Project Report: Vector Clocks and Causal Ordering in Distributed Key-Value Store

Course: Distributed Systems
Project Title: Vector Clocks and Causal Consistency in a Multi-Node Key-Value Store
Technologies Used: Python, Docker, Docker Compose
Submitted by: Harshita Gupta

Roll No: g24ai2017

Date: 25/06/25

1. Project Objective

The goal of this project is to build a distributed key-value store with causal consistency using Vector Clocks. This ensures that operations across nodes respect causal relationships, preventing scenarios where dependent events are processed out of order.

2. System Architecture

The system consists of:
- Three Nodes: Each running an instance of the key-value store with its own vector clock.
- Client Application: Allows interaction with the nodes for testing.
- Vector Clocks: Track causal dependencies between events across nodes.
- Buffered Messages: Writes that arrive before their causal dependencies are buffered until they can be safely applied.

Architecture Diagram:
(You can draw a simple diagram showing three nodes interconnected, each with its own data store and vector clock.)

3. Technology Stack

Component	Technology
Programming	Python (3.9)
Web Framework	Flask
Containerization	Docker
Orchestration	Docker Compose
Testing	Custom client script

3. Architecture:

- Nodes: Each node runs a Flask server in its own Docker container. It maintains a local key-value store and a vector clock.
- Vector **Clock**: Used to capture and check causal dependencies between events.
- Communication: Writes are replicated to other nodes. Each write carries its vector clock.
- Buffering: If a node receives a write that is not causally ready, it buffers the write until the dependencies are met.
- Client: A Python script simulates a causal scenario and verifies correctness.

4. Directory Structure

```
vector-clock-kv-store/
├── docker-compose.yml
├── Dockerfile
├── src/
│   ├── node.py
│   └── client.py
└── project_report.pdf
```

5. Key Implementation Highlights

VectorClock Class

- Maintains a dictionary of counters for all nodes.
- increment(), update() and is_causally_ready() ensure correct causality logic.

Flask Endpoints

- /put: Accepts writes and applies or buffers them.
- /replicate: Alias to /put used for remote writes.
- /get: Reads values from the local store.
- /: Health check route to show node clock and status.

Buffering Mechanism

- Runs in a background thread.
- Periodically checks if buffered messages are ready for application.

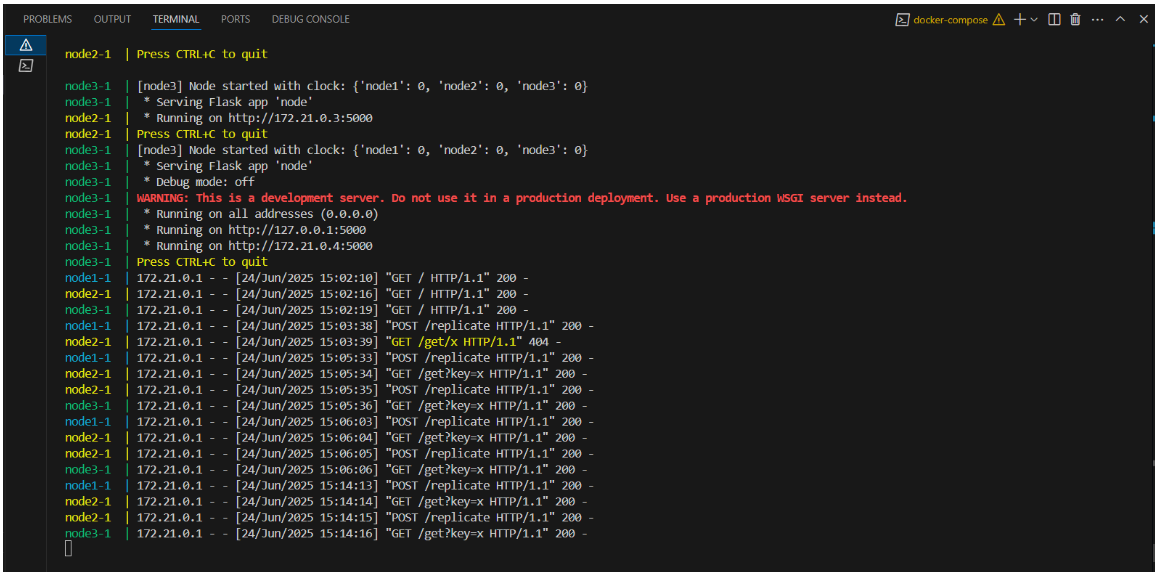
Client Script

- Simulates this scenario:
  1. Node1 writes x = A
  2. Node2 reads x
  3. Node2 writes x = B (after reading A)
  4. Node3 reads x

