

Distributed Application using RPC and RMI (Cloud Execution)

1. Aim

To design and implement a **distributed application using Remote Procedure Call (RPC) and Remote Method Invocation (RMI)** where the server is hosted in a **cloud environment**, and the client accesses services remotely.

2. Objective

- To understand RPC and RMI concepts
- To implement client-server communication
- To host and execute the server in a cloud virtual machine
- To invoke remote procedures and methods successfully

3. System Requirements

Hardware

- Client machine (Windows/Linux)
- Cloud Virtual Machine (AWS EC2)

Software

- Python 3 (for RPC)
- Java JDK (for RMI)
- AWS EC2 (Linux)
- Internet connection

4. Technologies Used

- **RPC:** Python (XML-RPC)
- **RMI:** Java
- **Cloud Platform:** AWS EC2
- **Protocols:** TCP/IP

5. Architecture Overview

- Server application is deployed on a cloud VM
- Client runs on a local machine
- Client sends requests to server over the network
- Server processes the request and sends back results

6. RPC Implementation

Description

Remote Procedure Call allows a client to execute procedures on a remote server as if they were local functions.

RPC Server Responsibilities

- Define remote procedures
- Accept client requests
- Return results

RPC Client Responsibilities

- Connect to cloud server
- Invoke remote procedures
- Display results

Sample RPC Output

Addition result received from server: 15

7. RMI Implementation

Description

Remote Method Invocation allows Java objects to invoke methods on objects running in another JVM across the network.

Steps Involved

1. Define a remote interface
2. Implement the interface
3. Register object with RMI registry

4. Client invokes remote methods

Sample RMI Output

Connected to remote RMI server

Addition result: 30

8. Cloud Deployment

- Server programs (RPC and RMI) are executed on AWS EC2
- Required ports are opened in Security Groups
- Client accesses services using public IP of EC2 instance

9. Error Handling

- Network exceptions handled using try-catch blocks
- Server availability checked before invocation
- Proper error messages displayed on failure

10. Result

The client successfully invoked remote procedures and methods hosted on a cloud server using RPC and RMI. Correct results were received from the server.

11. Conclusion

This assignment demonstrates the working of distributed systems using RPC and RMI. Hosting the server in the cloud ensures scalability and real-world applicability.

12. Screenshots

i.) RPC

The screenshot displays the AWS CloudShell interface for an EC2 instance named `i-0689cea1853f90873 (rpc-rmi-server)`. The terminal shows the following sequence of commands and outputs:

```
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/jhsdb to provide /usr/bin/jhsdb (jhsdb) in auto mode
Setting up openjdk-11-jdk:amd64 (11.0.29+7-1ubuntu1-22.04) ...
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/jconsole to provide /usr/bin/jconsole (jconsole) in auto mode
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-18-46:~$ python3 --version
Python 3.10.12
ubuntu@ip-172-31-18-46:~$ java --version
openjdk 11.0.29 2025-10-21
OpenJDK Runtime Environment (build 11.0.29+7-post-Ubuntu-1ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.29+7-post-Ubuntu-1ubuntu122.04, mixed mode, sharing)
ubuntu@ip-172-31-18-46:~$ nano rpc_server.py
ubuntu@ip-172-31-18-46:~$ python3 rpc_server.py
RPC Server running on port 8000
```

The terminal then shows the execution of `python3 rpc_client.py` in a Windows PowerShell environment, which outputs:

```
PS D:\Assignments\SEM 6\DSC\ASSIGNMENT_1> python3 rpc_client.py
Addition: 15
Subtraction: 5
PS D:\Assignments\SEM 6\DSC\ASSIGNMENT_1>
```

Below this, the terminal shows the output of `python3 rpc_server.py` on the EC2 instance, displaying incoming HTTP requests:

```
ubuntu@ip-172-31-18-46:~$ python3 rpc_server.py
RPC Server running on EC2 at port 8000...
14.139.161.3 - - [29/Jan/2026 13:59:57] "POST / HTTP/1.1" 200 -
14.139.161.3 - - [29/Jan/2026 13:59:58] "POST / HTTP/1.1" 200 -
14.139.161.3 - - [29/Jan/2026 14:00:13] "POST / HTTP/1.1" 200 -
14.139.161.3 - - [29/Jan/2026 14:00:14] "POST / HTTP/1.1" 200 -
```

The final part of the screenshot shows the `rpc_server.py` file in the nano editor:

```
GNU nano 6.2 rpc_server.py
from xmlrpc.server import SimpleXMLRPCServer

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

server = SimpleXMLRPCServer(("0.0.0.0", 8000), allow_none=True)
print("RPC Server running on EC2 at port 8000...")

server.register_function(add, "add")
server.register_function(subtract, "subtract")

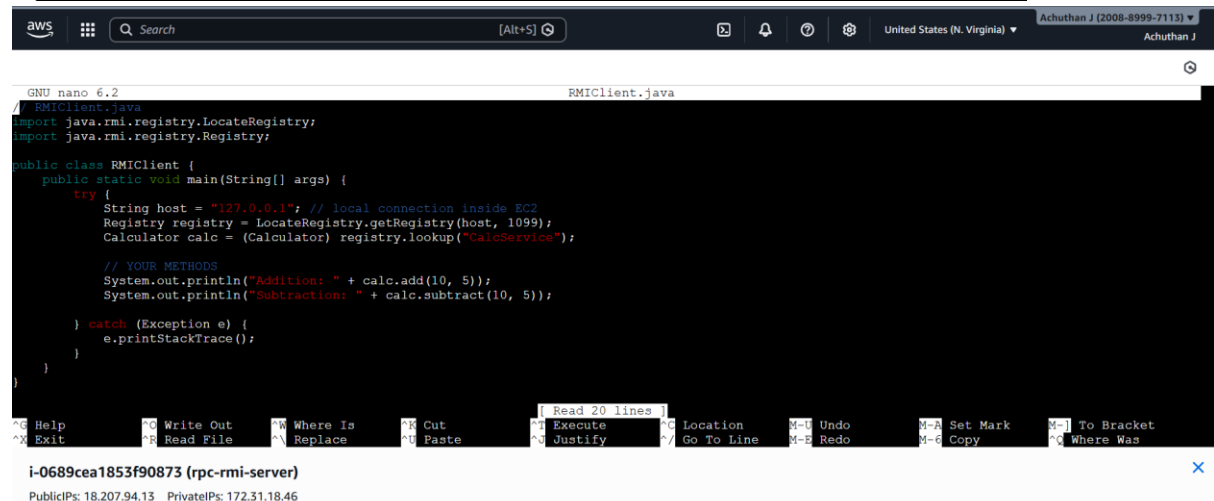
server.serve_forever()
```

