

# Table of Content

<b>Language used for the Implementation</b>	<b>3</b>
<b>System Overview</b>	<b>3</b>
Protocol	3
<b>Step-by-Step Work Flow</b>	<b>4</b>
1) Setting up the AWS infrastructure	4
Create key pairs	4
Create a Security Group	4
Launch EC2 instances	5
Record Public & Private IPs for all instances (used in scripts)	9
SSH & Install the necessary tools	10
Connectivity check	10
2) Writing the CLIENT script	11
Protocol Overview	11
Expected responses from SERVER1	11
Code Overview	11
Main execution flow	11
Source Code Listing	12
3) Writing the SERVER1 script	13
How the conversation works	13
Decision making	13
Code Walkthrough	13
Safety and Reliability	14
How to run it	14
Source Code Listing	14
4) Writing the SERVER2 script	17
Code Walkthrough	17
How to run it	17
Source Code Listing	17
5) Testing	19
Test Setup	19
Client Machine Setup	19
Server1 Setup	20
Server2 Setup	22
Test Case1	26
End-to-End Verification of Client-Server File Sync	26

● Precondition	26
● Test Steps	26
● Expected Result	27
Test Case2	28
File Read Operation with Missing File on One Server	28
● Precondition	28
● Test Steps	28
● Expected Result	29
Test Case3	32
File Read Operation When File Missing on Both Servers	32
● Precondition	32
● Test Steps	32
● Expected Result	33
Test Case4	34
File Read Operation When Servers Have Identical File	34
● Precondition	34
● Test Steps	34
● Expected Result	34

# Distributed Computing Assignment

## Language used for the Implementation

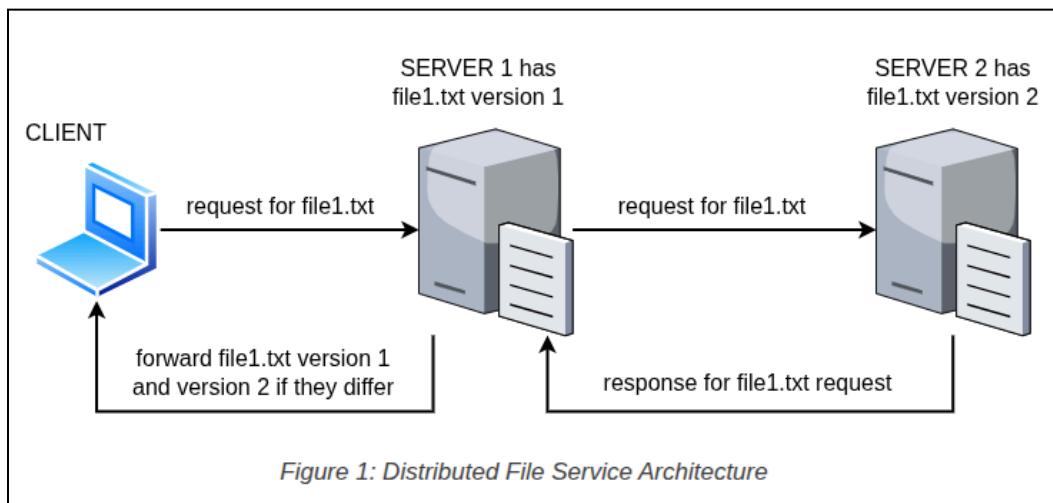
Python 3.

## System Overview

We implemented the required client-server distributed file service with one client, SERVER1 (coordinator), and SERVER2 (peer file server). Both servers maintain a replica directory `~/file_storage`. Due to possible update delays, SERVER1 always cross-checks with SERVER2 before replying to the client.

## Protocol

- Client → SERVER1 (TCP:5001): GET <pathname>\n
- SERVER1 → SERVER2 (TCP:5002): GET <pathname>\n
- SERVER2 → SERVER1:
  - FOUND <length>\n followed by <length> bytes, or
  - NOTFOUND\n.
- SERVER1 → Client:
  - OK ONE <length>\n followed by one file copy, or
  - OK BOTH <length1> <length2>\n followed by two file copies (SERVER1 then SERVER2), or
  - ERROR NOTFOUND\n when neither server has the file, or invalid request.

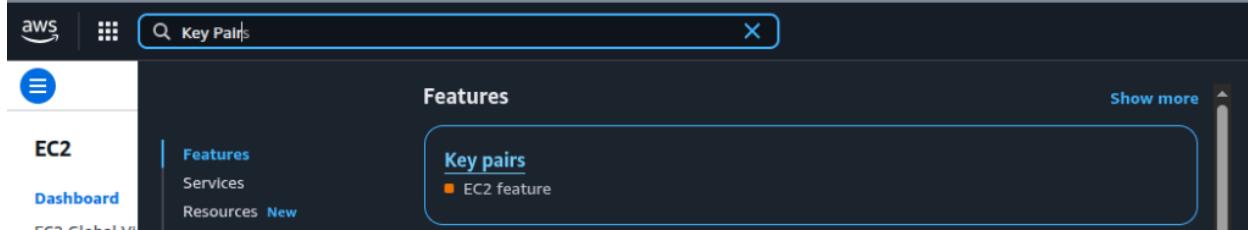


# Step-by-Step Work Flow

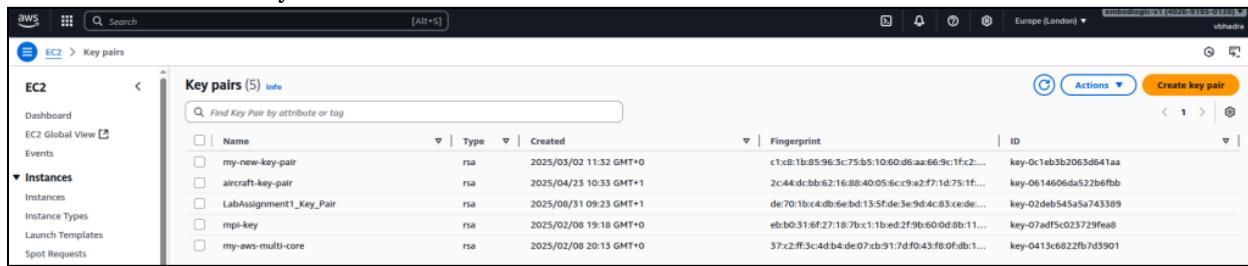
## 1) Setting up the AWS infrastructure

### Create key pairs

- Go to the AWS dashboard and search for key pairs.



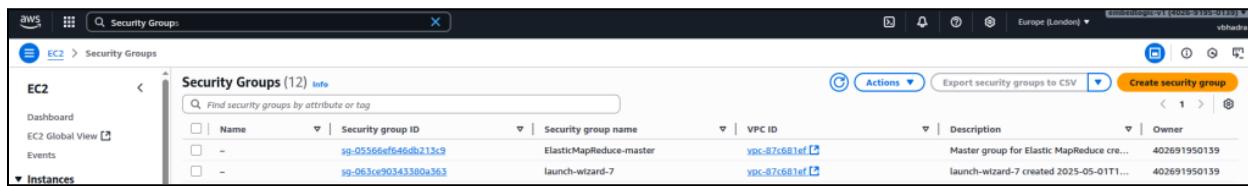
- Click on Create Key Pair.



