Indian Institute of Technology Jodhpur

Fundamentals of Distributed Systems

Assignment– 1

Student Name: Avinash Kumar
Roll Number: G24AI2025.

Question 1: Vector Clocks and Causal Ordering.

1. Introduction

In real-world distributed systems, events don't always arrive in the order we expect. A message might reach one machine before another, and a simple timestamp can't always capture the true sequence of events. That's where Vector Clocks come in. This project aims to solve that problem by creating a distributed key-value store where causal relationships are preserved across nodes—no matter the order of message delivery.

2. Objective

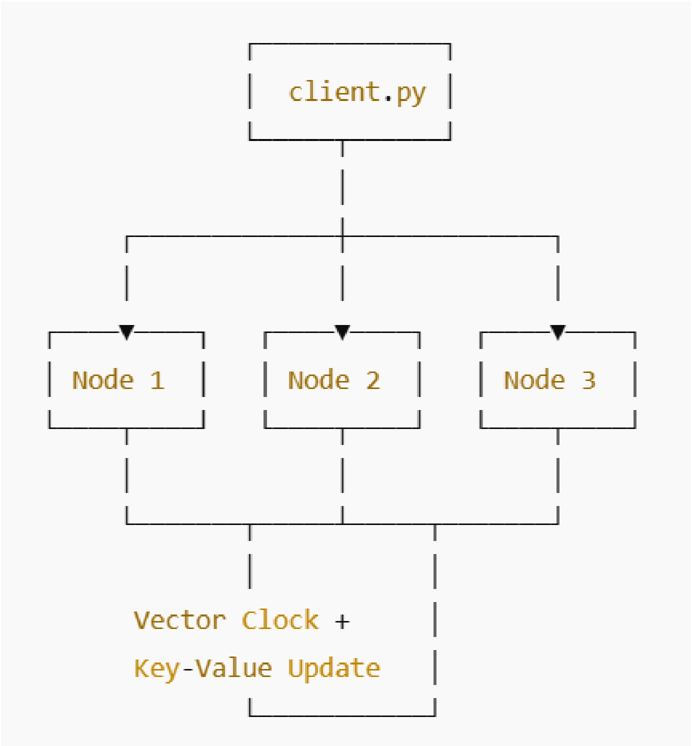
To build a distributed key-value store where:

- Updates are propagated with their causal history.
- Events are delivered only when causal dependencies are met.
- Each node handles its own **Vector Clock**.
- Communication is asynchronous and containerized using Docker and Docker Compose.

3. Components:

- **3 Nodes** (node1, node2, node3): Each maintains:
  - A local **key-value store**
  - A **Vector Clock** (dictionary {node_id: counter})
- **Client**:
  - Sends PUT/GET requests to random nodes.
  - Verifies causal consistency.
- **Message Passing**:
  - HTTP POST (JSON-based)
  - All messages contain a vector clock.
- **Buffer**:
  - Used when a received write's dependencies are not yet satisfied.

4. Communication Diagram.



5. Implementation.

A. Vector Clock Logic

- Each node maintains a vector: {'node1': 0, 'node2': 0, 'node3': 0}
- On local write:
  - Increment its own clock.

    - Broadcast to peers with updated clock.
- On receive:
    - If causally ready → apply write.
    - Else → buffer the write.

B. Python Files

node.py:

This file powers each node’s internal logic. Each node is a Flask web server with the following key components:

- Routes:
    - /put: Accepts key-value writes with vector clocks
    - /get: Returns values for given keys
    - /replicate: Receives updates from other nodes
- Buffering System:
    - Incoming messages are stored temporarily if dependencies are not met.
- Delivery Checker:
    - Periodically scans the buffer and delivers messages once they become causally safe.

client.py:

