

Implementation of Remote Procedure Call (RPC) in a Cloud Environment Using Python

NAME: NAVINESHARAN S

ROLL NUMBER: 2023115015

1. Introduction

Remote Procedure Call (RPC) is a communication protocol that allows a program to execute a procedure on a remote server as if it were a local function call. RPC hides the complexities of network communication and enables distributed systems to interact seamlessly.

In this experiment, RPC is implemented using Python's XML-RPC library. The server hosting the remote procedures is deployed in a cloud environment, and the client invokes those procedures remotely over the internet.

2. Objective

- To implement RPC using Python
 - To host the RPC server in a cloud environment
 - To allow a client to remotely invoke procedures from the cloud server
 - To receive correct results from remote execution
-

3. Cloud Environment Setup

- Cloud Platform: **Microsoft Azure Virtual Machine**
- Server OS: **Ubuntu Linux**
- Client OS: **Windows**
- Programming Language: **Python 3**
- Library Used: `xmlrpc.server`, `xmlrpc.client`

The RPC server runs on an Azure VM with a public IP, while the client runs locally and communicates through the internet using TCP/IP.

Create a virtual machine


[Help me choose the right VM size for my workload](#)

Running final validation...

Basics

Subscription	Azure for Students
Resource group	NAVINDIST
Virtual machine name	NAVIN
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	NAVIN_key
Public inbound ports	SSH, HTTP, HTTPS

4. RPC Architecture

The client sends a request to invoke a procedure, the server executes it, and the result is sent back to the client.

5. Port Selection and Configuration

- **Port Used:** 8000
- **Purpose:**
 - Used by XML-RPC server to listen for client requests

Azure Network Security Group (NSG) is configured to allow inbound TCP traffic on port 8000 so that the client can access the RPC service hosted in the cloud.

The screenshot shows the Azure portal interface for managing network security groups. The left sidebar is collapsed. The main content area displays the 'Network security group HARAN-nsg' settings. The 'Inbound port rules' section is expanded, showing eight rules:

Priority	Name	Port	Protocol	Source	Destination	Action
300	SSH	22	TCP	Any	Any	Allow
320	HTTP	80	TCP	Any	Any	Allow
340	HTTPS	443	TCP	Any	Any	Allow
1000	RPC	8000	TCP	Any	Any	Allow
2000	RMI	1099	TCP	Any	Any	Allow
65000	AllowVnetinBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInbound	Any	Any	Any	Any	Deny

6. Server Implementation

The server performs the following:

- Creates a class containing remote procedures (add, subtract, multiply, divide)
- Registers the class instance with the XML-RPC server
- Binds the server to port 8000
- Waits continuously for client requests

The server executes procedures requested by the client and returns the result.

7. Client Implementation

The client performs the following:

- Connects to the server using the cloud VM's public IP and port 8000
- Calls remote procedures as if they are local functions
- Receives the computed result from the server

8. Cloud Connection Establishment

- The client uses the server's **public IP address**
- Communication occurs using HTTP protocol over TCP
- Remote procedures are executed on the cloud server

This confirms distributed execution between local client and cloud server.

9. Output

Sample Output (Client Side):