- Name the key pair (e.g., MyFileSystemKeyPair). Leave the remaining fields at their default values and click Create key pair.
- Keep the .pem safe as this will be used to SSH into the AWS instances.

### Create a Security Group

- Search for security groups. On the Security Group page, click on the Create security group button.



- Name the new security group (e.g., MyFileServerSecurityGroup).
- On the inbound rules, open TCP port **22** for SSH, TCP port **5001** (SERVER1), TCP port **5002** (SERVER2).
- For the demonstration, keep **Source = Anywhere**.

- On the outbound rules, allow all traffic.

The screenshot shows the AWS Security Group configuration page. It includes three main sections: **Inbound rules**, **Outbound rules**, and **Tags - optional**. The **Inbound rules** section lists three rules: SSH (TCP 22), Custom TCP (TCP 5001), and Custom TCP (TCP 5002), all allowing 0.0.0.0/0. The **Outbound rules** section shows a single rule for 'All traffic' to 'Custom' destination, also allowing 0.0.0.0/0. The **Tags - optional** section indicates no tags are associated with the resource. At the bottom right are 'Cancel' and 'Create security group' buttons.

- Create a security group. Once the group is created, it should look somewhat like the example shown below:

The screenshot shows the details page for the security group 'sg-0dd4a58da8a9aa8ad - MyFailedServerSecurityGroup'. It displays the following information:

- Details** section: Security group name (MyFailedServerSecurityGroup), Security group ID (sg-0dd4a58da8a9aa8ad), Description (Security group for distributed file server demo), and VPC ID (vpc-87c681ef).
- Inbound rules** section: 3 Permission entries. The table shows three rules:
 

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0673024e4bd815925	IPv4	Custom TCP	TCP	5001	0.0.0.0/0	-
-	sgr-002ad83a9555e9921	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sgr-0ed670c477fa1f498	IPv4	Custom TCP	TCP	5002	0.0.0.0/0	-

- This security group will be shared by Server 1, Server 2, and the Client.

## Launch EC2 instances

- We will need at least three EC2 instances—one for the client and one for each of the two servers.
- Let's create three identical EC2 instances with the following details:

```

Name(s): CLIENT-Node, SERVER1, SERVER2
Security Group: MyFailedServerSecurityGroup
Key Pair: MyFileSystemKeyPair
AWS ASI Type: Ubuntu 22.04 t3.micro or t2.micro

```

- Go to EC2 Dashboard -> Instances -> Launch Instance.

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed, and the main content area is titled 'Instances Info'. At the top, there are filters for 'Instance state' (set to 'running') and 'Clear filters'. Below the filters, there are columns for 'Name', 'Instance ID', 'Instance state', 'Instance type', 'Status check', 'Alarm status', 'Availability Zone', and 'Public IP'. A search bar at the top says 'Find Instance by attribute or tag (case-sensitive)'. A message at the bottom of the table area says 'No matching instances found'.

- Choose Ubuntu as the OS image.
- Select Ubuntu 24.04 as the AMI. Select the instance type (t3.micro).

The screenshot shows the 'Launch instance' wizard. The first step, 'Name and tags', has a 'Name' field containing 'CLIENT-Node' and a 'Add additional tags' link. The second step, 'Application and OS Images (Amazon Machine Image)', shows a search bar and tabs for 'Recents', 'My AMIs', and 'Quick Start'. Under 'Quick Start', there are icons for Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. A search bar at the top says 'Search our full catalog including 1000s of application and OS Images'. To the right, there is a 'Browse more AMIs' link and a note about including AMIs from AWS, Marketplace, and the Community. At the bottom, there is a section for 'Amazon Machine Image (AMI)' with details: 'Ubuntu Server 24.04 LTS (HVM), SSD Volume Type', 'ami-046c2381f11878233 (64-bit (x86)) / ami-0122205e4fe2524d5 (64-bit (Arm))', 'Virtualization: hvm', 'ENA enabled: true', 'Root device type: ebs', and a 'Free tier eligible' link.

- Select the previously created key pair.

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t3.micro	Free tier eligible			
Family: t3	2 vCPU	1 GiB Memory	Current generation: true	On-Demand Linux base pricing: 0.0118 USD per Hour
On-Demand SUSE base pricing: 0.0118 USD per Hour	On-Demand Ubuntu Pro base pricing: 0.0153 USD per Hour			
On-Demand RHEL base pricing: 0.0406 USD per Hour	On-Demand Windows base pricing: 0.021 USD per Hour			

**Additional costs apply for AMIs with pre-installed software**

All generations

[Compare instance types](#)

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

<a href="#">Select</a>	<a href="#">Create new key pair</a>
<input type="text" value="Q  "/>	<a href="#">Edit</a>
Proceed without a key pair (Not recommended)	Default value
my-new-key-pair	Type: rsa
aircraft-key-pair	Type: rsa
LabAssignment1_Key_Pair	Type: rsa
mpi-key	Type: rsa
MyFileSystemKeyPair	Type: rsa
my-aws-multi-core	Type: rsa

your instance.

- Attach the created security group under Network settings.

▼ Network settings [Info](#)

Network [Info](#)  
vpc-87c681ef

Subnet [Info](#)  
No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)  
Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)  
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group  Select existing security group

Common security groups [Info](#)

<a href="#">Select security groups</a>	<a href="#">Compare security group rules</a>
<input type="text" value="Q  "/>	<a href="#">Edit</a>
launch-wizard-7 VPC: vpc-87c681ef	sg-003cc50545508883
launch-wizard-6 VPC: vpc-87c681ef	sg-06bb2b4f1416f3f5d
launch-wizard-4 VPC: vpc-87c681ef	sg-08fcfc085b6967063
launch-wizard-1 VPC: vpc-87c681ef	sg-0a9a725b8251db04a
MyFileServerSecurityGroup VPC: vpc-87c681ef	sg-0dd4a58da8a9aa8ad
default VPC: vpc-87c681ef	sg-aa35fec8

Advanced

The selected AMI contains instance store volumes, however the instance does not allow any instance store volumes. None of the instance store volumes from the AMI will be accessible from the instance

- Since we need three EC2 instances with the same base configurations, under Summary on the right hand side set the number of instances to 3.

