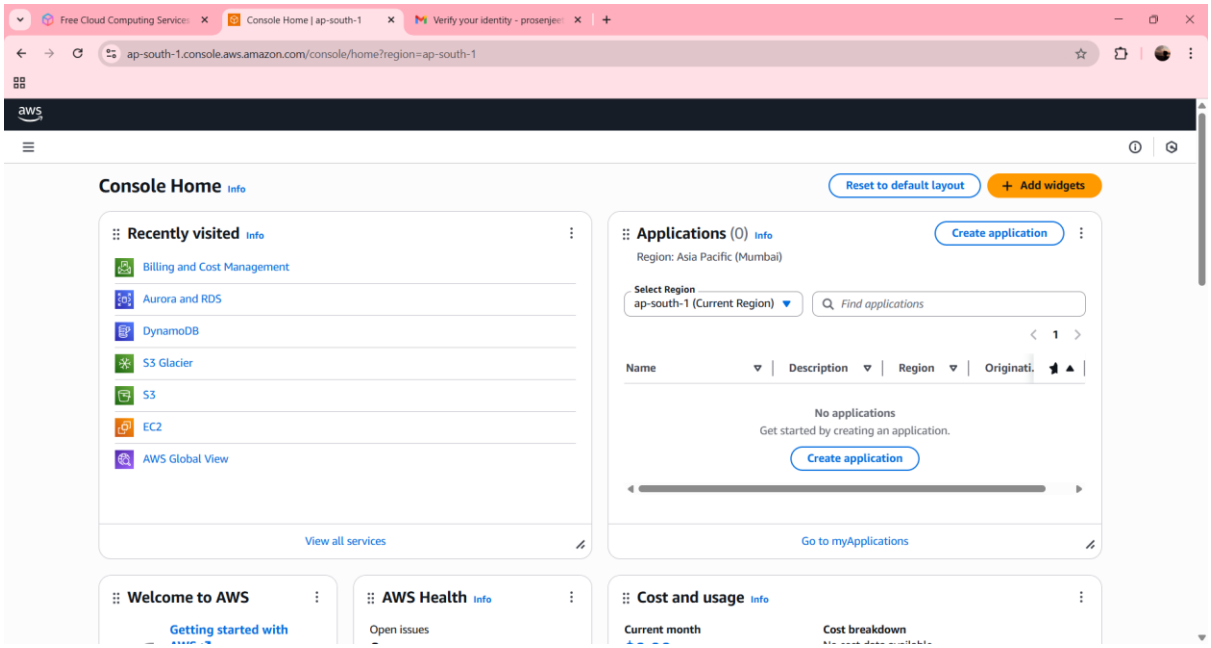
Step 14: Click **Close**, then **copy the Account ID**.

Step 15: Log out from the admin account.
Go to the AWS login page again and sign in as the **newly created user** using the Account ID, username, and password.



Step 16: On first login, AWS will prompt to **change the password**. After doing so, the user will be redirected to the **AWS Management Console**, confirming successful identity management.




Free Cloud Computing Services
Console Home | ap-south-1
Verify your identity - prosenjeet
ap-south-1.console.aws.amazon.com/console/home?region=ap-south-1
aws
Console Home Info
Reset to default layout
+ Add widgets
Recently visited Info
Billing and Cost Management
Aurora and RDS
DynamoDB
S3 Glacier
S3
EC2
AWS Global View
View all services
Applications (0) Info
Create application
Region: Asia Pacific (Mumbai)
Select Region
ap-south-1 (Current Region)
Find applications
Name
Description
Region
Originati.
No applications
Get started by creating an application.
Create application
Go to myApplications
Welcome to AWS
Getting started with
AWS Health Info
Open issues
Cost and usage Info
Current month
Cost breakdown

EXPERIMENT NO: 13

CREATE A VIRTUAL MACHINE (VM) ON ANY CLOUD PROVIDER (AWS/AZURE/GCP) OF YOUR CHOICE WITH THE SPECIFICATIONS

EXPERIMENT NO: 13

Q] Create a virtual machine (VM) on any cloud provider (AWS/Azure/GCP) of your choice with the specifications.

Theory

What is a virtual machine?

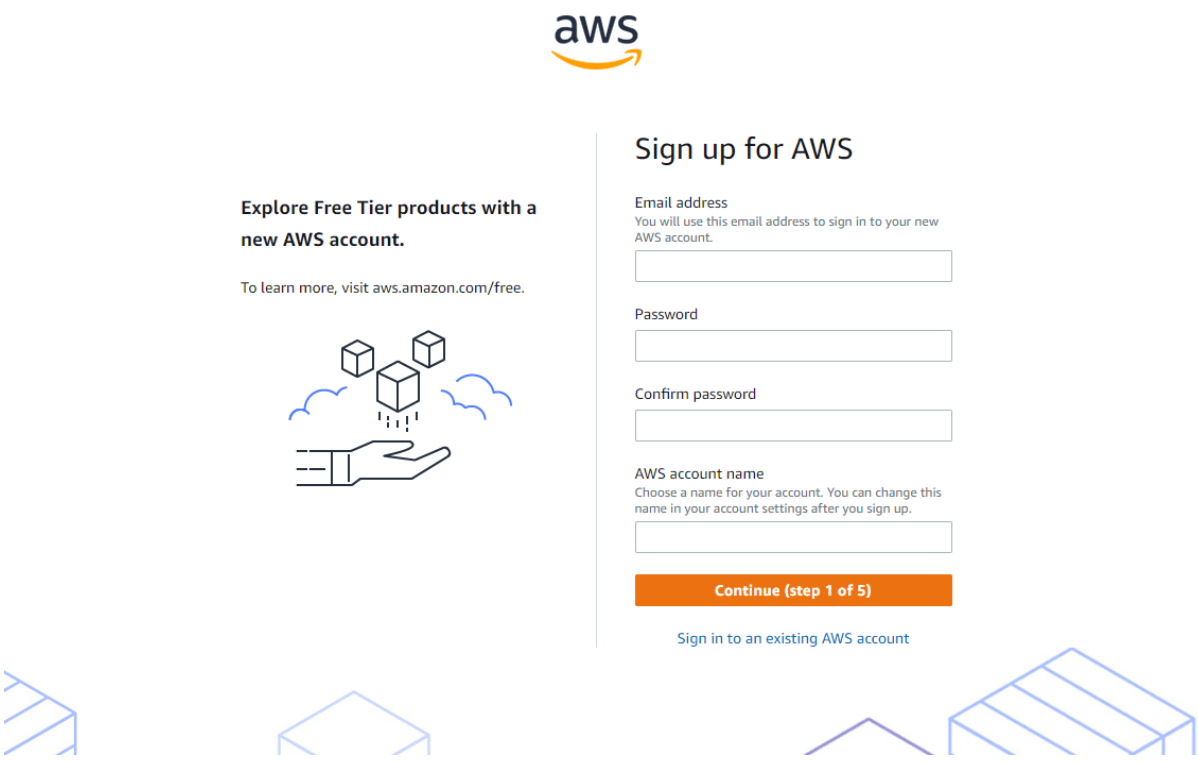
A virtual machine (VM) is a software demonstration and development model that simulates an operating system and its applications. VMs create a software simulation environment for testing software and application programs without physical hardware investment.

Procedure

Amazon Web Services provides scalable and affordable cloud computing services. You can join this platform for free and only pay for what you use. Use the following eight steps to set up a virtual machine with AWS.