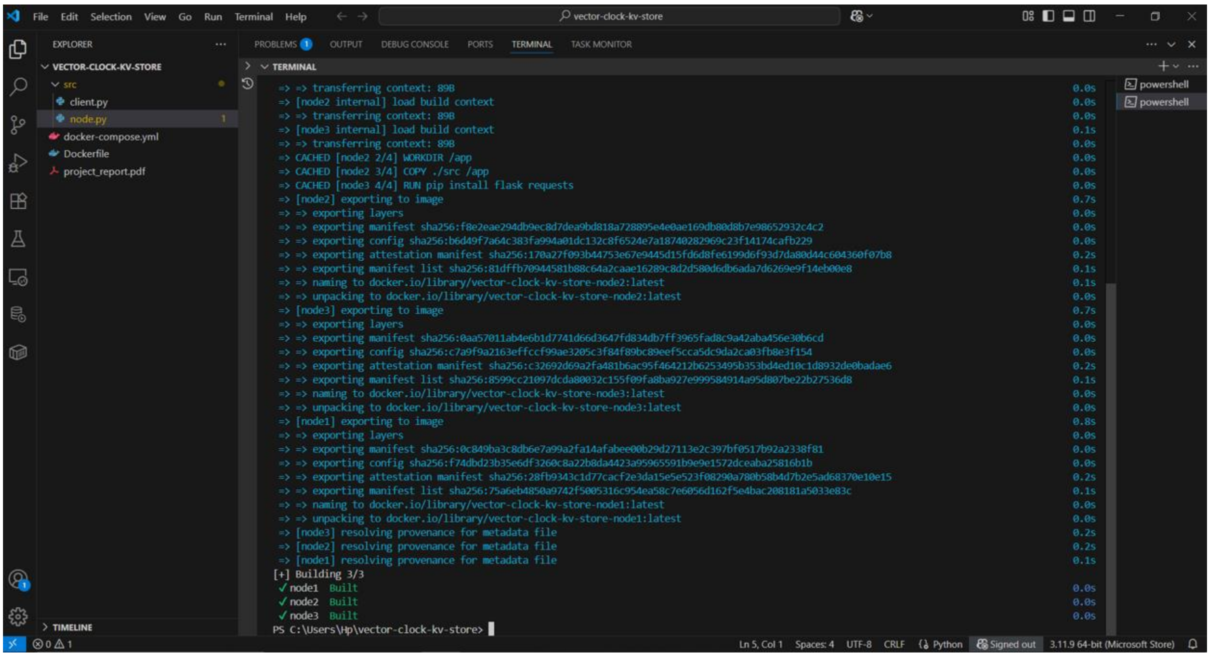
- Sends PUT → then GET to check data
- Simulates causality violation (e.g., update before receiving dependency)

5. Docker Setup

- **Dockerfile** (builds node container with Flask server)
- **docker-compose.yml**:
    - Defines 3 containers: node1, node2, node3
    - Network: bridge
    - Each node runs on a different port (e.g., 5001, 5002, 5003)