## ▼ Summary

**Number of instances** [Info](#)

3

When launching more than 1 instance, [consider EC2 Auto Scaling](#)

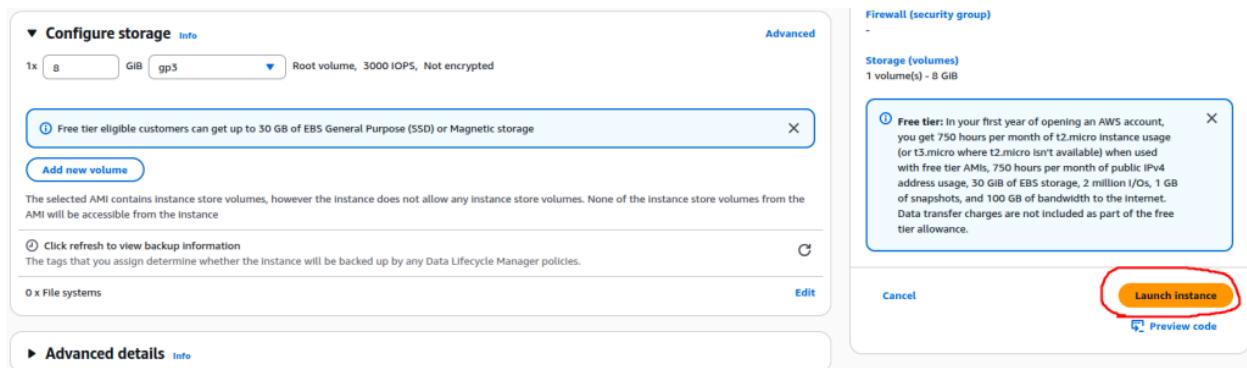
---

**Software Image (AMI)**  
 Canonical, Ubuntu, 24.04, amd6...[read more](#)  
 ami-02d26659fd82cf299

---

**Virtual server type (instance type)**  
 t3.micro

- Finally Click the Launch instance.



**Configure storage** [Info](#)

Advanced

1x 8 GiB gp3 Root volume, 3000 IOPS, Not encrypted

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

Add new volume

The selected AMI contains instance store volumes, however the instance does not allow any instance store volumes. None of the instance store volumes from the AMI will be accessible from the instance

Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

File systems

Advanced details [Info](#)

Firewall (security group)

Storage (volumes)

1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included as part of the free tier allowance.

Cancel Launch instance [Preview code](#)

- This will create three EC2 instances with the selected configurations.
- After the instances are created, go to the EC2 page. You should be able to see all three instances running.

## Resources

EC2 Global View   

You are using the following Amazon EC2 resources in the Europe (London) Region:

Instances (running)	3	Auto Scaling Groups	0	Capacity Reservations	0
Dedicated Hosts	0	Elastic IPs	0	Instances	3
Key pairs	6	Load balancers	0	Placement groups	0
Security groups	13	Snapshots	0	Volumes	3

- Name all the instances for easy identification.

Instances (3) 										
Last updated less than a minute ago  Connect  Actions 										
Find Instance by attribute or tag (case-sensitive)  All states 										
Instance state = running  Clear filters										
 Name  Instance ID  Instance state  Instance type  Status check  Alarm status  Availability Zone  Public IPv4 DNS  Public IPv4  Elastic										
 SERVER2	I-04426b791a2547fb7	  Running	t3.micro	 3/3 checks passed	<a href="#">View alarms</a> 	eu-west-2a	ec2-18-130-207-66.eu...	18.130.207.66	–	
 CLIENT-Node	I-0077d39fa8de96a8c	  Running	t3.micro	 3/3 checks passed	<a href="#">View alarms</a> 	eu-west-2a	ec2-18-130-101-24.eu...	18.130.101.24	–	
 SERVER1	I-01f30a606983a0675	  Running	t3.micro	 3/3 checks passed	<a href="#">View alarms</a> 	eu-west-2a	ec2-18-171-162-167.eu...	18.171.162.167	–	

## Record Public & Private IPs for all instances (used in scripts)

- These addresses will be required later when we write the scripts to establish communication between the client and the servers.
- In this case, the following are the Private and Public IPs of each instance:

CLINET-Node:

Public IPv4 address: 18.130.101.24

Private IPv4 addresses: 172.31.24.111

SERVER1:

Public IPv4 address: 18.171.162.167

Private IPv4 addresses: 172.31.16.124

SERVER2:

Public IPv4 address: 18.130.207.66

Private IPv4 addresses: 172.31.26.202

## SSH & Install the necessary tools

- First, we need to change the file permissions of the key pair file (downloaded in one of the previous steps) by running the following command.

```
chmod 400 ~/Downloads/MyFileSystemKeyPair.pem
```

- The sample SSH command will be like:

```
ssh -i ~/Downloads/MyFileSystemKeyPair.pem ubuntu@<EC2 Public IPv4 Addr>
```

- Let's SSH into the Client-Node and install the necessary packages:

```
~$ ssh -i ~/Downloads/MyFileSystemKeyPair.pem ubuntu@18.130.101.24
ubuntu@ip-172-31-24-111:~$ sudo apt update
ubuntu@ip-172-31-24-111:~$ sudo apt install netcat-openbsd -y
ubuntu@ip-172-31-24-111:~$ which nc
/usr/bin/nc
```

- Similarly, SSH into the Server 1 instance and run the commands

```
~$ ssh -i ~/Downloads/MyFileSystemKeyPair.pem ubuntu@18.171.162.167
~$ sudo apt update
~$ sudo apt install netcat-openbsd -y
~$ which nc
/usr/bin/nc
```

- Follow the exact set of steps for Server 2 as well.

```
~$ ssh -i ~/Downloads/MyFileSystemKeyPair.pem ubuntu@18.130.207.66
~$ sudo apt update
~$ sudo apt install netcat-openbsd -y
~$ which nc
/usr/bin/nc
```

- Now, all three instances are configured and ready for a quick connectivity check.

## Connectivity check

- Before we move into the actual File Service server implementation, let's check the connections between the instances to make sure everything is set up correctly.
- Run the following on SERVER1:

```
ubuntu@ip-172-31-26-202:~$ nc -l -p 5002
```

- Then run the following on the CLIENT:

```
ubuntu@ip-172-31-24-111:~$ echo "hello from CLIENT to SERVER1" | nc
172.31.16.124 5002
```

- You should be able to see the message from the client displayed on SERVER1:

```
ubuntu@ip-172-31-16-124:~$ nc -l -p 5002
```

```
hello from CLIENT to SERVER1
```

- This confirms that the connection between the CLIENT-Node and SERVER1 is working correctly. We could also have used port 5001, since both ports 5001 and 5002 are open for TCP connections.
- You can repeat the same steps to test the connection between SERVER1 and SERVER2.

## 2) Writing the CLIENT script

The `client.py` program is the interface for users to request files from our distributed file service. Its job is simple: connect to SERVER1 (the coordinator), request a file, and handle the response. It doesn't interact with SERVER2 directly – it trusts SERVER1 to coordinate and return the correct data.

### Protocol Overview

When the client connects to SERVER1 on port 5001, it sends a request in the form:

```
GET filename
```

SERVER1 then processes the request – including checking with SERVER2 – and replies with a structured response, followed by file data if available. The client parses this response and prints the result.

### Expected responses from SERVER1

- OK ONE <length> – a single version of the file follows.
- OK BOTH <length1> <length2> – two versions follow (from SERVER1 and SERVER2).
- ERROR NOTFOUND – the file doesn't exist on either server.

### Code Overview

The script uses two helper functions:

- `recv_line(conn)`: Reads a line from the socket (used for headers).
- `recv_exact(conn, n)`: Reads exactly n bytes (used for file contents).

### Main execution flow

1. Parses the filename from command-line args (defaults to "file1.txt").
2. Connects to SERVER1 over TCP.
3. Send a GET request.
4. Reads and interprets the response header.

