
RPC Implementation in Cloud Environment

Name:Rohit S

Roll No:2023115085

Aim

To design and implement a **Remote Method Invocation (RMI)** based **distributed application** using **Java**, where the **RMI server is hosted in a cloud environment (Microsoft Azure Virtual Machine)** and the **client executes on a local machine**, enabling invocation of remote methods over a network.

Software & Tools Used

Component	Description
Programming Language	Java
Distributed Model	Java RMI
Cloud Platform	Microsoft Azure
Server OS	Ubuntu Linux (Azure VM)
Client OS	Windows 10
JDK Version	OpenJDK 17
Network Port	1009

System Architecture

- RMI Server and Registry run on **Azure VM**
 - Client accesses remote object using **public IP**
 - Communication occurs over TCP/IP
-

Remote Methods Implemented

Method Name	Description
multiplyMatrix	Perform Matrix Multiplication
getPrimes	Print prime numbers in a range

Execution Steps

Step 1: Install Java on Azure VM

```
sudo apt update
```

```
sudo apt install openjdk-17-jdk -y
```

Step 2: Compile RMI Programs

```
javac *.java
```

Step 3: Run RMI Server

```
java RMIServer
```

Output:

RMI Server running on Azure VM at port 1099

```
Linux x + - _ 
⑤ ~ > ssh -i ~/azureuser.pem azureuser@98.70.25.35
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1017-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Thu Jan 29 19:07:38 UTC 2026

System load: 0.03      Processes:           160
Usage of /: 9.6% of 28.02GB  Users logged in:   1
Memory usage: 5%          IPv4 address for eth0: 172.17.0.4
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

34 updates can be applied immediately.
27 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

4 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Thu Jan 29 18:58:15 2026 from 49.206.15.107
azureuser@VM01:~$ cd RMI/
azureuser@VM01:~/RMI$ java RMIServer
RMI Server running on port 1099
```

Step 4: Configure Azure NSG

- Allow **Inbound TCP Port 8080**
 - Source: Any
 - Destination: Any

Home > Compute infrastructure | Virtual machines

Create a virtual machine

 Help me create a VM optimized for high availability | Help me choose the right VM size for my workload | Help me create a low cost VM

 Validation passed

Basics

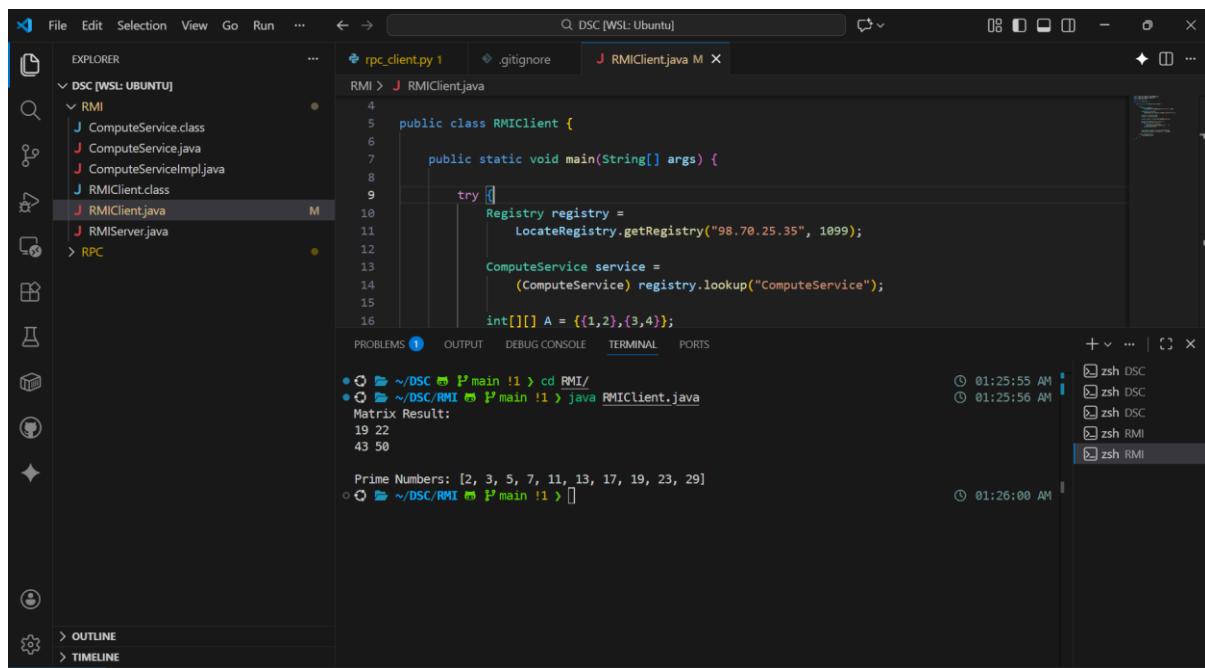
Subscription	Azure for Students
Resource group	Distributed-Assigments
Virtual machine name	VM01
Region	Central India
Availability options	Availability zone
Zone options	Self-selected zone
Availability zone	1
Security type	Trusted launch virtual machines
Enable secure boot	Yes
Enable vTPM	Yes
Integrity monitoring	No
Image	Ubuntu Server 24.04 LTS - Gen2
VM architecture	x64
Size	Standard D2s v3 (2 vcpus, 8 GiB memory)
Enable Hibernation	No
Authentication type	SSH public key
Username	azureuser
SSH Key format	RSA
Key pair name	azureuser
Public inbound ports	SSH, HTTP, HTTPS, RDP
Azure Spot	No

Next

< Previous | Next > | **Create** |  Give feedback

Step 5: Run Client on Local Machine

java RMIClient



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "DSC [WSL: UBUNTU]". The "RMI" folder contains "ComputeService.class", "ComputeService.java", "ComputeServiceImpl.java", and "RMIClient.class". The "RPC" folder contains "RMIClient.java" and "RMIServer.java".
- Code Editor:** Displays the "RMIClient.java" file with Java code for an RMI client. The code imports "java.rmi.registry.LocateRegistry" and "java.util.Properties". It defines a class "RMIClient" with a main method that looks up a "ComputeService" from the registry and prints its matrix result.
- Terminal:** Shows the command-line output of running the client:

```
cd RMI/
java RMIClient.java
```

Matrix Result:
19 22
43 50

Prime Numbers: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
- Output:** Shows log entries in the terminal area, including session IDs and timestamps.

Error Handling

- RemoteException handled in all methods
- Network timeout handled via try–catch
- Registry lookup failure handled

Result

Thus, a **Java RMI-based distributed application** was successfully implemented and executed in a **cloud environment using Microsoft Azure**, allowing a local client to invoke methods on a remote server and receive correct outputs.

Conclusion

Java RMI provides a robust mechanism for building distributed applications by allowing remote method calls with object-oriented abstraction. Hosting the RMI server on Azure ensures scalability and remote accessibility.