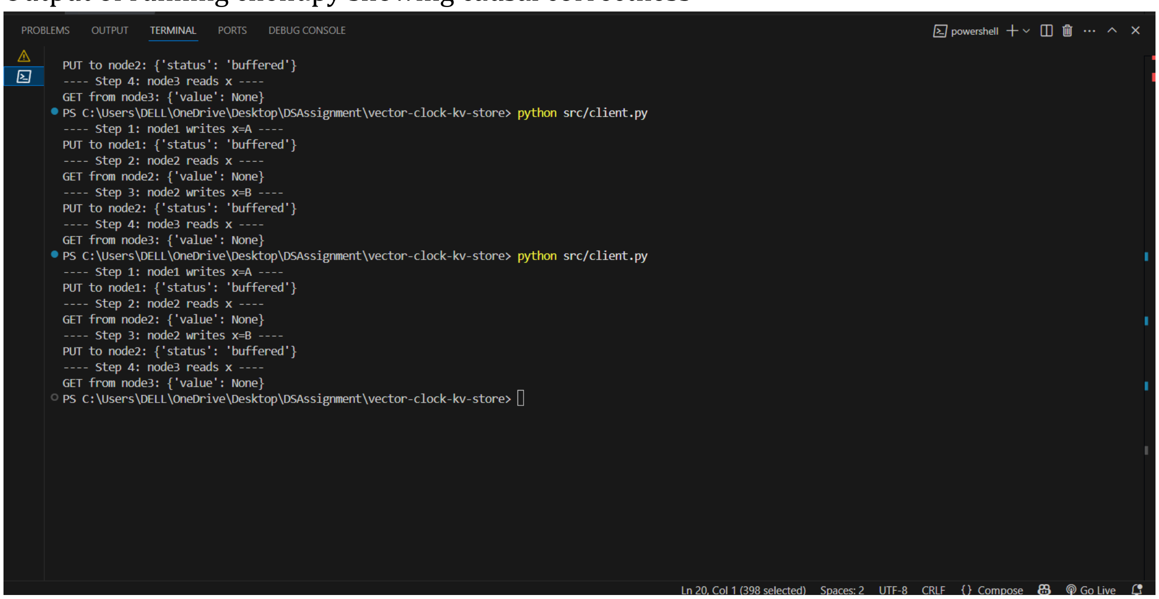
  → This verifies that x = B is only applied after x = A.