5. Receives and displays file data based on the response type.

## Source Code Listing

```
#!/usr/bin/env python3
# client.py
import socket, sys

SERVER1_IP = "172.31.16.124" # <-- PUT SERVER1 PRIVATE IP HERE
SERVER1_PORT = 5001
filename = sys.argv[1] if len(sys.argv) > 1 else "file1.txt"

def recv_line(conn):
    buf = bytearray()
    while True:
        b = conn.recv(1)
        if not b:
            break
        buf += b
        if buf.endswith(b"\n"):
            break
    return bytes(buf).decode(errors="replace").rstrip("\n")

def recv_exact(conn, n):
    buf = bytearray()
    while len(buf) < n:
        chunk = conn.recv(n - len(buf))
        if not chunk:
            raise ConnectionError("socket closed while reading body")
        buf += chunk
    return bytes(buf)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((SERVER1_IP, SERVER1_PORT))
    s.sendall(f"GET {filename}\n".encode())

    header = recv_line(s) # e.g., OK ONE <len> / OK BOTH <l1> <l2> / ERROR
    NOTFOUND
    parts = header.split()
    if parts[:2] == ["OK", "ONE"]:
        length = int(parts[2]); data = recv_exact(s, length)
        print("[CLIENT] Received ONE copy:")
        print(data.decode(errors="ignore"))
```

```

# Optionally save:
# open("out_one.txt", "wb").write(data)
elif parts[:2] == ["OK", "BOTH"]:
    l1, l2 = int(parts[2]), int(parts[3])
    d1 = recv_exact(s, l1); d2 = recv_exact(s, l2)
    print("[CLIENT] Received BOTH copies:")
    print("\n--- SERVER1 version ---\n", d1.decode(errors="ignore"))
    print("\n--- SERVER2 version ---\n", d2.decode(errors="ignore"))
    # Optionally save:
    # open("out_server1.txt", "wb").write(d1)
    # open("out_server2.txt", "wb").write(d2)
else:
    print("[CLIENT]", header)

```

### 3) Writing the SERVER1 script

The `server1.py` program is the coordinator of our small distributed file service. Its job is simple to explain: it waits for clients to connect on port 5001, checks if the requested file exists locally, always asks SERVER2 for the same file, and then decides what to send back. Think of it as the “middle manager” – it doesn’t just rely on its own copy but always double-checks with SERVER2 before answering the client.

#### How the conversation works

When a client connects, it sends a request like `GET file1.txt`. SERVER1 then talks to SERVER2 in exactly the same way. If SERVER2 replies that the file exists, it also sends the file contents back; if not, it says `NOTFOUND`. Based on this exchange, SERVER1 then prepares its response for the client.

#### Decision making

- If both SERVER1 and SERVER2 have the file and their contents are *identical*, SERVER1 keeps things efficient and returns just one copy.
- If both servers have the file but the contents are *different*, SERVER1 sends both versions so the client can see the mismatch.
- If only one of the two servers has the file, SERVER1 forwards that copy.
- If neither server has it, SERVER1 sends back a simple “not found” error.

#### Code Walkthrough

The script uses helper functions to keep things tidy. `read_local()` looks up the file under `~/file_storage` on SERVER1. `get_from_server2()` connects out to SERVER2, asks for the same file, and collects the result. Once both responses are in, SERVER1 compares them and

decides which of the `send_one()`, `send_both()`, or `send_error()` functions to use when replying to the client.

## Safety and Reliability

The script is careful about what paths it accepts (so clients can't wander outside the storage folder), it checks that full files are received before making decisions, and it catches errors so that one misbehaving server doesn't crash the whole service. Logging messages such as "Versions differ → sent BOTH" make it easy to see what happened during each request.

## How to run it

Before starting, the `SERVER2_IP` variable needs to be updated to the private IP of SERVER2. Once SERVER2 is already running, you can start this script on SERVER1 with the following command on the command prompt:

```
ubuntu@ip-172-31-16-124:~$ ./server1.py
[SERVER1] Listening on 0.0.0.0:5001
```

It will sit listening on port 5001, handle each incoming client request, and coordinate with SERVER2 in the background. In short, SERVER1 is the smart middle layer: it listens, checks both sides, makes a fair decision, and keeps the client informed.

Screenshot of running `server1.py` on EC2 instance:

```
Last login: Sun Sep  7 14:41:05 2025 from 81.129.72.226
ubuntu@ip-172-31-16-124:~$ ./server1.py
[SERVER1] Listening on 0.0.0.0:5001
```

## Source Code Listing

```
#!/usr/bin/env python3
import socket, os

HOST = "0.0.0.0"
PORT = 5001
SERVER2_IP = "172.31.xx.yy"    # <-- replace with SERVER2 private IP
SERVER2_PORT = 5002
ROOT = os.path.expanduser("~/file_storage")

def recv_line(conn):
    buf = bytearray()
    while True:
```

```

        b = conn.recv(1)
        if not b: break
        buf += b
        if buf.endswith(b"\n"): break
    return bytes(buf).decode(errors="replace").rstrip("\n")

def recv_exact(conn, n):
    buf = bytearray()
    while len(buf) < n:
        chunk = conn.recv(n - len(buf))
        if not chunk: raise ConnectionError("socket closed early")
        buf += chunk
    return bytes(buf)

def get_from_server2(filename: str) -> bytes | None:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s2:
            s2.connect((SERVER2_IP, SERVER2_PORT))
            s2.sendall(f"GET {filename}\n".encode())
            header = recv_line(s2)
            if header == "NOTFOUND": return None
            if header.startswith("FOUND "):
                length = int(header.split()[1])
                return recv_exact(s2, length)
    except Exception as e:
        print("[SERVER1] Error contacting SERVER2:", e)
    return None

def read_local(filename: str) -> bytes | None:
    rel = filename.lstrip("/")
    path = os.path.normpath(os.path.join(ROOT, rel))
    if not path.startswith(ROOT): return None
    if os.path.isfile(path):
        with open(path, "rb") as f: return f.read()
    return None

def send_one(conn, data: bytes):
    conn.sendall(f"OK ONE {len(data)}\n".encode() + data)

def send_both(conn, d1: bytes, d2: bytes):
    conn.sendall(f"OK BOTH {len(d1)} {len(d2)}\n".encode() + d1 + d2)

def send_error(conn, msg="ERROR NOTFOUND"):

```

```

conn.sendall((msg + "\n").encode())

def handle_client(conn, addr):
    try:
        line = recv_line(conn)
        if not line.startswith("GET "):
            send_error(conn, "ERROR BADREQUEST"); return
        filename = line[4:].strip()
        print(f"[SERVER1] CLIENT {addr} requested {filename}")

        local = read_local(filename)
        remote = get_from_server2(filename)

        if local and remote:
            if local == remote:
                send_one(conn, local)
                print("[SERVER1] Both matched → sent ONE")
            else:
                send_both(conn, local, remote)
                print("[SERVER1] Versions differ → sent BOTH")
        elif local:
            send_one(conn, local)
            print("[SERVER1] Only SERVER1 had it → sent ONE")
        elif remote:
            send_one(conn, remote)
            print("[SERVER1] Only SERVER2 had it → sent ONE")
        else:
            send_error(conn)
            print("[SERVER1] Not found on either")
    except Exception as e:
        print("[SERVER1] Error:", e)
        try: send_error(conn, "ERROR INTERNAL")
        except: pass

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s1:
        s1.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s1.bind((HOST, PORT))
        s1.listen()
        print(f"[SERVER1] Listening on {HOST}:{PORT}")
        while True:
            conn, addr = s1.accept()
            with conn: handle_client(conn, addr)

```

```
if __name__ == "__main__":
    main()
```

## 4) Writing the SERVER2 script

The `server2.py` program is a peer server. Its sole purpose is to serve file content to Server 1 when requested. It doesn't communicate with the client directly, but rather acts as a data source within the distributed network. It listens for requests from server1 on port 5002.

