Implementation of Remote Method Invocation (RMI) in a Cloud Environment

ROLL NUMBER: 2023115015

NAME: NAVINESHARAN S

1. Introduction

- Remote Method Invocation (RMI) is a Java-based distributed computing mechanism that allows an object running in one Java Virtual Machine (JVM) to invoke methods of an object running in another JVM, possibly on a different machine.
  In this experiment, Java RMI is implemented to demonstrate remote communication between a client system and a cloud-hosted server using a calculator service

---

2. Objective

To design and implement a Java RMI-based distributed application

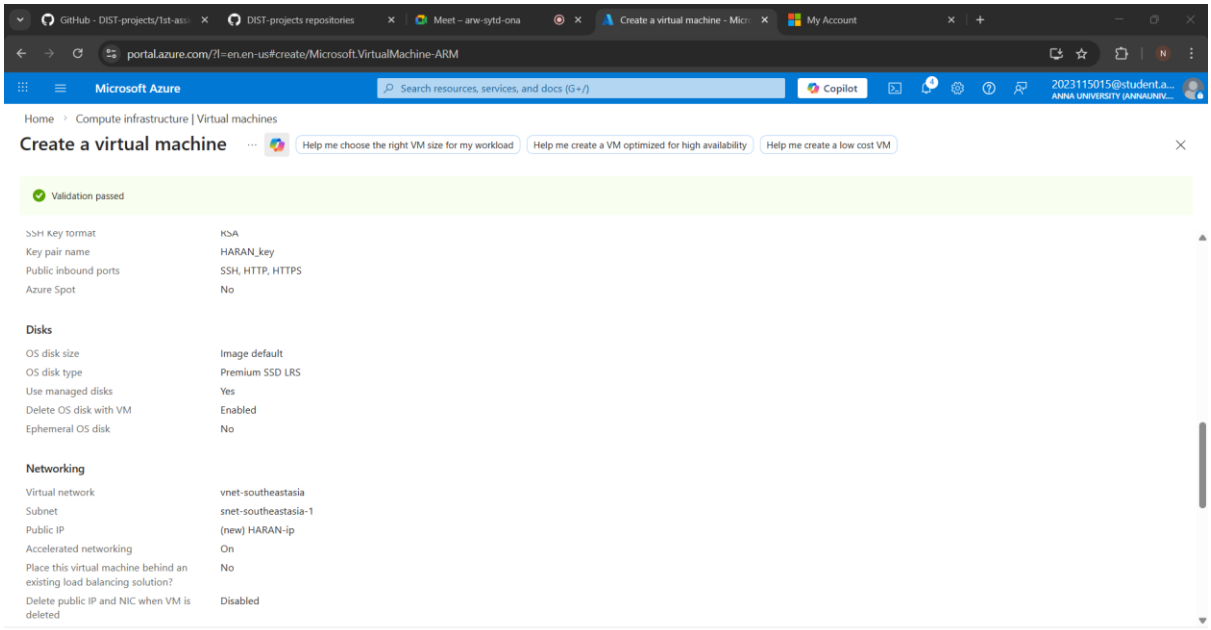
- To host the RMI server in a cloud environment
- To enable a client to remotely invoke methods from the cloud server
- To verify successful remote method execution and result retrieval

---

3. Cloud Environment Setup

- Cloud Platform: **Microsoft Azure Virtual Machine**
- Operating System (Server): **Ubuntu Linux**
- Client System: **Windows OS**
- Java Version: **JDK 17**
- Network Configuration: Public IP-based communication

The RMI server is deployed on an Azure Virtual Machine, while the client runs on a local machine. Communication happens over the public internet using TCP/IP.



---

4. RMI Architecture

The RMI architecture consists of:

- **Remote Interface**: Declares methods that can be invoked remotely
- **Remote Object Implementation**: Implements the interface methods
- **RMI Registry**: Maintains references to remote objects
- **Client**: Looks up the remote object and invokes methods