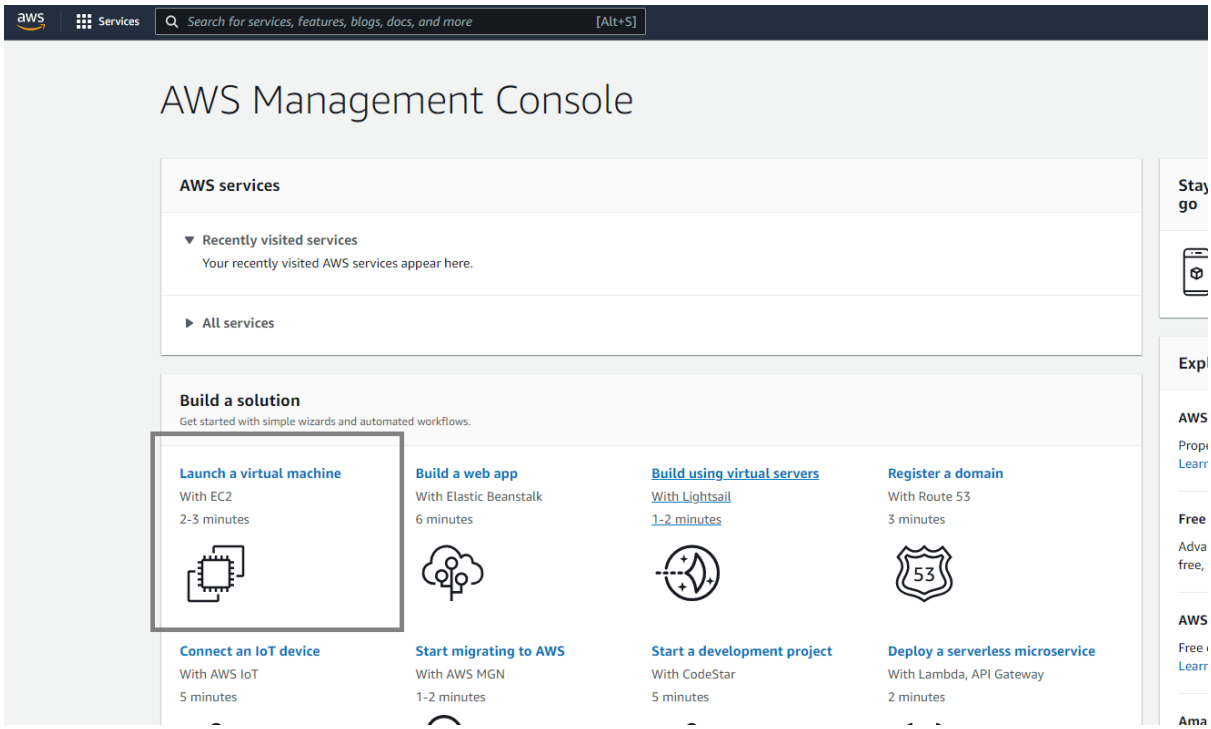
1. Create an AWS account

You can easily create an AWS account on the AWS Console. All new sign-ups get a free-tier offer.



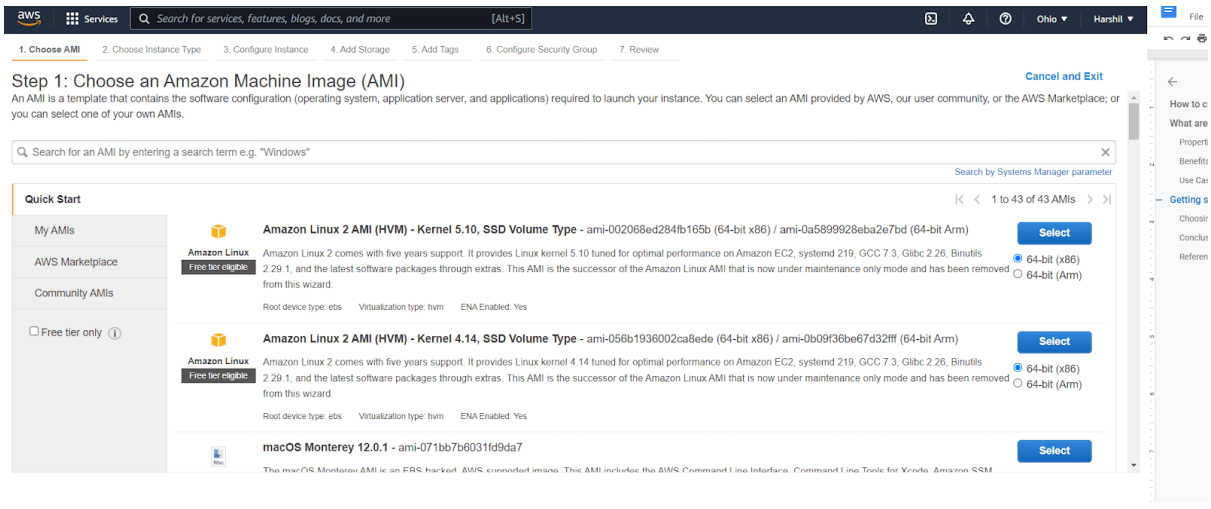
2. Launch AWS virtual machine

Once you finish setting up your account, you can click on the AWS logo on the top left corner or search “console” on the search bar. You’ll find a number of options in the AWS console. Select “Launch a virtual machine” to get started with VMs. If you’re a new user, it can take up to 24 hours for your account to activate.



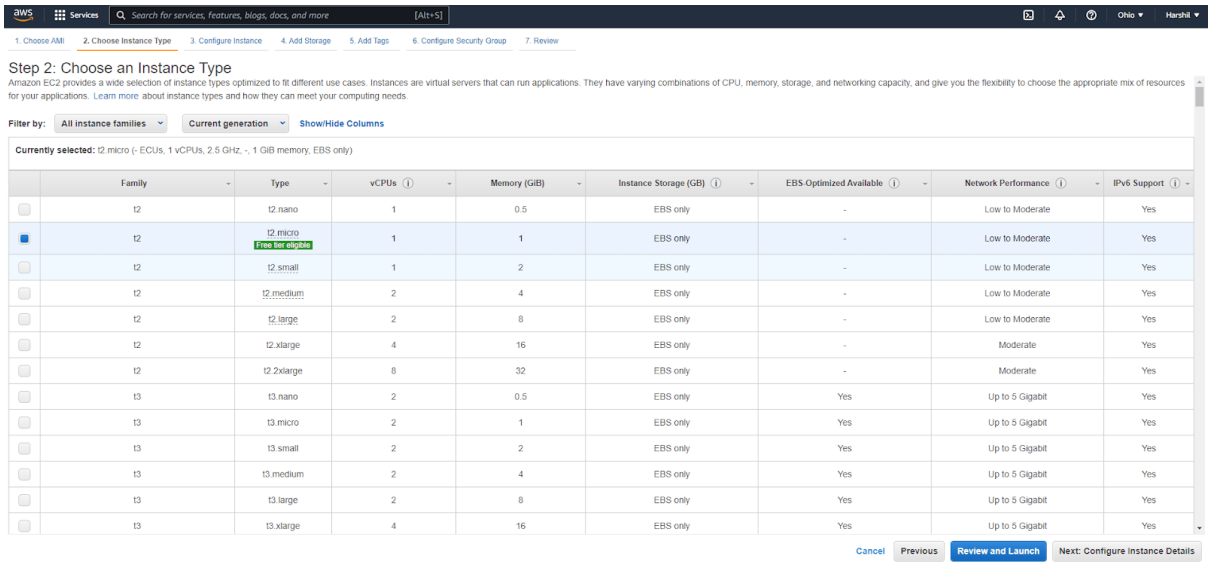
3. Choose AMI

Amazon Machine Image (AMI) highlights the software setup (OS, application server, and apps). You can select Mac, Linux, or Windows OS. We’ll look at the setup for Windows virtual machines here.