### Code Walkthrough

The script uses helper functions for the smooth and error free execution. It uses a `recv_line` function, which is used to receive the request from server1 and send the request as a string to the `handle` function which is responsible for reading the file path from the `recv_line` function and checks for the filename if it exists. `Send_found` responsible for sending the file to server1 if found and `send_notfound` to let server1 know no such file exists in server2.

### How to run it

SERVER2 is defined to be running on port 5002, you can start this script on SERVER2 with the following command on the command prompt:

```
ubuntu@ip-172-31-31-119:~$ ./server2.py
[SERVER2] Listening on 0.0.0.0:5002
```

It will sit listening on port 5002, handle each incoming server1 request  
Screenshot of running SERVER2.py on EC2 instance:

```
[ec2-user@ip-172-31-31-119 ~]$ ./server2.py
[SERVER2] Listening on 0.0.0.0:5002
```

### Source Code Listing

```
#!/usr/bin/env python3
import socket, os
HOST = "0.0.0.0"
PORT = 5002
ROOT = os.path.expanduser("~/file_storage")

def recv_line(conn):
```

```

buf = bytearray()
while True:
    b = conn.recv(1)
    if not b: break
    buf += b
    if buf.endswith(b"\n"): break
return bytes(buf).decode(errors="replace").rstrip("\n")

def send_found(conn, data: bytes):
    header = f"FOUND {len(data)}\n".encode()
    conn.sendall(header + data)

def send_notfound(conn):
    conn.sendall(b"NOTFOUND\n")

def handle(conn, addr):
    try:
        line = recv_line(conn)
        if not line.startswith("GET "):
            send_notfound(conn); return
        rel = line[4:].strip().lstrip("/")
        path = os.path.normpath(os.path.join(ROOT, rel))

        if not path.startswith(ROOT):
            send_notfound(conn); return
        if os.path.isfile(path):
            with open(path, "rb") as f: data = f.read()
            send_found(conn, data)
            print(f"[SERVER2] Sent {rel} ({len(data)} bytes) to {addr}")
        else:
            send_notfound(conn)
            print(f"[SERVER2] {rel} not found for {addr}")
        elif local:
            send_one(conn, local)
            print("[SERVER1] Only SERVER1 had it → sent ONE")
    except Exception as e:
        print("[SERVER2] Error:", e)

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s1.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s1.bind((HOST, PORT))
        s1.listen()

```

```

print(f"[SERVER1] Listening on {HOST}:{PORT}")
while True:
    conn, addr = s.accept()
    with conn: handle_client(conn, addr)

if __name__ == "__main__":
    main()

```

## 5) Testing

### Test Setup

- Different files on SERVER1 and SERVER2 (initial state) → client receives BOTH copies (shows each version).
- Identical files on both servers (edit SERVER2's file to match SERVER1) → client receives ONE copy.
- Nested path provided by client (e.g., docs/report.txt with different contents) → client receives BOTH copies.

### Client Machine Setup

- Configured client machine with the PKF file.
- Created a file client.py in /home/ubuntu on the client node.

```
ubuntu@ip-172-31-34-69: ~
  login as: ubuntu
  Authenticating with public key "DCFileSystemKeyValuePair070925"
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

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

System information as of Sat Sep 13 06:59:28 UTC 2025

System load: 0.09          Temperature:      -273.1 C
Usage of /: 29.7% of 6.71GB Processes:          113
Memory usage: 22%          Users logged in:    0
Swap usage:  0%            IPv4 address for ens5: 172.31.34.69

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

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

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Sun Sep  7 14:30:04 2025 from 122.172.80.6
ubuntu@ip-172-31-34-69:~$
```

```
ubuntu@ip-172-31-34-69: ~
ubuntu@ip-172-31-34-69:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-34-69:~$ ls -ltrh
total 4.0K
-rwxrwxrwx 1 ubuntu ubuntu 1.7K Sep  7 14:54 client.py
ubuntu@ip-172-31-34-69:~$
```

## Server1 Setup

1. Configured Server1 using the PKF file.
2. Created a file server1.py in /home/ubuntu.
3. Created a directory file\_storage in /home/ubuntu.
4. Created a file file1.txt inside /home/ubuntu/file\_storage.

```
ubuntu@ip-172-31-34-27: ~
login as: ubuntu
Authenticating with public key "DCFileSystemKeyValuePair070925"
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

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

System information as of Sat Sep 13 07:16:15 UTC 2025

System load:  0.0          Temperature:      -273.1  C
Usage of /:   28.1% of 6.71GB  Processes:        111
Memory usage: 22%          Users logged in:   0
Swap usage:   0%           IPv4 address for ens5: 172.31.34.27

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Sun Sep  7 14:35:03 2025 from 122.172.80.6
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-34-27:~$
```

```
ubuntu@ip-172-31-34-27: ~
ubuntu@ip-172-31-34-27:~$ ls
file_storage  server1.py
ubuntu@ip-172-31-34-27:~$
```

```
ubuntu@ip-172-31-34-27: ~/file_storage
ubuntu@ip-172-31-34-27:~$ ls
file_storage  server1.py
ubuntu@ip-172-31-34-27:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-34-27:~$ ls
file_storage  server1.py
ubuntu@ip-172-31-34-27:~$ ls -ltrh
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 3.2K Sep  7 14:43 server1.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 07:35 file_storage
ubuntu@ip-172-31-34-27:~$ cd file_storage/
ubuntu@ip-172-31-34-27:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$
```

```
ubuntu@ip-172-31-34-27: ~/file_storage
Content: "Report version from SERVER1"
[~]
[~]
[~]
```

## Server2 Setup

1. Configured Server2 using the PKF file.
2. Created a file `server2.py` in `/home/ubuntu`.
3. Created a directory `file_storage` in `/home/ubuntu`.
4. Created a file `file1.txt` inside `/home/ubuntu/file_storage`.

```
ubuntu@ip-172-31-45-10: ~
└─ login as: ubuntu
└─ Authenticating with public key "DCFileSystemKeyPair070925"
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

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

System information as of Sat Sep 13 07:16:57 UTC 2025

System load: 0.0          Temperature:          -273.1 C
Usage of /:   34.4% of 6.71GB  Processes:           108
Memory usage: 31%          Users logged in:      0
Swap usage:   0%          IPv4 address for ens5: 172.31.45.10

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

2 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sun Sep  7 14:33:13 2025 from 122.172.80.6
ubuntu@ip-172-31-45-10:~$ █
```

```
ubuntu@ip-172-31-45-10: ~
ubuntu@ip-172-31-45-10:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-45-10:~$ ls
file_storage  server2.py
ubuntu@ip-172-31-45-10:~$ █
```

```
ubuntu@ip-172-31-45-10:~/file_storage
ubuntu@ip-172-31-45-10:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-45-10:~$ ls
file_storage  server2.py
ubuntu@ip-172-31-45-10:~$ cd file_storage/
ubuntu@ip-172-31-45-10:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$
```

```
ubuntu@ip-172-31-45-10:~/file_storage
Content: "Report version from SERVER2"
[REDACTED]
~
```