6. Sample Logs & Screenshots.

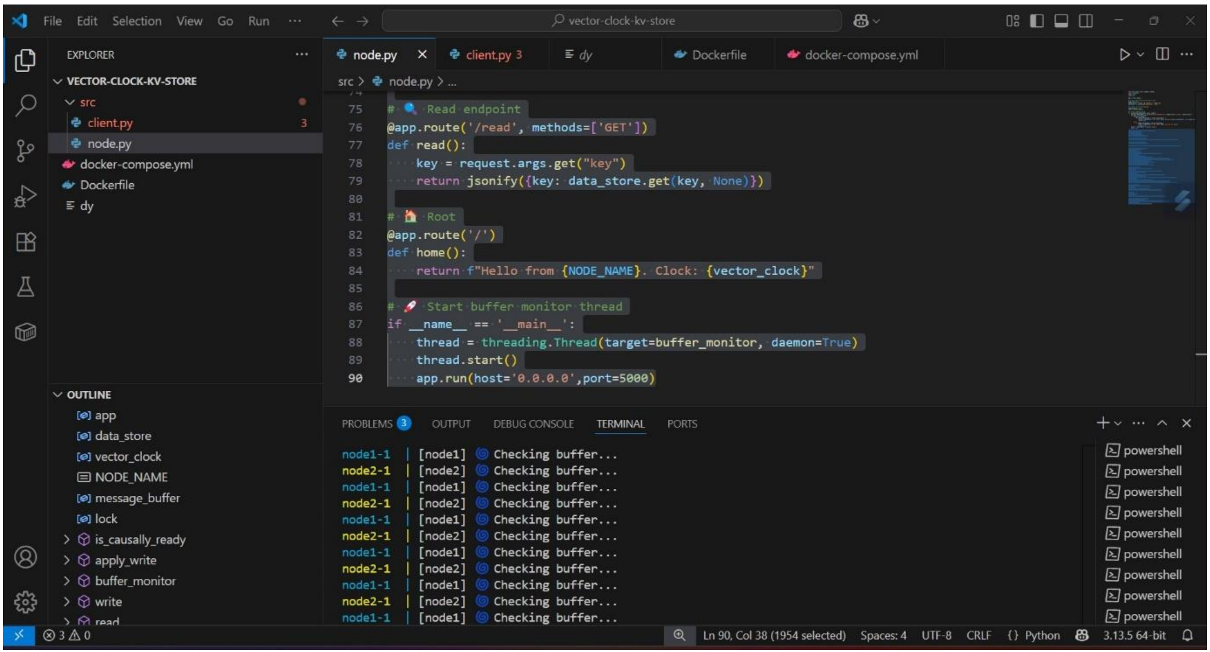
- Docker context was transferred and prepared.
- pip install fetched necessary Python dependencies, specifically flask and requests.
- Each node was assigned and tagged properly, with layers cached where applicable.



```
 => => transferring context: 89B                                                                                    0.0s
 => [node2 internal] load build context                                                                             0.0s
 => => transferring context: 89B                                                                                    0.0s
 => [node3 internal] load build context                                                                             0.1s
 => => transferring context: 89B                                                                                    0.0s
 => CACHED [node2 2/4] WORKDIR /app                                                                                 0.0s
 => CACHED [node2 3/4] COPY ./src /app                                                                              0.0s
 => CACHED [node3 4/4] RUN pip install flask requests                                                               0.0s
 => [node2] exporting to image                                                                                      0.7s
 => => exporting layers                                                                                             0.0s
 => => exporting manifest sha256:f8e2eae294db9ec8d7dea9bd818a728895e4e0ae169db80d8b7e98652932c4c2                   0.0s
 => => exporting config sha256:b6d49f7a64c383fa994a01dc132c8f6524e7a18740282969c23f14174cafb229                     0.0s
 => => exporting attestation manifest sha256:170a27f093b44753e67e9445d15fd6d8fe6199d6f93d7da80d44c604360f07b8       0.2s
 => => exporting manifest list sha256:81dffb70944581b88c64a2caae16289c8d2d580d6db6ada7d6269e9f14eb00e8              0.1s
 => => naming to docker.io/library/vector-clock-kv-store-node2:latest                                               0.1s
 => => unpacking to docker.io/library/vector-clock-kv-store-node2:latest                                            0.0s
 => [node3] exporting to image                                                                                      0.7s
 => => exporting layers                                                                                             0.0s
 => => exporting manifest sha256:0aa57011ab4e6b1d7741d66d3647fd834db7ff3965fad8c9a42aba456e30b6cd                   0.0s
 => => exporting config sha256:c7a9f9a2163effccf99ae3205c3f84f89bc89eef5cca5dc9da2ca03fb8e3f154                     0.0s
 => => exporting attestation manifest sha256:c32692d69a2fa481b6ac95f464212b6253495b353bd4ed10c1d8932de0badae6       0.2s
 => => exporting manifest list sha256:8599cc21097dcda80032c155f09fa8ba927e999584914a95d807be22b27536d8              0.1s
 => => naming to docker.io/library/vector-clock-kv-store-node3:latest                                               0.0s
 => => unpacking to docker.io/library/vector-clock-kv-store-node3:latest                                            0.0s
 => [node1] exporting to image                                                                                      0.8s
 => => exporting layers                                                                                             0.0s
 => => exporting manifest sha256:0c849ba3c8db6e7a99a2fa14afabee00b29d27113e2c397bf0517b92a2338f81                   0.0s
 => => exporting config sha256:f74dbd23b35e6df3260c8a22b8da4423a95965591b9e9e1572dceaba25816b1b                     0.0s
 => => exporting attestation manifest sha256:28fb9343c1d77cacf2e3da15e5e523f08290a780b58b4d7b2e5ad68370e10e15       0.2s
 => => exporting manifest list sha256:75a6eb4850a9742f5005316c954ea58c7e6056d162f5e4bac208181a5033e83c              0.1s
 => => naming to docker.io/library/vector-clock-kv-store-node1:latest                                               0.0s
 => => unpacking to docker.io/library/vector-clock-kv-store-node1:latest                                            0.0s
 => [node3] resolving provenance for metadata file                                                                  0.2s
 => [node2] resolving provenance for metadata file                                                                  0.2s
 => [node1] resolving provenance for metadata file                                                                  0.1s
[+] Building 3/3
 ✓ node1  Built                                                                                                     0.0s
 ✓ node2  Built                                                                                                     0.0s
 ✓ node3  Built                                                                                                     0.0s
PS C:\Users\Hp\vector-clock-kv-store> 
```