The interface also includes a menu bar at the bottom with options like Help, Write Out, Where Is, Cut, Paste, Read 15 lines, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, and Where Was.

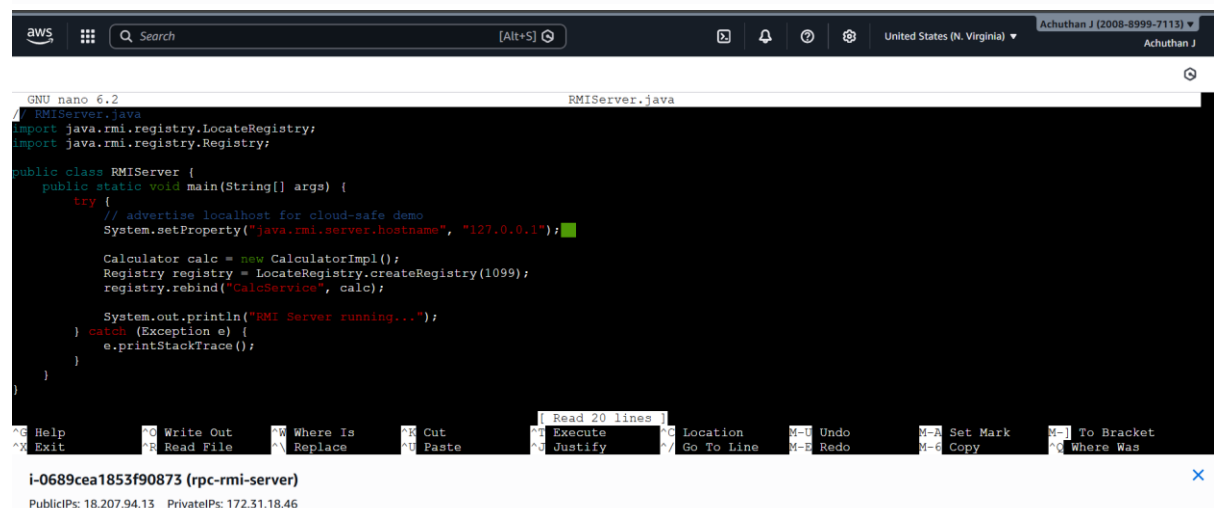
ii.) RMI

```
Last login: Thu Jan 29 15:18:32 2026 from 18.206.107.29
ubuntu@ip-172-31-18-46:~$ java RMIClient
Addition: 15
Subtraction: 5
```

```
ubuntu@ip-172-31-18-46:~$ javac *.java
ubuntu@ip-172-31-18-46:~$ java RMIServer
RMI Server running...
```



The screenshot shows an AWS console terminal window with the title "i-0689cea1853f90873 (rpc-rmi-server)". The terminal displays the source code for `RMIClient.java` in the `nano` editor. The code imports `java.rmi.registry.*` and `java.rmi.registry.Registry`. The `main` method sets the host to `"127.0.0.1"`, locates the registry at port 1099, looks up the `CalcService`, and prints the results of `add(10, 5)` and `subtract(10, 5)`. The terminal output at the top shows the results of running this client: `Addition: 15` and `Subtraction: 5`. The nano editor interface includes a menu bar with options like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Read 20 lines, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, and Where Was.



The screenshot shows the same AWS console terminal window, now displaying the source code for `RMIServer.java` in the `nano` editor. The code imports `java.rmi.registry.*` and `java.rmi.registry.Registry`. The `main` method sets the `java.rmi.server.hostname` property to `"127.0.0.1"`, creates a registry at port 1099, binds the `CalcService`, and prints `"RMI Server running..."`. The terminal output at the top shows the results of running this server: `RMI Server running...`. The nano editor interface is identical to the one in the previous screenshot.

```
GNU nano 6.2 CalculatorImpl.java
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {

    protected CalculatorImpl() throws RemoteException {
        super();
    }

    public int add(int a, int b) throws RemoteException {
        return a + b;
    }

    public int subtract(int a, int b) throws RemoteException {
        return a - b;
    }
}
```

i-0689cea1853f90873 (rpc-rmi-server)

PublicIPs: 18.207.94.13 PrivateIPs: 172.31.18.46

```
GNU nano 6.2 Calculator.java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {
    int add(int a, int b) throws RemoteException;
    int subtract(int a, int b) throws RemoteException;
}
```

i-0689cea1853f90873 (rpc-rmi-server)

PublicIPs: 18.207.94.13 PrivateIPs: 172.31.18.46