Update the SERVER2\_IP variable in the server1.py script with the private IP address of Server2.

```

ubuntu@ip-172-31-34-27: ~
#!/usr/bin/env python3
import socket, os

HOST = "0.0.0.0"
PORT = 5001
SERVER2_IP = "172.31.45.10"    # <-- replace with SERVER2 private IP
SERVER2_PORT = 5002
ROOT = os.path.expanduser("~/file_storage")

def recv_line(conn):
    buf = bytearray()
    while True:
        b = conn.recv(1)
        if not b: break
        buf += b
        if buf.endswith(b"\n"): break
    return bytes(buf).decode(errors="replace").rstrip("\n")

def recv_exact(conn, n):
    buf = bytearray()
    while len(buf) < n:
        chunk = conn.recv(n - len(buf))
        if not chunk: raise ConnectionError("socket closed early")
        buf += chunk
    return bytes(buf)

def get_from_server2(filename: str) -> bytes | None:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s2:
            s2.connect((SERVER2_IP, SERVER2_PORT))
            s2.sendall(f"GET {filename}\n".encode())
            header = recv_line(s2)
            if header == "NOTFOUND": return None
            if header.startswith("FOUND "):
                length = int(header.split()[1])
                return recv_exact(s2, length)
    except Exception as e:
        print("[SERVER1] Error contacting SERVER2:", e)
    return None

def read_local(filename: str) -> bytes | None:
    rel = filename.lstrip("/")
    path = os.path.normpath(os.path.join(ROOT, rel))

```

Update the SERVER1\_IP variable in the client.py script with the private IP address of Server1.

```

ubuntu@ip-172-31-34-69: ~
#!/usr/bin/env python3
# client.py
import socket, sys

SERVER1_IP = "172.31.34.27" # <-- PUT SERVER1 PRIVATE IP HERE
SERVER1_PORT = 5001
filename = sys.argv[1] if len(sys.argv) > 1 else "file1.txt"

def recv_line(conn):
    buf = bytearray()
    while True:
        b = conn.recv(1)
        if not b:
            break
        buf += b
        if buf.endswith(b"\n"):
            break
    return bytes(buf).decode(errors="replace").rstrip("\n")

def recv_exact(conn, n):
    buf = bytearray()
    while len(buf) < n:
        chunk = conn.recv(n - len(buf))
        if not chunk:
            raise ConnectionError("socket closed while reading body")
        buf += chunk
    return bytes(buf)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((SERVER1_IP, SERVER1_PORT))
    s.sendall(f"GET {filename}\n".encode())

    header = recv_line(s) # e.g., OK ONE <len> / OK BOTH <l1> <l2> / ERROR NOTFOUND
    parts = header.split()
    if parts[:2] == ["OK", "ONE"]:
        length = int(parts[2]); data = recv_exact(s, length)
        print("[CLIENT] Received ONE copy:")
        print(data.decode(errors="ignore"))
        # Optionally save:
        # open("out_one.txt", "wb").write(data)
    elif parts[:2] == ["OK", "BOTH"]:
        l1, l2 = int(parts[2]), int(parts[3])
        d1 = recv_exact(s, l1); d2 = recv_exact(s, l2)

```

## Test Case1

### End-to-End Verification of Client-Server File Sync

- **Precondition**

- Server1 (`server1.py`) and Server2 (`server2.py`) are deployed and accessible from the Client.
- `client.py` is configured to connect to both servers.
- Test files are available on both the servers .

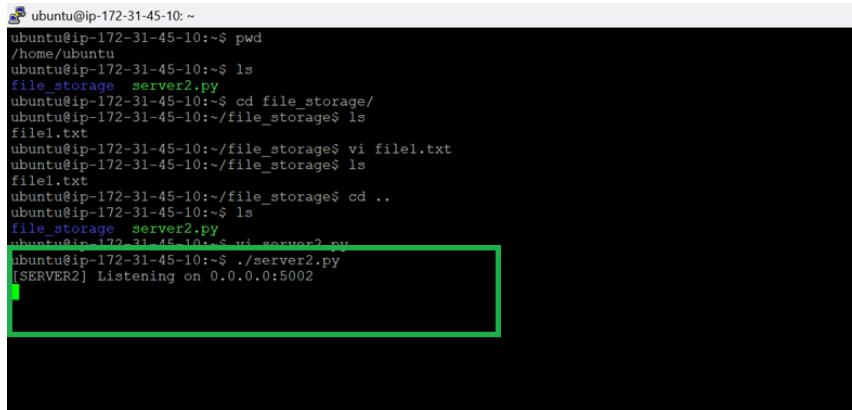
- **Test Steps**

- Start **Server1** by running `server1.py`.
- Start **Server2** by running `server2.py`.
- Start the **Client** by running `client.py`.

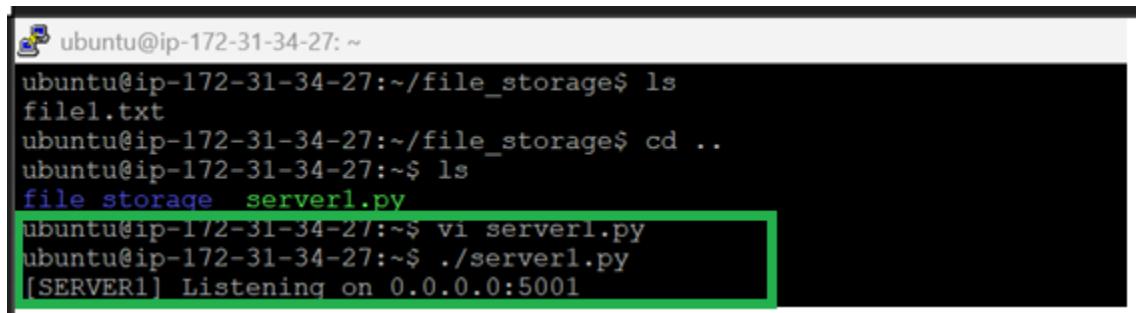
- Initiate file synchronization from the client.