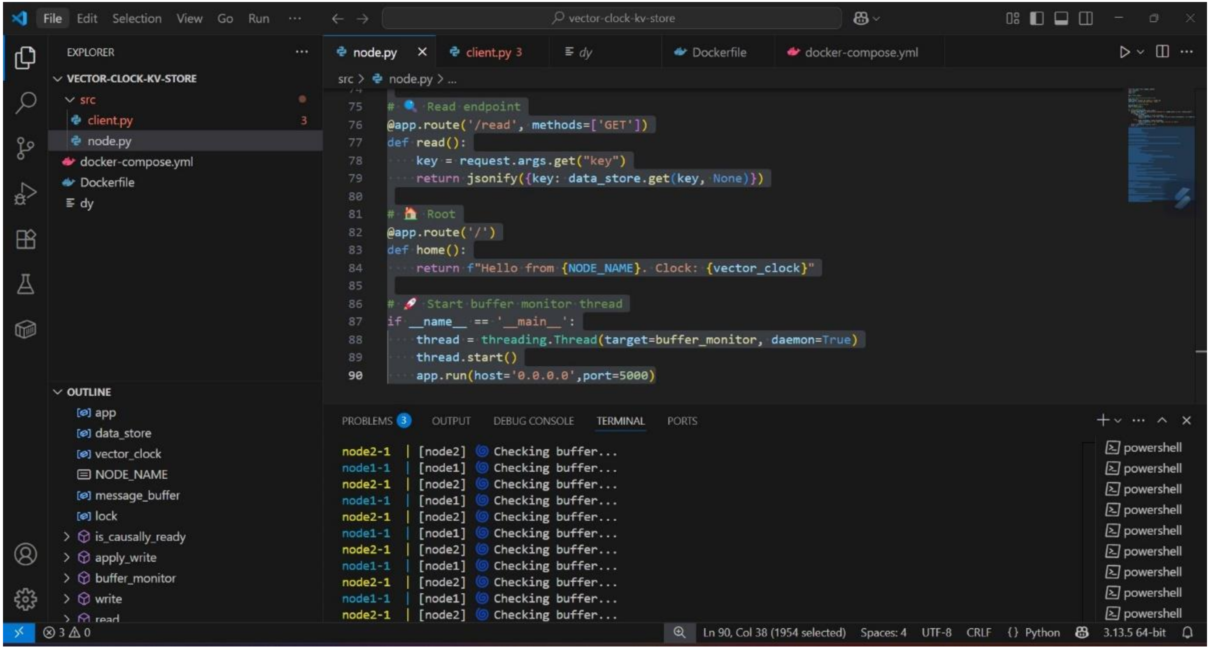
Now the below screenshot shows

- A GET route (/read) that allows the client to query key-value pairs.
- A home route that returns a heartbeat message with the node’s ID and current vector clock.
- A threaded buffer monitor, started in the __main__ block, which constantly checks for causally safe messages to apply.
- All three nodes (node1, node2, node3) are running in parallel.
- The logs show each node periodically checking its buffer:



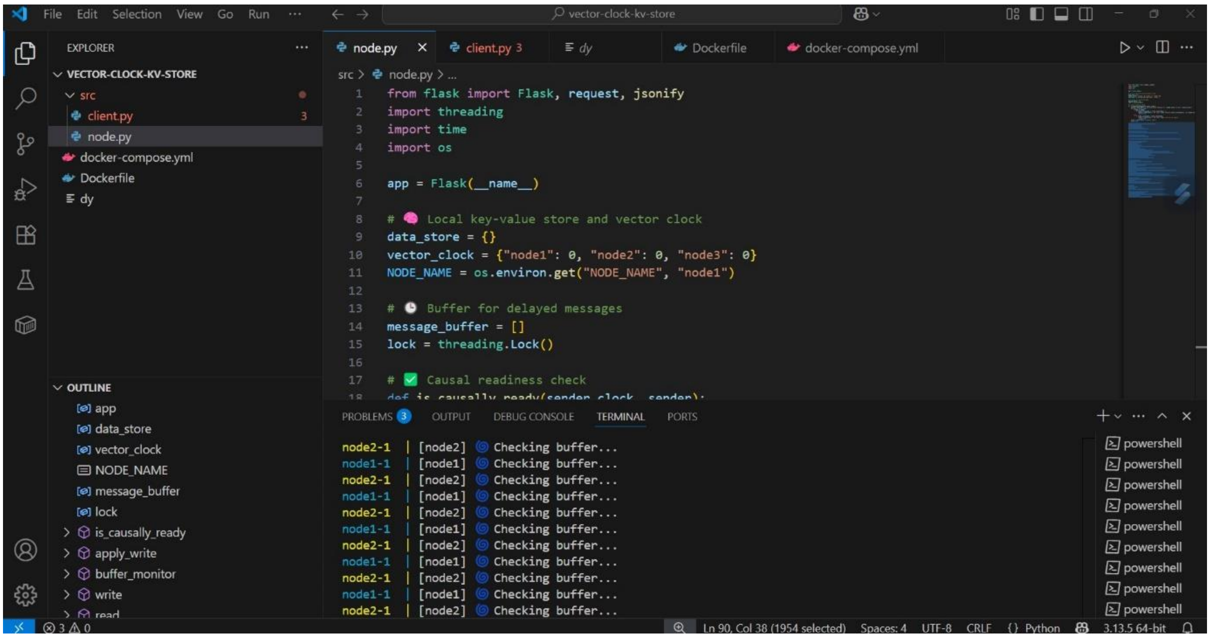
```python
# 🔍 Read endpoint
@app.route('/read', methods=['GET'])
def read():
    key = request.args.get("key")
    return jsonify({key: data_store.get(key, None)})

# 🏠 Root
@app.route('/')
def home():
    return f"Hello from {NODE_NAME}. Clock: {vector_clock}"

# 🚀 Start buffer monitor thread
if __name__ == '__main__':
    thread = threading.Thread(target=buffer_monitor, daemon=True)
    thread.start()
    app.run(host='0.0.0.0',port=5000)
```

```
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
```



```python
# 🔍 Read endpoint
@app.route('/read', methods=['GET'])
def read():
    key = request.args.get("key")
    return jsonify({key: data_store.get(key, None)})

# 🏠 Root
@app.route('/')
def home():
    return f"Hello from {NODE_NAME}. Clock: {vector_clock}"

# 🚀 Start buffer monitor thread
if __name__ == '__main__':
    thread = threading.Thread(target=buffer_monitor, daemon=True)
    thread.start()
    app.run(host='0.0.0.0',port=5000)
```

```
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
```



```python
from flask import Flask, request, jsonify
import threading
import time
import os

app = Flask(__name__)

# 🧠 Local key-value store and vector clock
data_store = {}
vector_clock = {"node1": 0, "node2": 0, "node3": 0}
NODE_NAME = os.environ.get("NODE_NAME", "node1")

# 🕒 Buffer for delayed messages
message_buffer = []
lock = threading.Lock()

# ✅ Causal readiness check
def is_causally_ready(sender_clock, sender):
```

```
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
node1-1  | [node1] 🌀 Checking buffer...
node2-1  | [node2] 🌀 Checking buffer...
```

Now the below screenshot shows

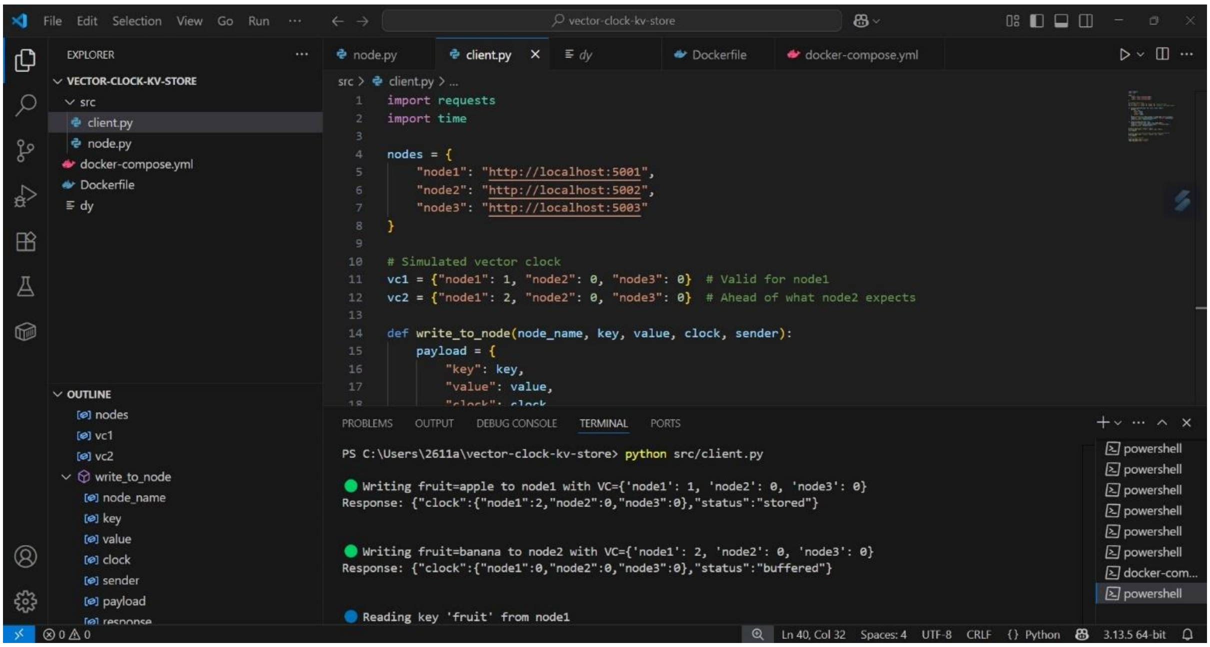
The client defines a dictionary of node URLs (Node 1–3).