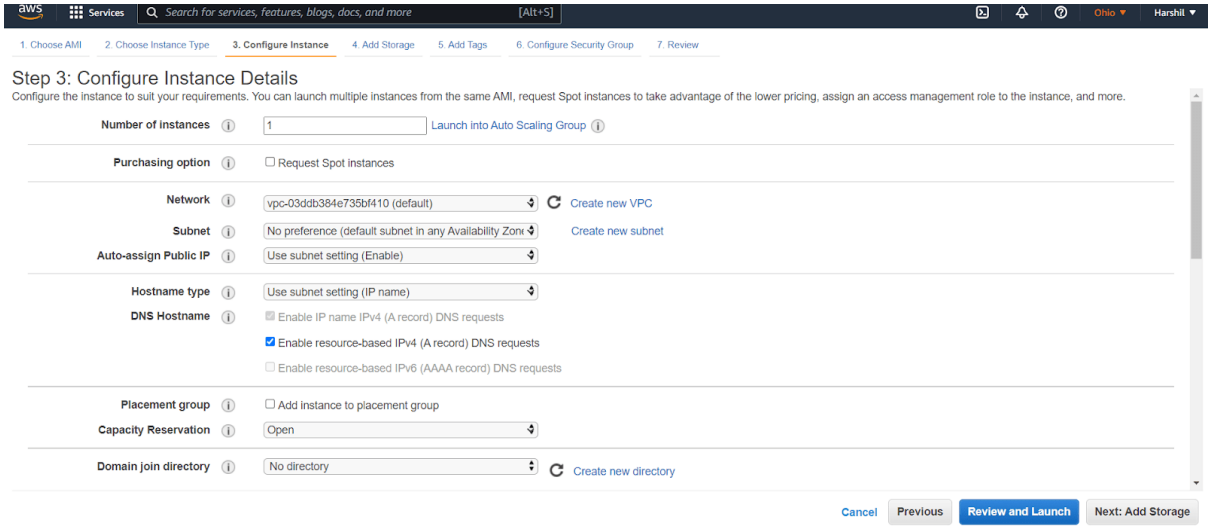


4. Choose and configure instance type

After choosing your operating system, you need to pick an instance type. Amazon EC2 offers many instance types tailored to specific use cases. An instance is a virtual server or virtual machine. They come in a variety of CPU, memory, storage, networking, and a lot more.



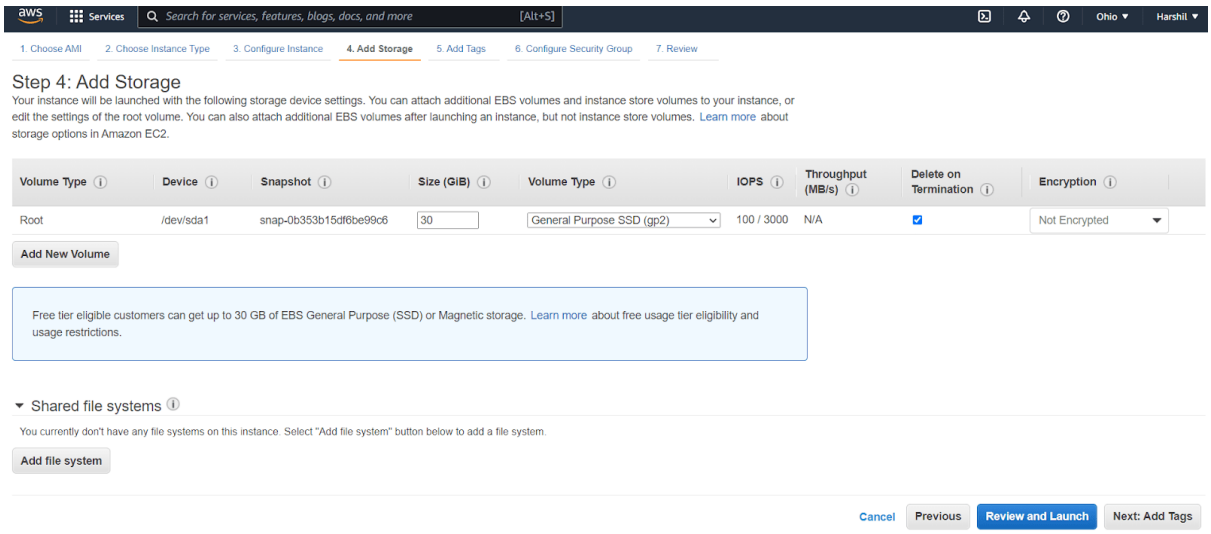
You can configure instance details, such as the number of instances, network, host type, and so on. Here, we'll use one instance and keep the remaining details default.



5. Add storage and tags

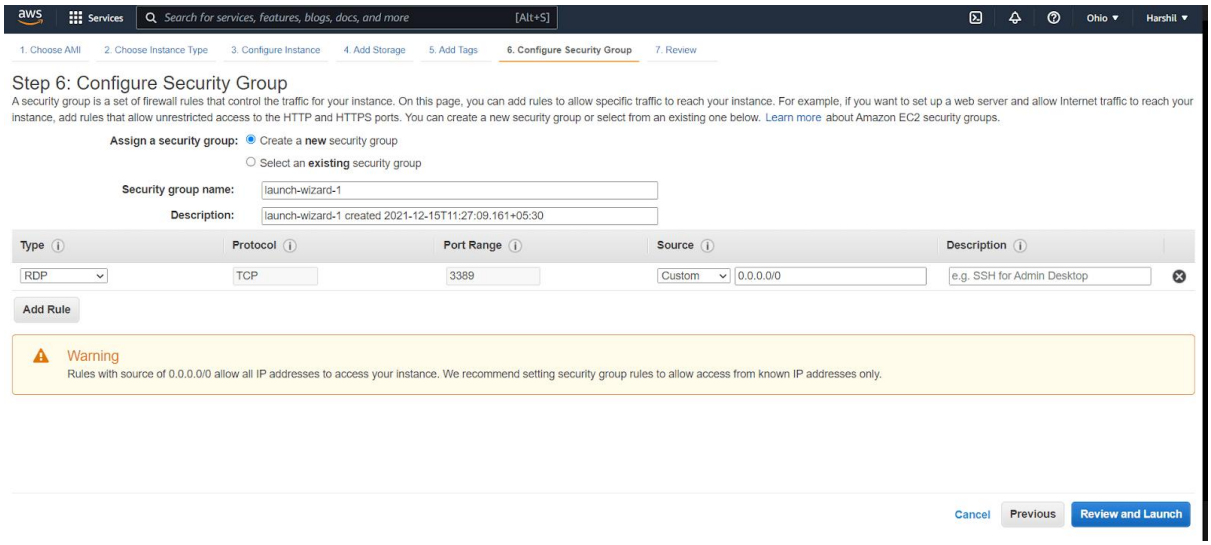
Once you configure an instance type, you can add or update storage info. AWS allows you to add more EBS volumes and instance store volumes, as well as change the root volume's parameters. Amazon Elastic Block Store (EBS) provides block-level storage volumes for use with EC2 instances. It behaves like raw, unformatted block devices. You can mount these volumes as devices on your instances.

The next step is adding tags. A tag is a label applied to an AWS resource. Each tag has a key and an optional value, which users define.



6. Configure security

A security group is a set of firewall rules that control data entering and exiting your instance. You may either recreate it or pick an existing security group.