- **Expected Result**

- The client should successfully connect to both **Server1** and **Server2**.
- Files from the servers should be visible in the client's file system after sync.
- When file conflicts exist (e.g., identical files on both servers), synchronization should complete without data loss.



```
ubuntu@ip-172-31-45-10:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-45-10:~$ ls
file_storage  server2.py
ubuntu@ip-172-31-45-10:~$ cd file_storage/
ubuntu@ip-172-31-45-10:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$ vi file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$ cd ..
ubuntu@ip-172-31-45-10:~$ ls
file_storage  server2.py
ubuntu@ip-172-31-45-10:~$ vi server2.py
ubuntu@ip-172-31-45-10:~$ ./server2.py
[SERVER2] Listening on 0.0.0.0:5002
```



```
ubuntu@ip-172-31-34-27:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-34-27:~$ ls
file1.txt
ubuntu@ip-172-31-34-27:~$ cd ..
ubuntu@ip-172-31-34-27:~$ ls
file_storage  server1.py
ubuntu@ip-172-31-34-27:~$ vi server1.py
ubuntu@ip-172-31-34-27:~$ ./server1.py
[SERVER1] Listening on 0.0.0.0:5001
```

```
ubuntu@ip-172-31-34-69: ~
ubuntu@ip-172-31-34-69:~$ ls -lthr
total 4.0K
-rwxrwxrwx 1 ubuntu ubuntu 1.7K Sep  7 14:54 client.py
ubuntu@ip-172-31-34-69:~$ ./client.py
[CLIENT] Received BOTH copies:

--- SERVER1 version ---
Content: "Report version from SERVER1"

--- SERVER2 version ---
Content: "Report version from SERVER2"

ubuntu@ip-172-31-34-69:~$
```

## Test Case2

### File Read Operation with Missing File on One Server

- **Precondition**

1. `file1.txt` exists on both Server1 and Server2.
2. Client, Server1, and Server2 are configured properly with the Python scripts.

- **Test Steps**

1. Remove `file1.txt` from **Server1**.
2. Execute the Python script ([client.py](#)) from **Client**.
3. Execute the Python script ([server1.py](#)) from **Server1**.
4. Execute the Python script ([server2.py](#)) from **Server2**.

- **Expected Result**

Since **file1.txt** is removed from Server1, the client should successfully read the file content from Server2 as per the configuration.

```
ubuntu@ip-172-31-34-27: ~
* Management:      https://landscape.canonical.com
* Support:        https://ubuntu.com/pro

System information as of Sun Sep 14 06:49:03 UTC 2025

System load: 0.07          Temperature:      -273.1 C
Usage of /: 34.5% of 6.71GB  Processes:        120
Memory usage: 29%          Users logged in:   0
Swap usage:  0%          IPv4 address for ens5: 172.31.34.27

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

9 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Sep 13 11:18:39 2025 from 122.172.87.11
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-34-27:~$ ls -ltrh
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 3.2K Sep  7 14:43 server1.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 08:04 file_storage/
ubuntu@ip-172-31-34-27:~$ cd file_storage/
ubuntu@ip-172-31-34-27:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ rm -rf file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ ls
```

```
ubuntu@ip-172-31-45-10: ~
login as: ubuntu
Authenticating with public key "DCFileSystemKeyPair070925"
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

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

System information as of Sun Sep 14 06:49:13 UTC 2025

System load:  0.0          Temperature:          -273.1 C
Usage of /:   34.5% of 6.71GB  Processes:            110
Memory usage: 31%          Users logged in:      0
Swap usage:   0%          IPv4 address for ens5: 172.31.45.10

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

9 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Sep 13 11:19:17 2025 from 122.172.87.11
ubuntu@ip-172-31-45-10:~$ ls -ltrh
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 1.6K Sep  7 14:33 server2.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 08:10 file_storage
ubuntu@ip-172-31-45-10:~$ ./server2.py
[SERVER2] Listening on 0.0.0.0:5002
[SERVER2] Sent file1.txt (39 bytes) to ('172.31.34.27', 41946)
```

```
ubuntu@ip-172-31-34-27: ~
* Management:      https://landscape.canonical.com
* Support:        https://ubuntu.com/pro

System information as of Sun Sep 14 06:49:03 UTC 2025

System load: 0.07          Temperature:          -273.1 C
Usage of /: 34.5% of 6.71GB Processes:          120
Memory usage: 29%          Users logged in:      0
Swap usage:  0%          IPv4 address for ens5: 172.31.34.27

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

9 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Sep 13 11:18:39 2025 from 122.172.87.11
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-34-27:~$ ls -ltrh
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 3.2K Sep  7 14:43 server1.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 08:04 file_storage
ubuntu@ip-172-31-34-27:~$ cd file_storage/
ubuntu@ip-172-31-34-27:~/file_storage$ ls
file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ rm -rf file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ ls
ubuntu@ip-172-31-34-27:~/file_storage$ cd ..
ubuntu@ip-172-31-34-27:~$ ./server1.py
[SERVER1] Listening on 0.0.0.0:5001
[SERVER1] CLIENT ('172.31.34.69', 50420) requested file1.txt
[SERVER1] Only SERVER2 had it → sent ONE
```

```
ubuntu@ip-172-31-34-69: ~
└─ login as: ubuntu
└─ Authenticating with public key "DCFileSystemKeyPair070925"
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

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

System information as of Sun Sep 14 06:48:47 UTC 2025

System load: 0.0 Temperature: -273.1 C
Usage of /: 34.5% of 6.71GB Processes: 110
Memory usage: 30% Users logged in: 0
Swap usage: 0% IPv4 address for ens5: 172.31.34.69

Expanded Security Maintenance for Applications is not enabled.

6 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Sep 13 11:19:26 2025 from 122.172.97.11
buntu@ip-172-31-34-69:~$ ./client.py
CLIENT] Received ONE copy:
Content: "Report version from SERVER2"

buntu@ip-172-31-34-69:~$
```

## Test Case3

### File Read Operation When File Missing on Both Servers

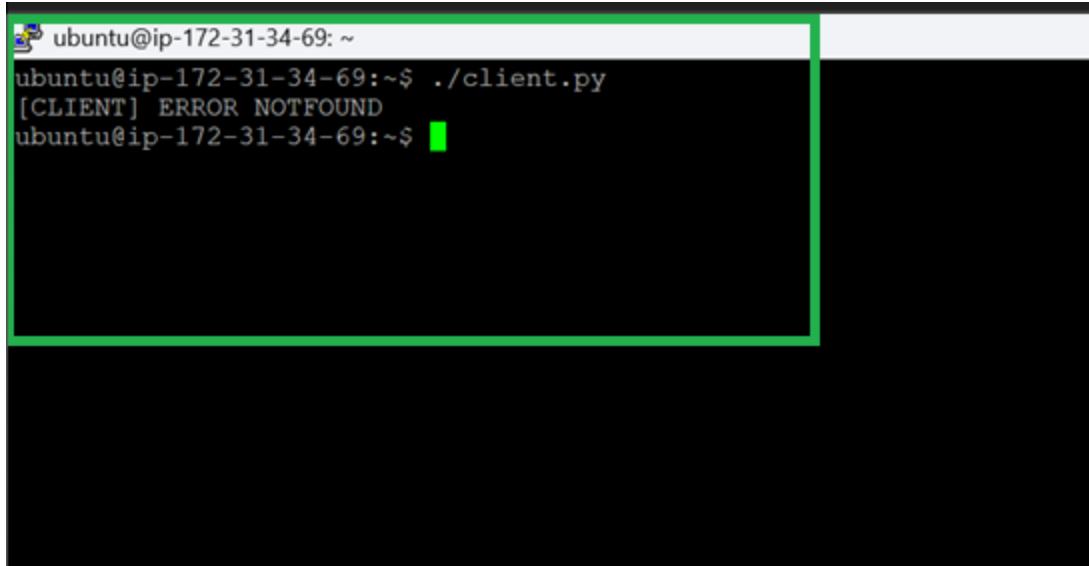
- **Precondition**
  - **file1.txt** is **not present** on either Server1 or Server2.
  - Client, Server1, and Server2 are configured properly with the Python scripts.
- **Test Steps**
  - Ensure **file1.txt** is deleted from **Server1**.
  - Ensure **file1.txt** is deleted from **Server2**.
  - Execute the Python script from the **Client** machine.