It prepares simulated vector clocks:

- vc1 is a valid state for Node 1.
- vc2 simulates an event that is ahead of what Node 2 has seen, triggering a buffered response.

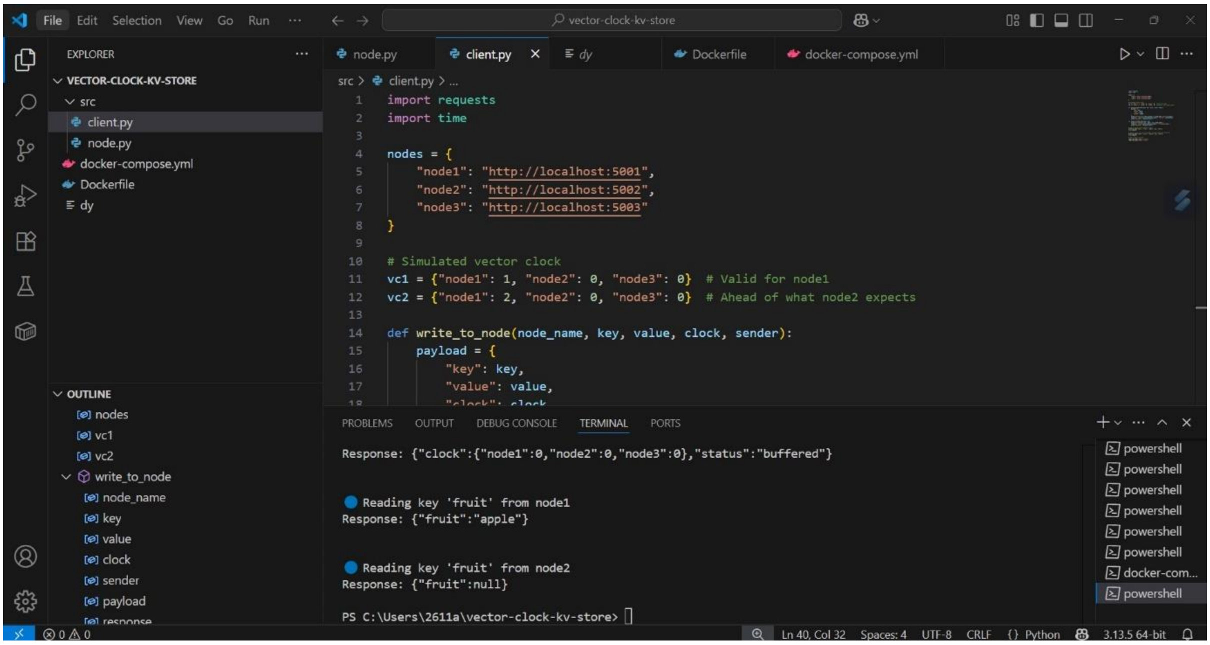
Function write_to_node() sends a PUT request containing:

- key, value, vector clock, and sender info.