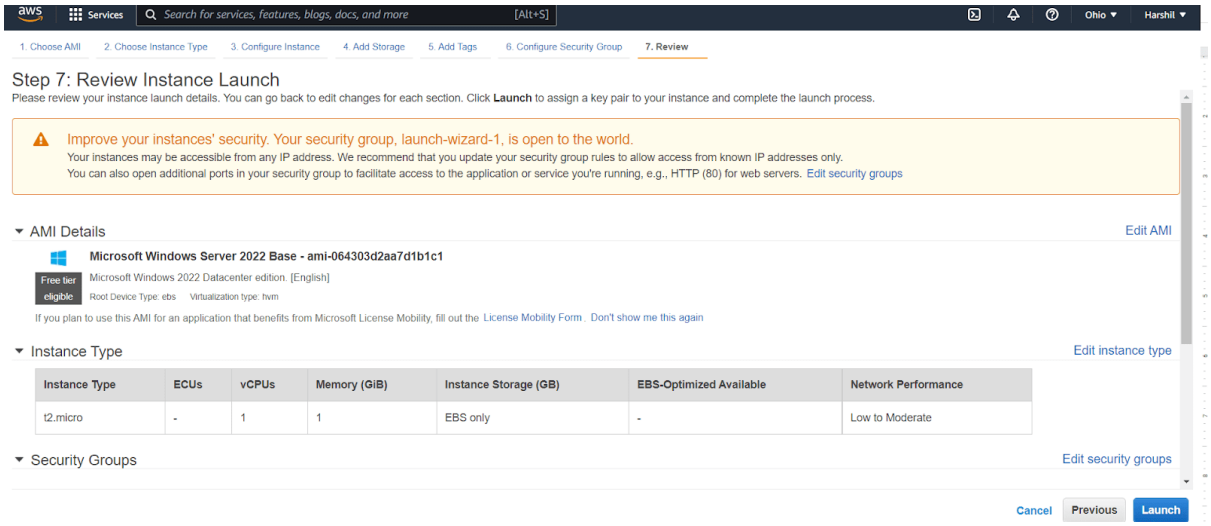


Security is a major concern when working on public clouds like AWS and Google Cloud Platform (GCP). Attackers can launch different attacks on public cloud deployments for lack of provider security. These attacks include DOS, DDOS, website defacement, and brute-force.

- Public clouds have poor security, but with the right set of rules, they can be improved.
- Public clouds offer limited customization. Clients can choose the operating system and size of the virtual machine.
- Cloud data breaches are frequently caused by misconfigured cloud security settings. Many companies' cloud security posture management solutions are insufficient for safeguarding their cloud-based infrastructure.

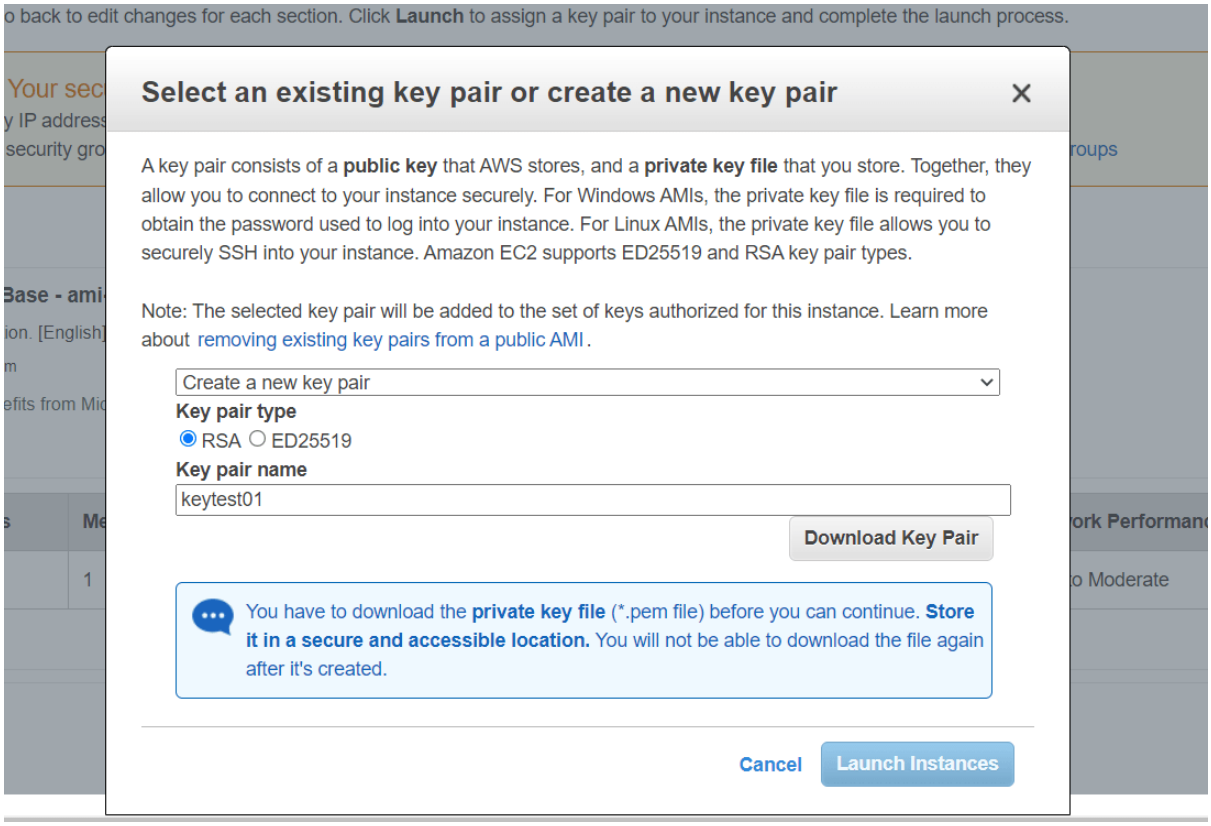
7. Review and launch your AWS virtual machine

The final step in creating an AWS virtual machine is to go through your instance details. Make sure every detail is correct, then click “Launch”.



When you click “Launch,” you need to provide a key. To create a new key, select “Create a new key pair” from the drop-down menu and set a key name, for example, keytask, keytest1, and so on. Make sure you download “key pair” before launching your instance.

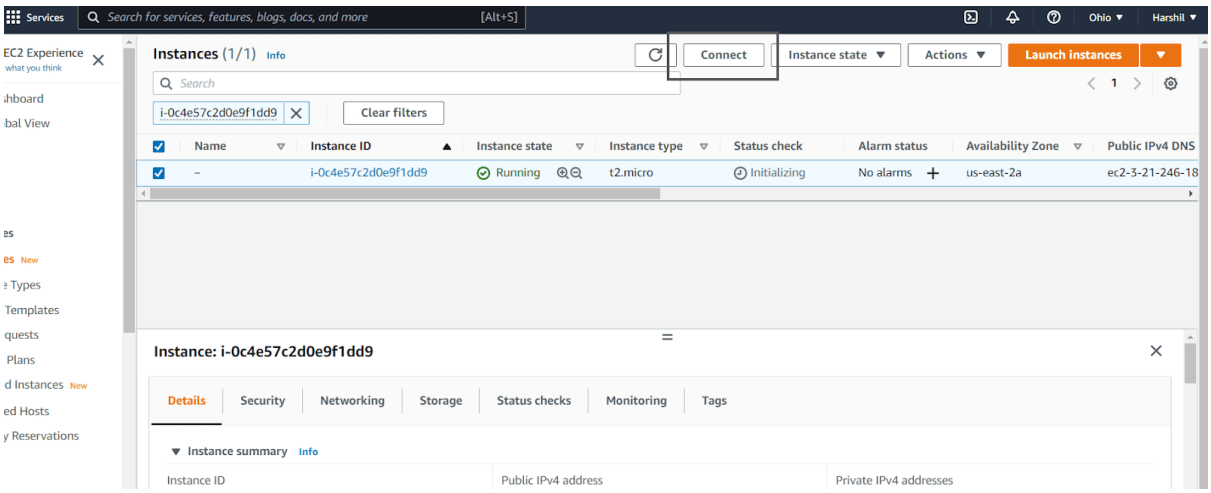
A key pair is made up of a public key stored by AWS and your private key file. They work together to allow you to connect to your instance safely.