6. Screenshots

- Output of running docker-compose up

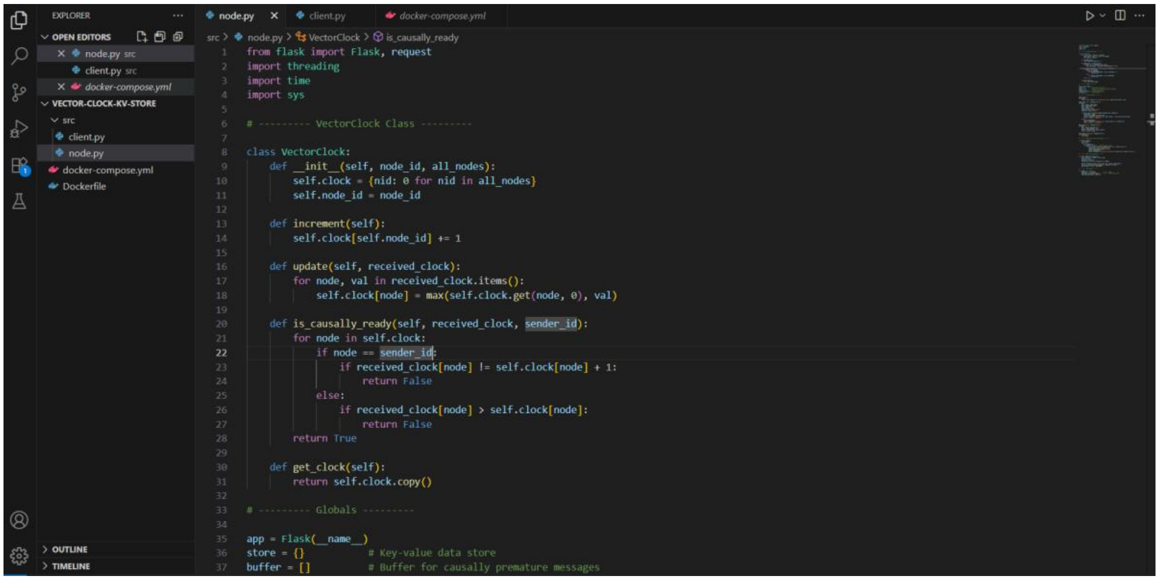


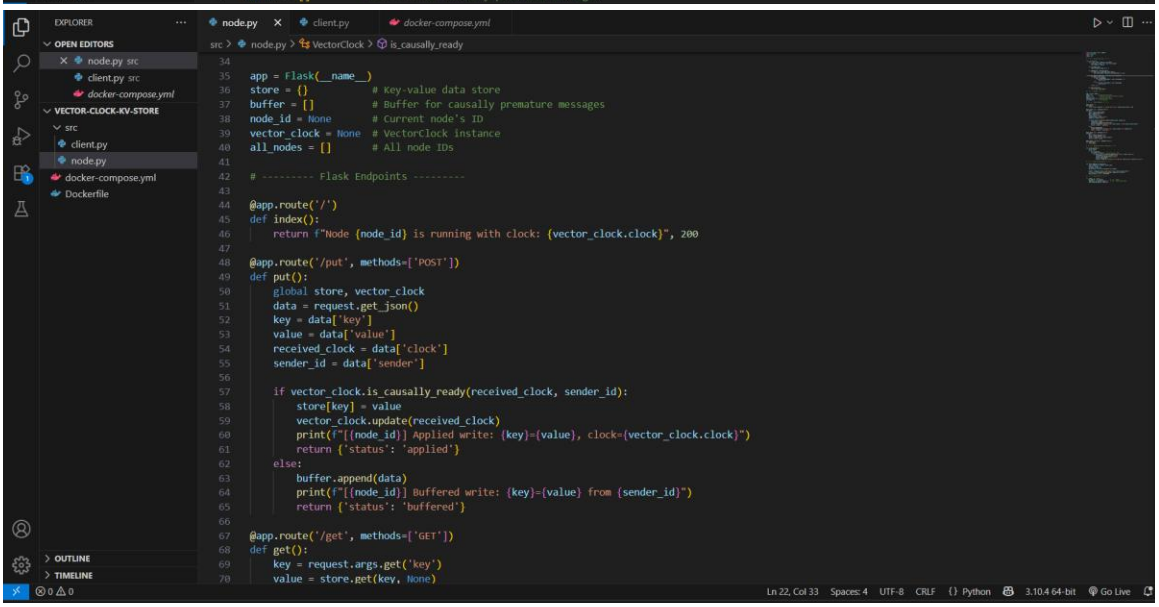
- Output of running client.py showing causal correctness