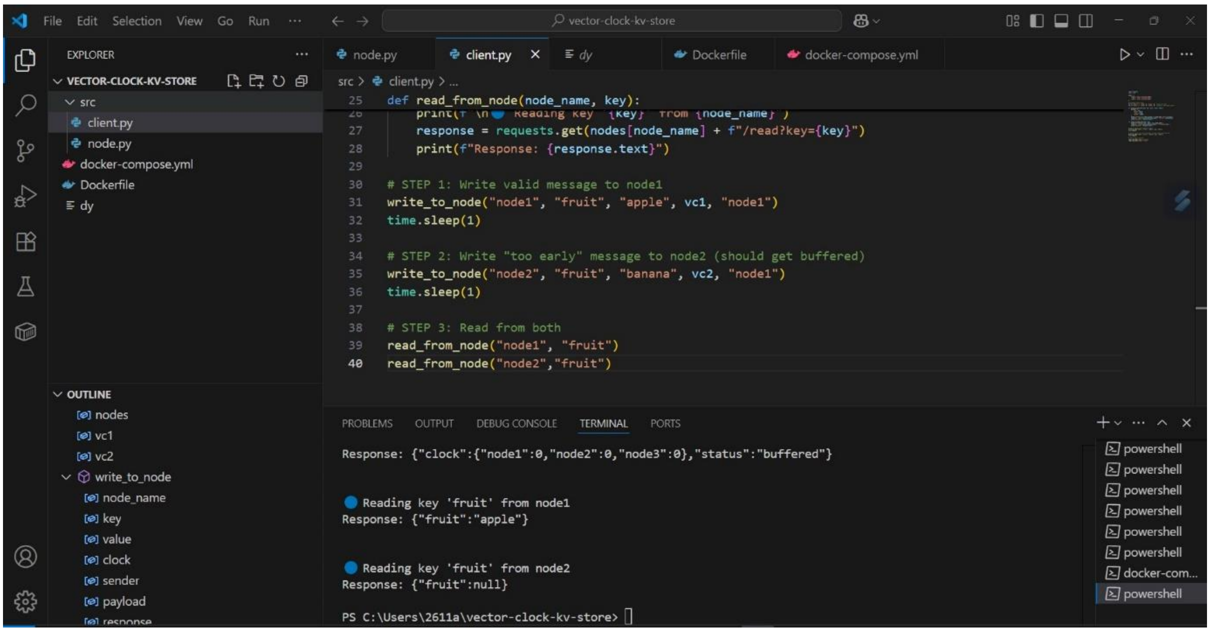
```python
import requests
import time

nodes = {
    "node1": "http://localhost:5001",
    "node2": "http://localhost:5002",
    "node3": "http://localhost:5003"
}

# Simulated vector clock
vc1 = {"node1": 1, "node2": 0, "node3": 0}  # Valid for node1
vc2 = {"node1": 2, "node2": 0, "node3": 0}  # Ahead of what node2 expects

def write_to_node(node_name, key, value, clock, sender):
    payload = {
        "key": key,
        "value": value,
        "clock": clock
```

```
PS C:\Users\2611a\vector-clock-kv-store> python src/client.py

● Writing fruit=apple to node1 with VC={'node1': 1, 'node2': 0, 'node3': 0}
Response: {"clock":{"node1":2,"node2":0,"node3":0},"status":"stored"}

● Writing fruit=banana to node2 with VC={'node1': 2, 'node2': 0, 'node3': 0}
Response: {"clock":{"node1":0,"node2":0,"node3":0},"status":"buffered"}

● Reading key 'fruit' from node1
```



```python
import requests
import time

nodes = {
    "node1": "http://localhost:5001",
    "node2": "http://localhost:5002",
    "node3": "http://localhost:5003"
}

# Simulated vector clock
vc1 = {"node1": 1, "node2": 0, "node3": 0}  # Valid for node1
vc2 = {"node1": 2, "node2": 0, "node3": 0}  # Ahead of what node2 expects

def write_to_node(node_name, key, value, clock, sender):
    payload = {
        "key": key,
        "value": value,
        "clock": clock
```

```
Response: {"clock":{"node1":0,"node2":0,"node3":0},"status":"buffered"}

● Reading key 'fruit' from node1
Response: {"fruit":"apple"}

● Reading key 'fruit' from node2
Response: {"fruit":null}

PS C:\Users\2611a\vector-clock-kv-store>
```