Addition: 15

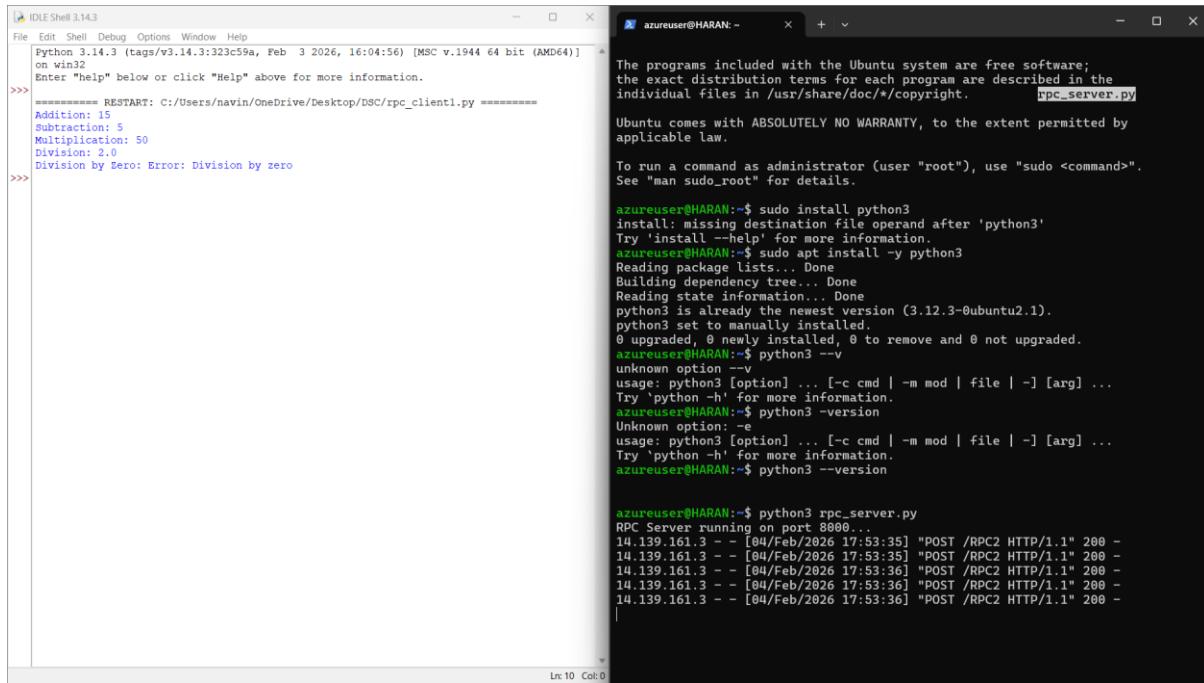
Subtraction: 5

Multiplication: 50

Division: 2.0

Division by Zero: Error: Division by zero

This proves that the procedures are executed on the cloud server and results are returned to the client.



The screenshot shows two terminal windows side-by-side. The left window is titled 'IDLE Shell 3.14.3' and contains Python code for a client. The right window is titled 'azureuser@HARAN:' and contains Python code for a server. Both windows show command-line interactions and output related to the RPC system.

```

IDLE Shell 3.14.3
File Edit Shell Debug Options Window Help
Python 3.14.3 (tags/v3.14.3:323c59a, Feb 3 2026, 16:04:56) [MSC v.1944 64 bit (AMD64)]
on win32
Enter "help" below or click "Help" above for more information.
>>> ===== RESTART: C:/Users/navin/OneDrive/Desktop/DSC/rpc_client1.py =====
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Division by Zero: Error: Division by zero
>>>

azureuser@HARAN: ~
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.
rpc_server.py
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

azureuser@HARAN: ~$ sudo apt install python3
install: missing destination file operand after 'python3'
Try 'install --help' for more information.
azureuser@HARAN: ~$ sudo apt install -y python3
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu2.1).
python3 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
azureuser@HARAN: ~$ python3 --version
unknown option --v
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try 'python -h' for more information.
azureuser@HARAN: ~$ python3 -version
Unknown option: -e
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try 'python -h' for more information.
azureuser@HARAN: ~$ python3 --version

azureuser@HARAN: ~$ python3 rpc_server.py
RPC Server running on port 8900...
14.139.161.3 - - [04/Feb/2026 17:53:35] "POST /RPC2 HTTP/1.1" 200 -
14.139.161.3 - - [04/Feb/2026 17:53:35] "POST /RPC2 HTTP/1.1" 200 -
14.139.161.3 - - [04/Feb/2026 17:53:36] "POST /RPC2 HTTP/1.1" 200 -
14.139.161.3 - - [04/Feb/2026 17:53:36] "POST /RPC2 HTTP/1.1" 200 -
14.139.161.3 - - [04/Feb/2026 17:53:36] "POST /RPC2 HTTP/1.1" 200 -
14.139.161.3 - - [04/Feb/2026 17:53:36] "POST /RPC2 HTTP/1.1" 200 -
|
```

10. Error Handling

- Division by zero handled using exception handling
- Network errors handled using Python exception mechanisms

- Server remains active even after multiple client requests
-

11. Conclusion

The RPC-based distributed application was successfully implemented using Python XML-RPC and deployed in a cloud environment. The client was able to remotely invoke procedures hosted on the cloud server and receive correct results. This demonstrates how RPC enables transparent communication between distributed systems over the internet.