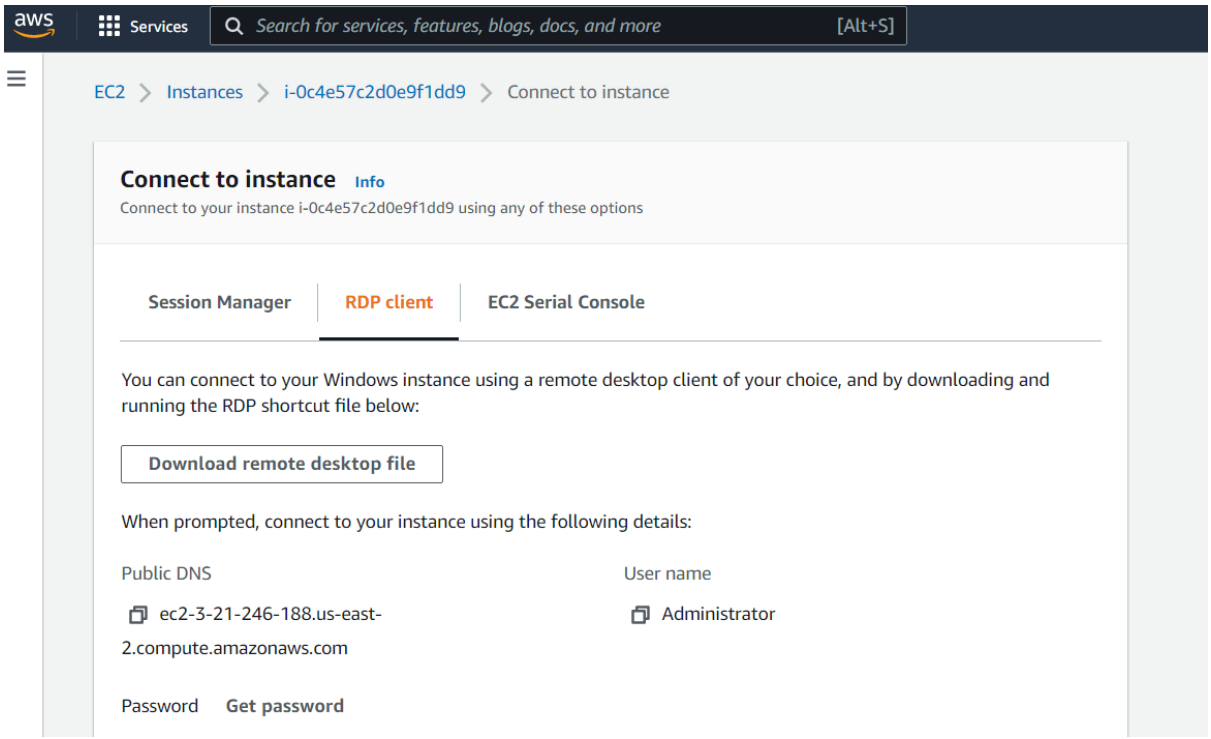
Voila! You successfully created and launched a virtual machine on AWS. Now, check the launch status and connect to an instance.

8. Connect to an instance

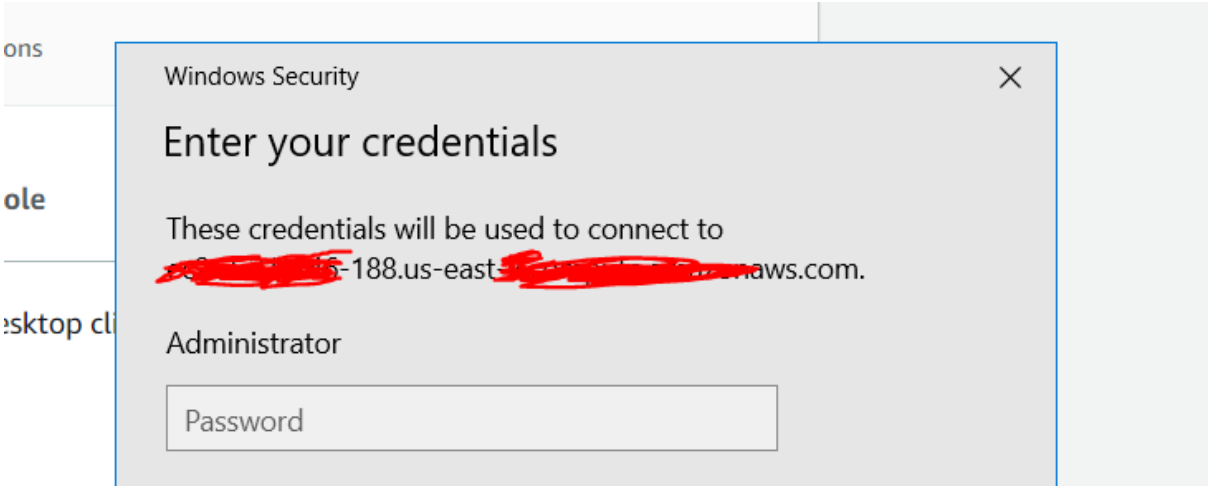
After starting the instance, you can check the status using “Dashboard>Instances”. Select your instance in the instance dashboard and click “Connect”.



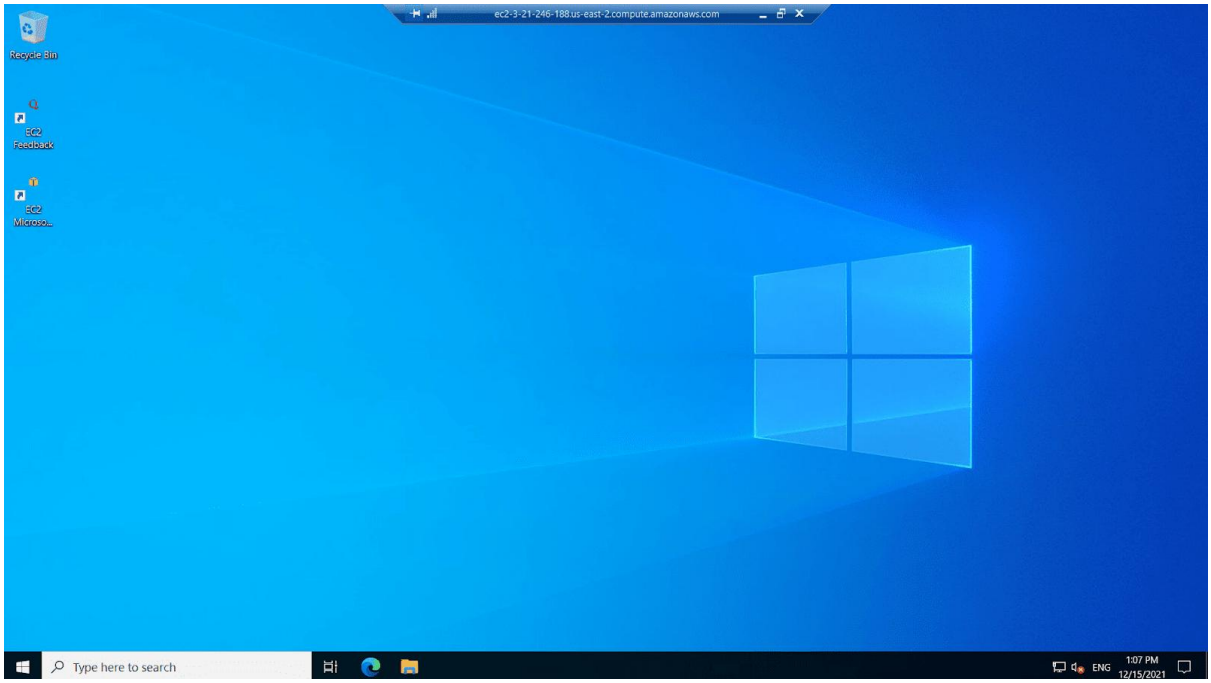
Select “RDP client,” click “Get password,” then upload the key pair downloaded when the instance launched (in step 7). After uploading the file, click “decrypt password” and download the remote desktop file.