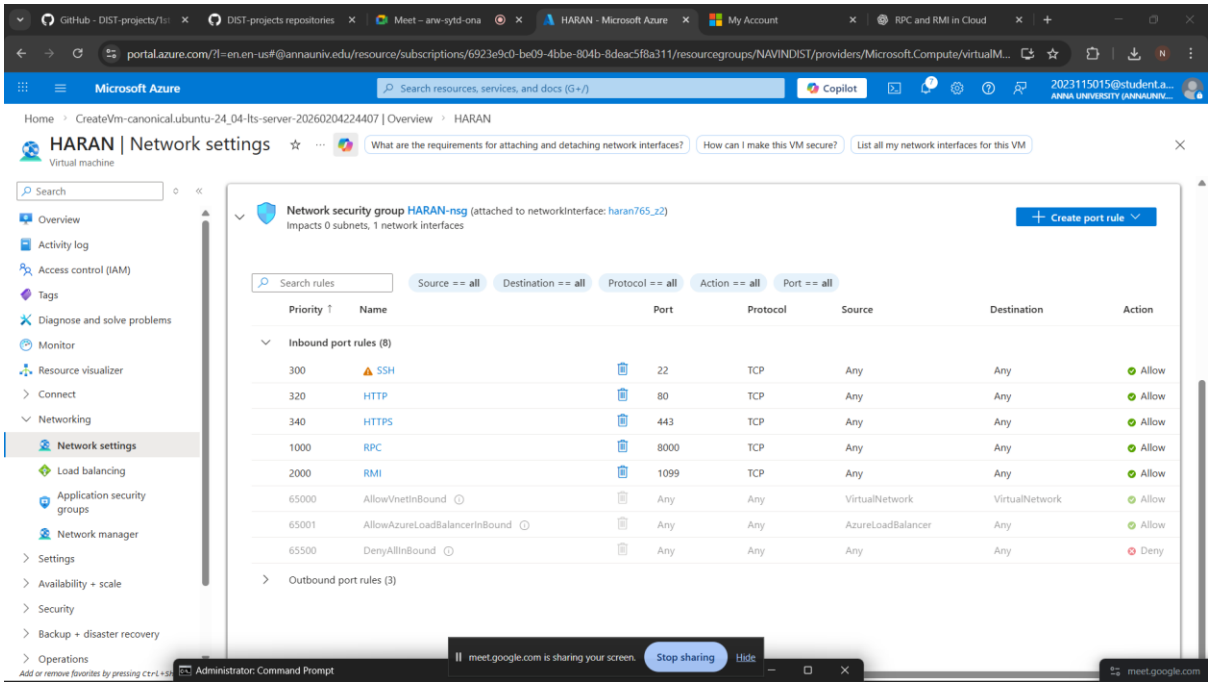
---

5. Port Selection and Configuration

- **Port Used**: 1099
- Purpose:
    - RMI Registry
    - Remote object communication

To avoid dynamic port issues in cloud environments, the RMI object is explicitly bound to port 1099. This simplifies firewall configuration and ensures reliable connectivity.

Azure Network Security Group (NSG) is configured to allow inbound TCP traffic on port 1099.



---

6. Server Implementation

The server performs the following steps:

1. Sets the public IP as the RMI hostname
2. Creates the remote object
3. Exports the object on a fixed port (1099)
4. Registers the object with the RMI registry
5. Waits for client requests

The server continuously listens for remote method invocation requests from the client.

---

7. Client Implementation

The client performs the following:

1. Connects to the RMI registry using the cloud server's public IP
2. Looks up the registered remote object
3. Invokes remote methods (add, subtract, multiply, divide)
4. Receives computed results from the cloud server

The client does not require the server implementation class, only the remote interface.

---

8. Cloud Connection Establishment

- The client connects to the cloud server using the server's **public IP address**
- Communication is established over TCP/IP
- All method invocations are executed remotely on the cloud server JVM

Successful connection confirms distributed execution.

---

9. Output

Sample Output (Client Side):

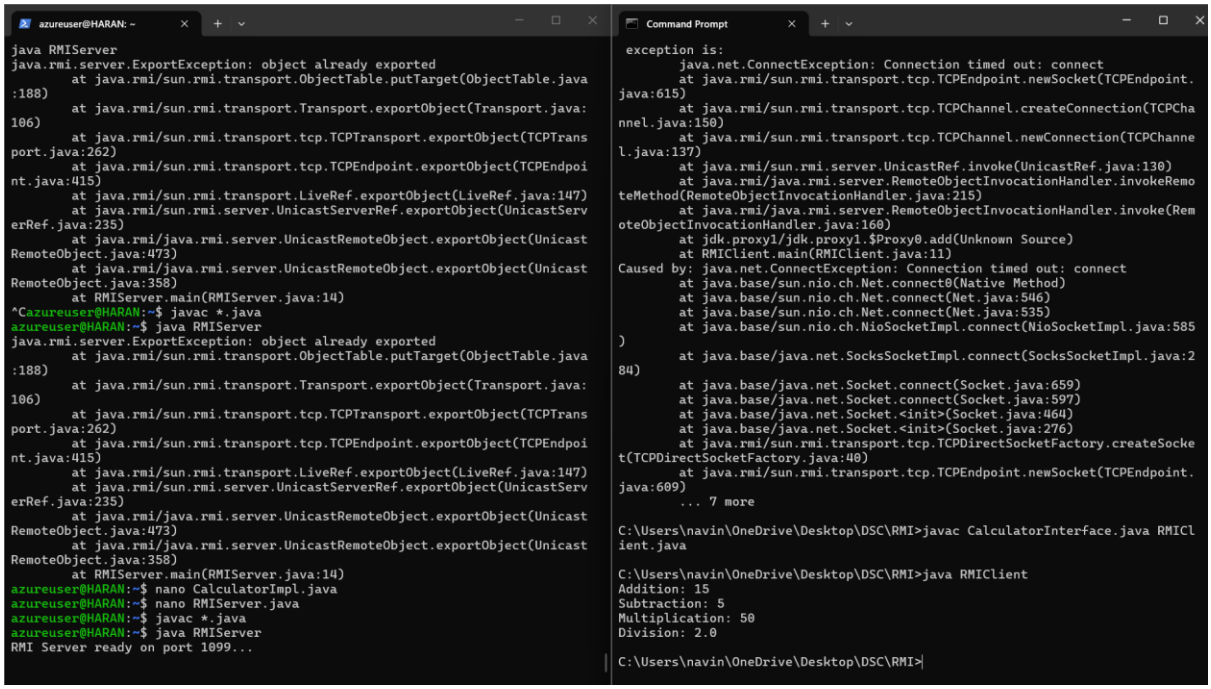
Addition: 15

Subtraction: 5

Multiplication: 50

Division: 2.0

This output confirms that method execution occurs on the cloud server and results are returned to the client.



---

10. Error Handling

- Network connection issues are handled using Java exception handling

- Division-by-zero cases are safely managed in server logic
- RMI lookup and invocation exceptions are caught and logged

---

11. Conclusion

The RMI-based distributed application was successfully implemented and deployed in a cloud environment. The client was able to remotely invoke methods from a cloud-hosted server using Java RMI. Proper port configuration and hostname binding ensured seamless communication. This experiment demonstrates the effectiveness of RMI for building distributed Java applications in cloud environments.