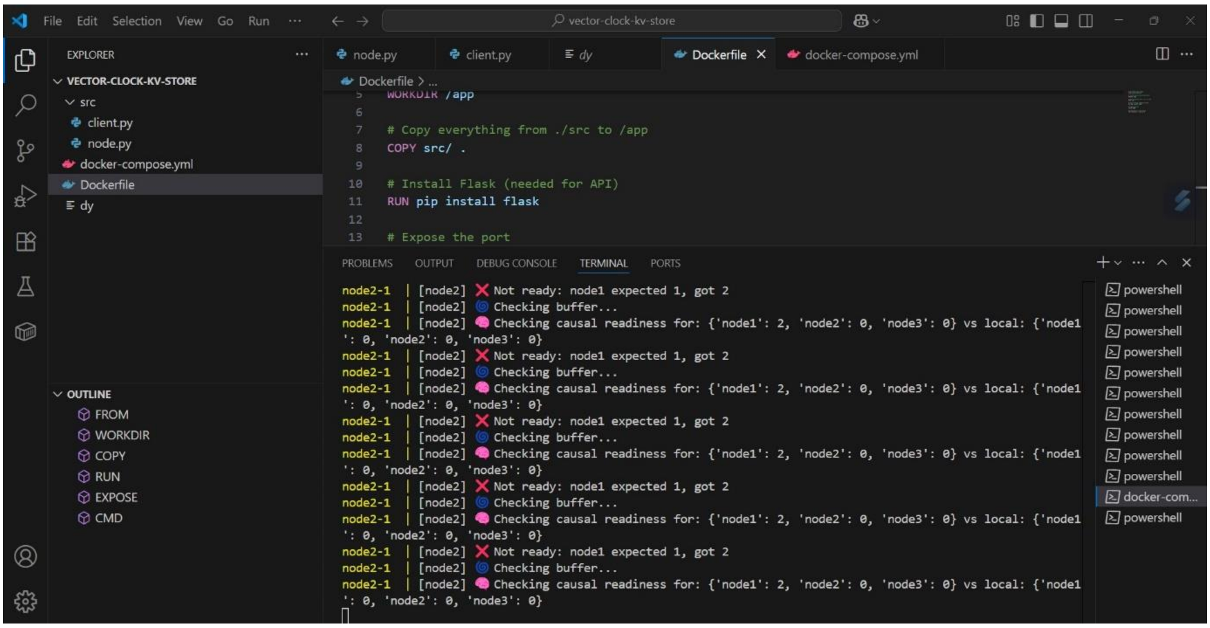
src > client.py > ...

```python
def read_from_node(node_name, key):
    print(f"\n🔵 Reading key '{key}' from {node_name}")
    response = requests.get(nodes[node_name] + f"/read?key={key}")
    print(f"Response: {response.text}")

# STEP 1: Write valid message to node1
write_to_node("node1", "fruit", "apple", vc1, "node1")
time.sleep(1)

# STEP 2: Write "too early" message to node2 (should get buffered)
write_to_node("node2", "fruit", "banana", vc2, "node1")
time.sleep(1)

# STEP 3: Read from both
read_from_node("node1", "fruit")
read_from_node("node2","fruit")
```

```
Response: {"clock":{"node1":0,"node2":0,"node3":0},"status":"buffered"}


🔵 Reading key 'fruit' from node1
Response: {"fruit":"apple"}


🔵 Reading key 'fruit' from node2
Response: {"fruit":null}

PS C:\Users\2611a\vector-clock-kv-store> 
```

Now the below screenshot shows

- The system is not blindly applying updates.
- Every node does a vector-wise causal check before applying a message.
- The buffer is working exactly as intended—holding back events until they're safe to deliver.



```
WORKDIR /app

# Copy everything from ./src to /app
COPY src/ .

# Install Flask (needed for API)
RUN pip install flask

# Expose the port
```

```
node2-1  | [node2] ❌ Not ready: node1 expected 1, got 2
node2-1  | [node2] 🌀 Checking buffer...
node2-1  | [node2] 🧠 Checking causal readiness for: {'node1': 2, 'node2': 0, 'node3': 0} vs local: {'node1
': 0, 'node2': 0, 'node3': 0}
node2-1  | [node2] ❌ Not ready: node1 expected 1, got 2
node2-1  | [node2] 🌀 Checking buffer...
node2-1  | [node2] 🧠 Checking causal readiness for: {'node1': 2, 'node2': 0, 'node3': 0} vs local: {'node1
': 0, 'node2': 0, 'node3': 0}
node2-1  | [node2] ❌ Not ready: node1 expected 1, got 2
node2-1  | [node2] 🌀 Checking buffer...
node2-1  | [node2] 🧠 Checking causal readiness for: {'node1': 2, 'node2': 0, 'node3': 0} vs local: {'node1
': 0, 'node2': 0, 'node3': 0}
node2-1  | [node2] ❌ Not ready: node1 expected 1, got 2
node2-1  | [node2] 🌀 Checking buffer...
node2-1  | [node2] 🧠 Checking causal readiness for: {'node1': 2, 'node2': 0, 'node3': 0} vs local: {'node1
': 0, 'node2': 0, 'node3': 0}
node2-1  | [node2] ❌ Not ready: node1 expected 1, got 2
node2-1  | [node2] 🌀 Checking buffer...
node2-1  | [node2] 🧠 Checking causal readiness for: {'node1': 2, 'node2': 0, 'node3': 0} vs local: {'node1
': 0, 'node2': 0, 'node3': 0}
```

Video Demo Link.

https://drive.google.com/file/d/1zbZ-tpDGxyrB7qJTjiQEmGathZD45Pzp/view?usp=sharing