Open the downloaded file and enter your password.



You should now see a screen similar to the one below, indicating that your AWS Windows virtual machine successfully launched!



EXPERIMENT NO: 14

INSTALL VIRTUAL BOX / VMWARE / EQUIVALENT OPEN-SOURCE CLOUD WORKSTATION WITH DIFFERENT FLAVOURS OF LINUX OR WINDOWS OS ON TOP OF WINDOWS 8 AND ABOVE.

EXPERIMENT NO: 14

Q] Install Virtual Box / VMware / Equivalent open-source cloud Workstation with different flavours of Linux or Windows OS on top of Windows 8 and above.

Theory

Introduction

VMware Workstation is software that helps you run different operating systems on one computer. It creates virtual machines (VMs), which act like separate computers inside your main system. This makes it easy to test new software, try out different operating systems, and develop applications without needing more hardware. It's popular with IT professionals and developers because it helps them work more easily and safely by managing different systems all in one place.

What is VMware Workstation?

VMware Workstation is a program that lets you run different operating systems, like Windows or Linux, on one computer at the same time. It creates "virtual machines" (VMs), which work like separate computers inside your main computer. This is useful for testing software, learning new systems, or doing development work without needing more computers.

Install VMware Workstation on Windows 11

VMware Workstation is a useful program that lets you run several virtual computers (VMs) on one main computer. Whether you need to test software, try out different operating systems, or set up a development space, VMware makes it simple to create and manage VMs. In this guide, we will show you the easy steps to install VMware Workstation on Windows 11.

System Requirements

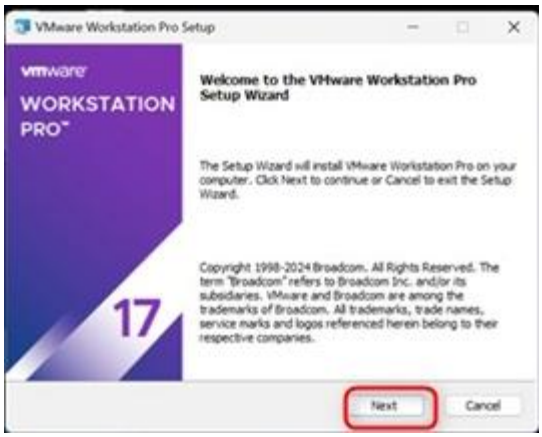
Requirement	Details
Operating System	Windows 11 (64-bit)
Processor	64-bit processor that supports virtualization (Intel VT-x/AMD-V)
RAM	At least 4 GB (8 GB or more is better)
Storage	1 GB of free disk space for the program plus extra space for each VM
Graphics	DirectX 11 or newer compatible graphics card

Step 1. Needs to Download VMware Workstation.

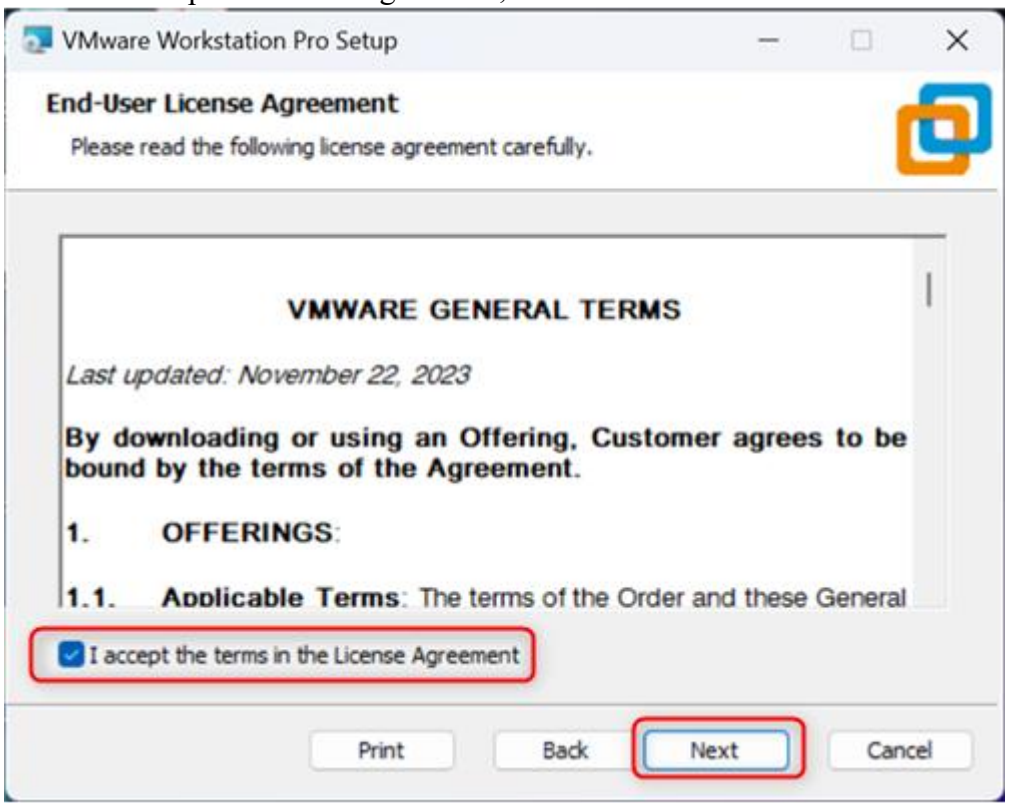
1. Open your web browser and go to the VMware website: https://blogs.vmware.com/workstation/2024/05/vmware-workstation-pro-now-available-free-for-personal-use.html
2. Then Download based on your.
3. Save the installer file on your computer.

Step 2. Install VMware Workstation.

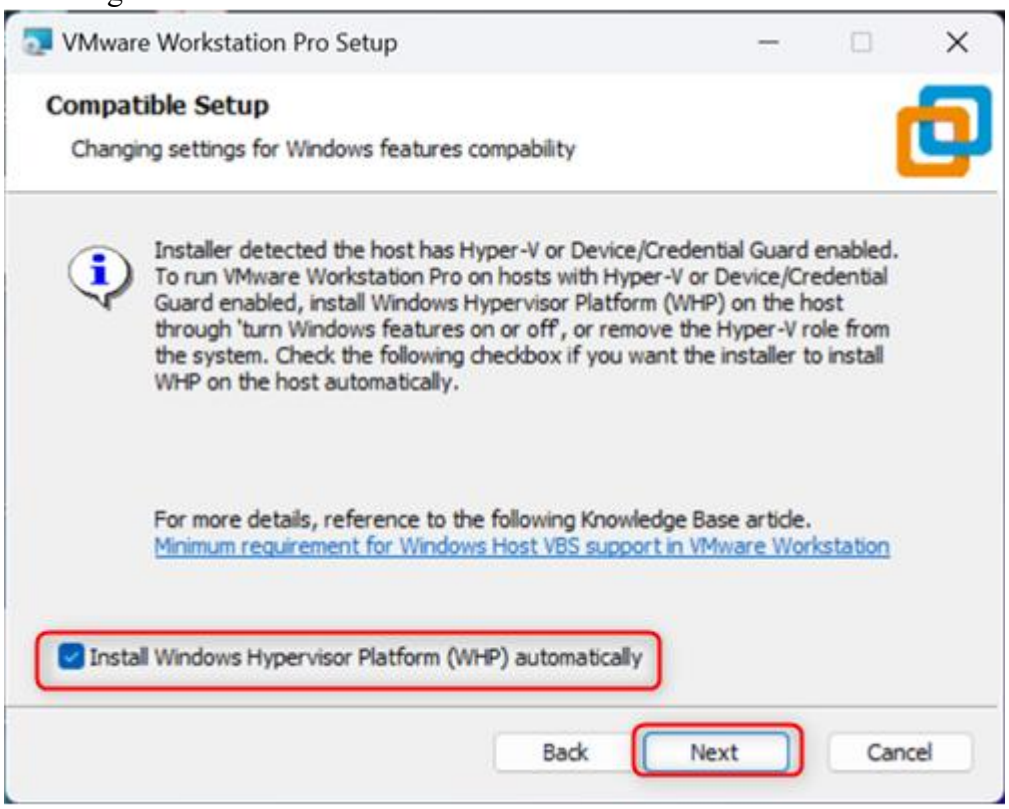
1. Find the downloaded installer file
2. Right-click on the file and select Run as administrator.
3. The setup wizard will open. Click Next to continue.