- **Expected Result**

- The client script should fail to retrieve the file from either server.
- The client should display an error message “[CLIENT] ERROR NOTFOUND”.

```
ubuntu@ip-172-31-34-27:~  
* Management: https://landscape.canonical.com  
* Support: https://ubuntu.com/pro  
  
System information as of Sun Sep 14 06:49:03 UTC 2025  
  
System load: 0.07 Temperature: -273.1 C  
Usage of /: 34.5% of 6.71GB Processes: 120  
Memory usage: 29% Users logged in: 0  
Swap usage: 0% IPv4 address for ens5: 172.31.34.27  
  
* Ubuntu Pro delivers the most comprehensive open source security and  
compliance features.  
  
https://ubuntu.com/aws/pro  
  
Expanded Security Maintenance for Applications is not enabled.  
  
9 updates can be applied immediately.  
To see these additional updates run: apt list --upgradable  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
*** System restart required ***  
Last login: Sat Sep 13 11:18:39 2025 from 122.172.87.11  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
ubuntu@ip-172-31-34-27:~$ ls -ltrh  
total 8.0K  
-rwxrwxr-x 1 ubuntu ubuntu 3.2K Sep 7 14:43 server1.py  
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 08:04 file_storage  
ubuntu@ip-172-31-34-27:~$ cd file_storage/  
ubuntu@ip-172-31-34-27:~/file_storage$ ls  
file1.txt  
ubuntu@ip-172-31-34-27:~/file_storage$ rm -rf file1.txt  
ubuntu@ip-172-31-34-27:~/file_storage$ ls
```

```
ubuntu@ip-172-31-45-10:~  
ubuntu@ip-172-31-45-10:~$ ls -ltrh  
total 8.0K  
-rwxrwxr-x 1 ubuntu ubuntu 1.6K Sep 7 14:55 server2.py  
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 13 08:10 file_storage/  
ubuntu@ip-172-31-45-10:~/file_storage$ ls  
file1.txt  
ubuntu@ip-172-31-45-10:~/file_storage$ rm -rf file1.txt  
ubuntu@ip-172-31-45-10:~/file_storage$ cd ..  
ubuntu@ip-172-31-45-10:~$ ./server2.py  
[SERVER2] Listening on 0.0.0.0:5002  
[SERVER2] file1.txt not found for ('172.31.34.27', 51210)
```



ubuntu@ip-172-31-34-69: ~  
ubuntu@ip-172-31-34-69:~\$ ./client.py  
[CLIENT] ERROR NOTFOUND  
ubuntu@ip-172-31-34-69:~\$

## Test Case4

### File Read Operation When Servers Have Identical File

Single Copy Sent by Server1 to client if files are identical both on Server 1 and Server 2.

- **Precondition**
  - `file1.txt` exists on both **Server1** and **Server2** with identical content.
  - Client, Server1, and Server2 are configured properly with the Python scripts.
- **Test Steps**
  - Keep `file1.txt` on **Server1** and **Server2** with the same content.
  - Execute the Python script from the **Client** machine.
  - Server1 should handle the client request and send **one copy** of `file1.txt`.
- **Expected Result**
  - The client should successfully receive and read the file content from **Server1**.
  - The client should **not** receive duplicate copies, even though the file also exists on Server2.
  - Output should show the file content exactly **once**.

**Server1 Content:**

```
ubuntu@ip-172-31-34-27:~/file_storage
ubuntu@ip-172-31-34-27:~/file_storage$ cd file_storage/
ubuntu@ip-172-31-34-27:~/file_storage$ ls
ubuntu@ip-172-31-34-27:~/file_storage$ touch file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ vi file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ cat file1.txt
Hello, this is the test file.
This file is identical on both Server1 and Server2.
Only one copy should be sent to the client.

ubuntu@ip-172-31-34-27:~/file_storage$
```

### Server2 Content:

```
ubuntu@ip-172-31-45-10:~/file_storage
ubuntu@ip-172-31-45-10:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-45-10:~$ ls -ltrh
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 1.6K Sep 7 14:33 server2.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 14 07:06 file_storage
ubuntu@ip-172-31-45-10:~$ cd file_storage/
ubuntu@ip-172-31-45-10:~/file_storage$ ls
ubuntu@ip-172-31-45-10:~/file_storage$ vi file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$ cat file1.txt
Hello, this is the test file.
This file is identical on both Server1 and Server2.
Only one copy should be sent to the client.

ubuntu@ip-172-31-45-10:~/file_storage$
```

## Execute scripts `client.py`, `server1.py` & `server2.py`

```
ubuntu@ip-172-31-45-10: ~
ubuntu@ip-172-31-45-10:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-45-10:~$ ls -lthr
total 8.0K
-rwxrwxr-x 1 ubuntu ubuntu 1.6K Sep  7 14:33 server2.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Sep 14 07:06 file_storage
ubuntu@ip-172-31-45-10:~$ cd file_storage/
ubuntu@ip-172-31-45-10:~/file_storage$ ls
ubuntu@ip-172-31-45-10:~/file_storage$ vi file1.txt
ubuntu@ip-172-31-45-10:~/file_storage$ cat file1.txt
Hello, this is the test file.
This file is identical on both Server1 and Server2.
Only one copy should be sent to the client.

ubuntu@ip-172-31-45-10:~/file_storage$ cd ..
ubuntu@ip-172-31-45-10:~$ ./server2.py
[SERVER2] Listening on 0.0.0:5002
[SERVER2] Sent file1.txt (127 bytes) to ('172.31.34.27', 44882)
```

```
ubuntu@ip-172-31-34-27: ~
ubuntu@ip-172-31-34-27:~$ cd file_storage/
ubuntu@ip-172-31-34-27:~/file_storage$ ls
ubuntu@ip-172-31-34-27:~/file_storage$ touch file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ vi file1.txt
ubuntu@ip-172-31-34-27:~/file_storage$ cat file1.txt
Hello, this is the test file.
This file is identical on both Server1 and Server2.
Only one copy should be sent to the client.

ubuntu@ip-172-31-34-27:~/file_storage$ cd ..
ubuntu@ip-172-31-34-27:~$ ./server1.py
[SERVER1] Listening on 0.0.0.0:5001
[SERVER1] CLIENT ('172.31.34.69', 60126) requested file1.txt
[SERVER1] Both matched → sent ONE
```

```
ubuntu@ip-172-31-34-69: ~
ubuntu@ip-172-31-34-69:~$ ./client.py
[CLIENT] Received ONE copy:
Hello, this is the test file.
This file is identical on both Server1 and Server2.
Only one copy should be sent to the client.
```

```
ubuntu@ip-172-31-34-69:~$
```