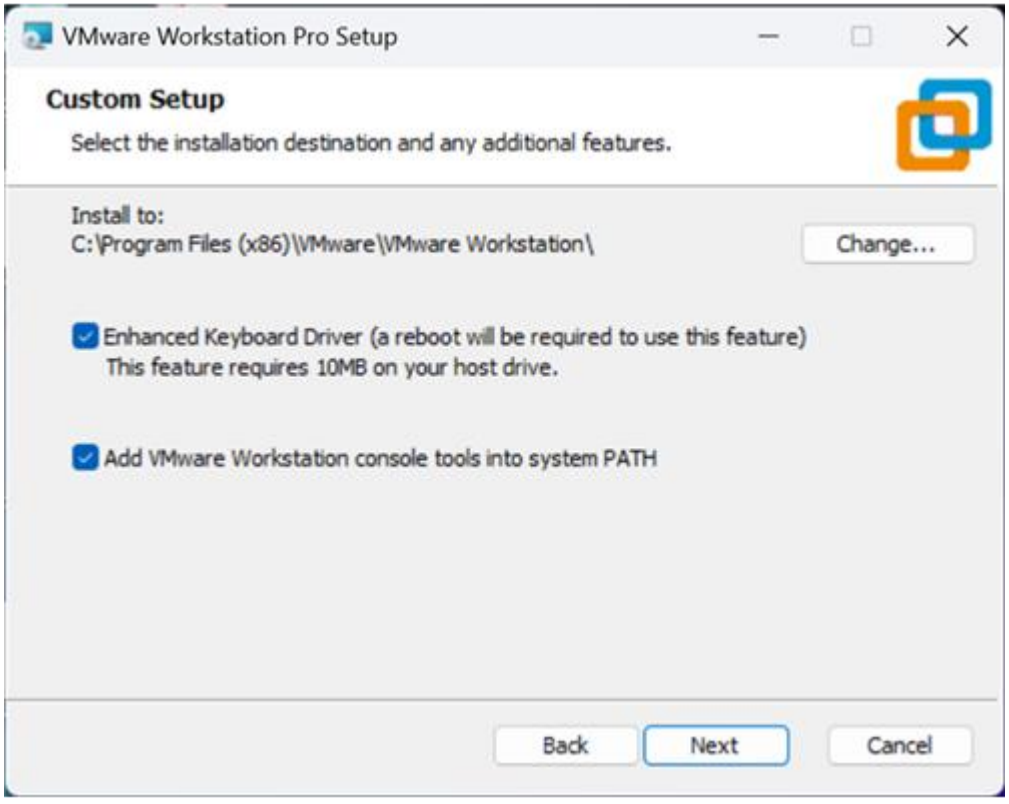
4. Read and accept the license agreement, then click Next.



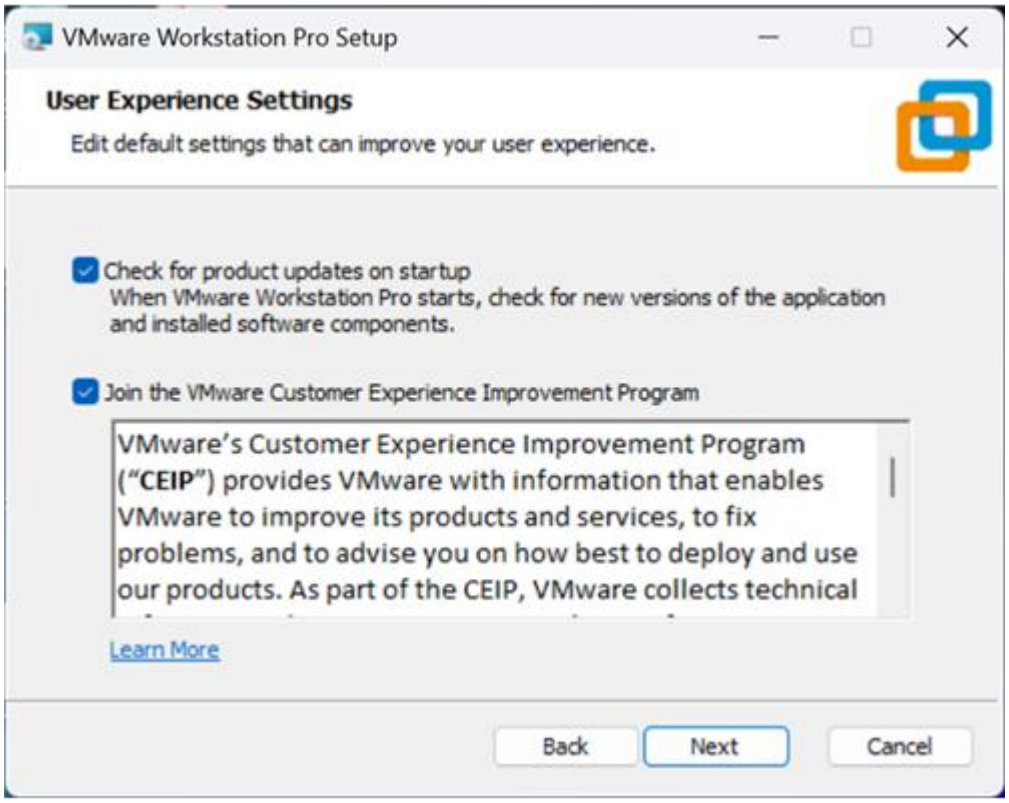
5. Selecting WHP to enable Windows Features for the virtualization.



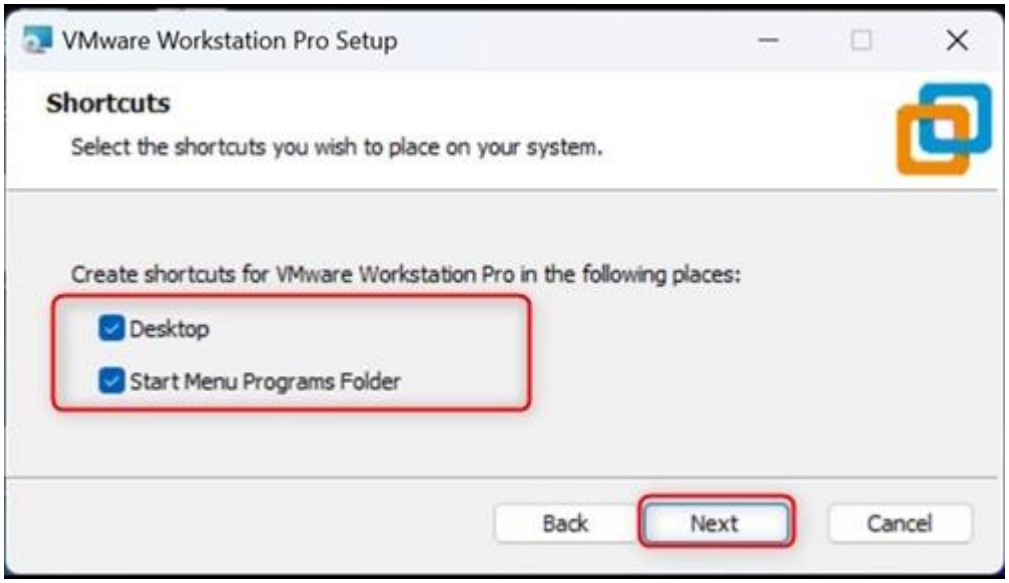
6. Choose where to install the program (or leave it as default).
7. Enable Enhanced Keyboard Driver: Better keyboard support for VMs.
8. Then click Next.



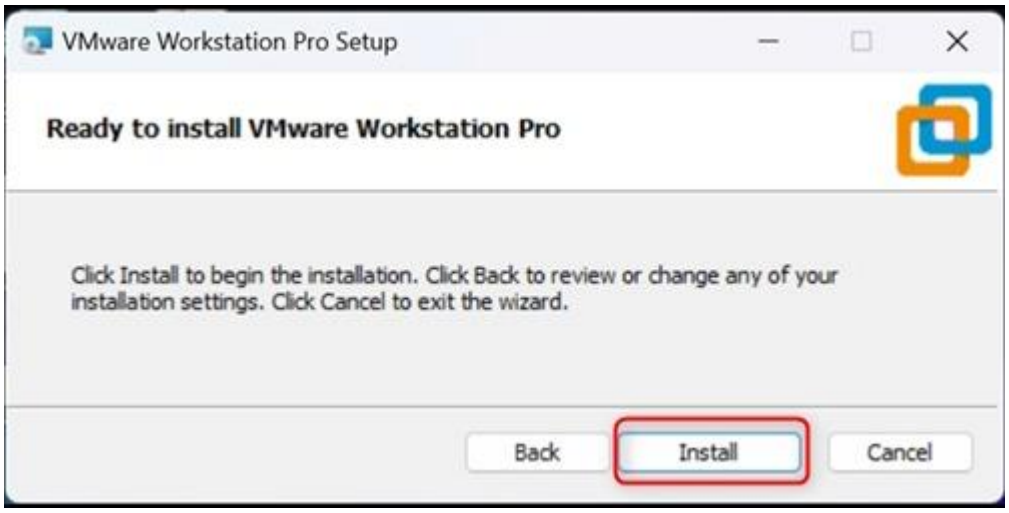
9. Select the Check for product updates on startup to Check the software updates automatically.
10. Then click Next.



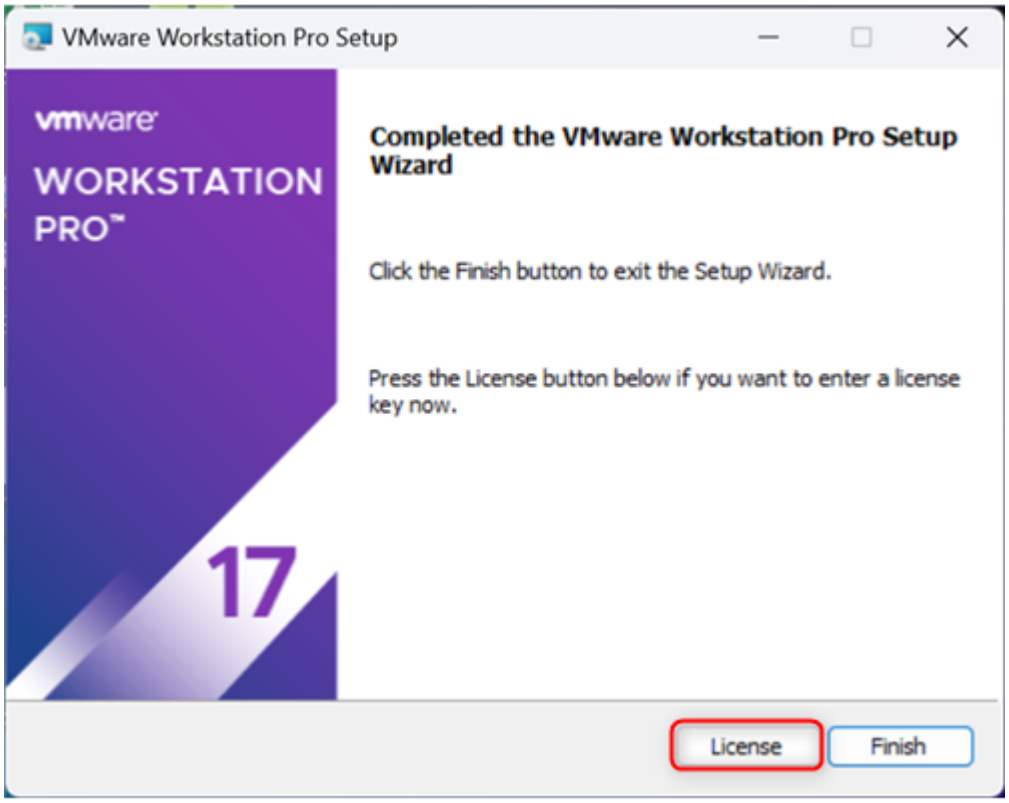
11. Decide if you want desktop or Start menu shortcuts,
12. Then click Next.



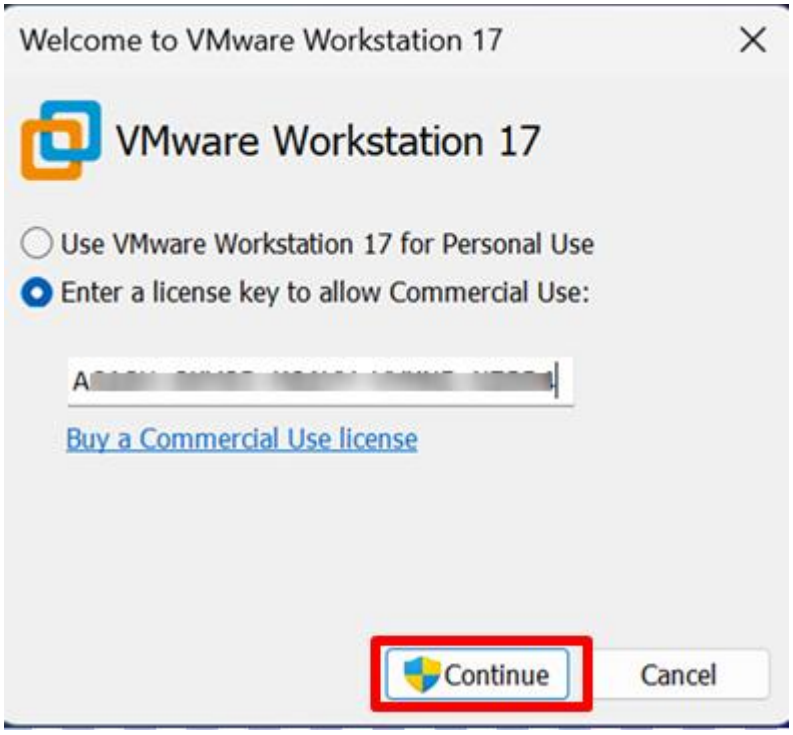
13. Click Install to begin the installation.



14. Click Finish to complete the Wizard.



15. It's required to restart, then click Yes.



Step 3. Activate VMware Workstation.

- If you installed VMware Workstation Pro, you will be asked to enter a license key to activate it.
- If you installed VMware Workstation Player, you can use the free version for personal use or upgrade to Pro.

After the computer restarts, Double-Click to Run VMware Workstation.

Then, Click Finish to enjoy the product.