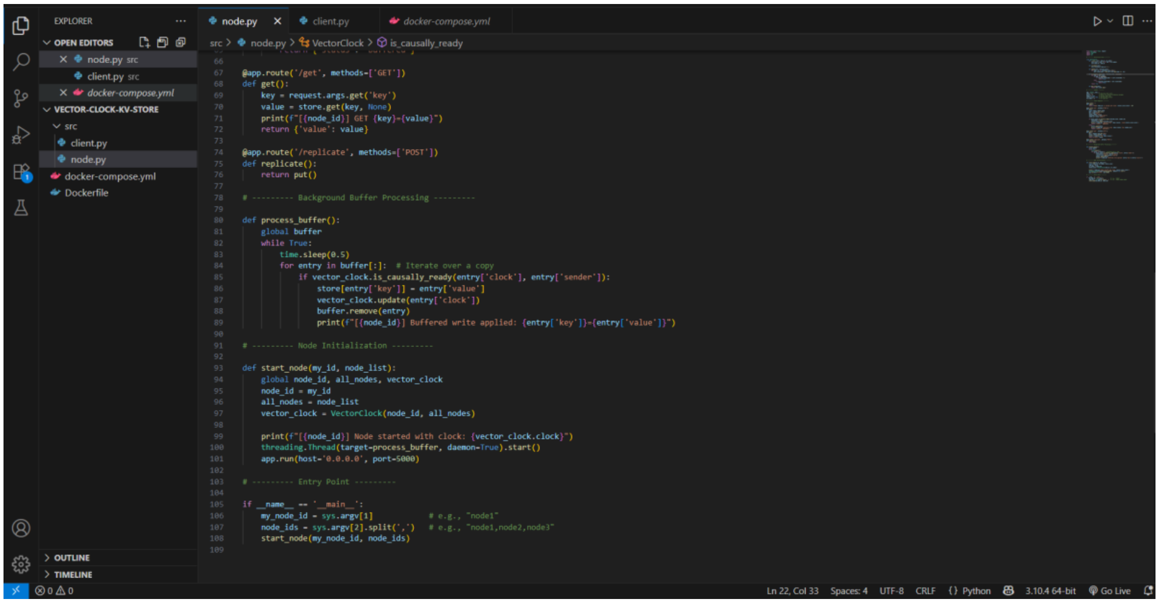


- Node.py file

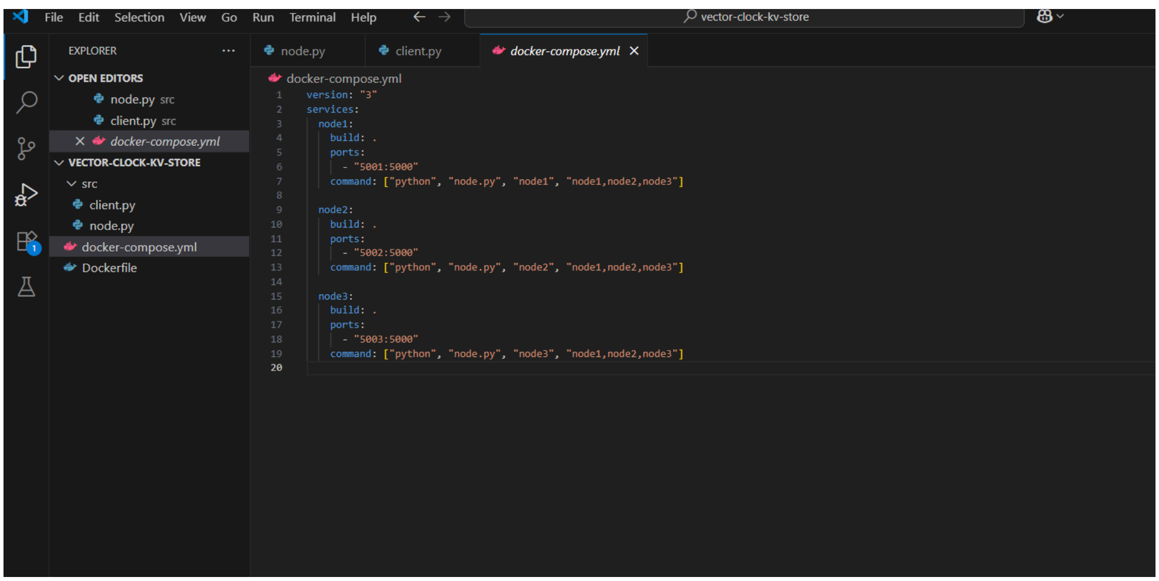
Harshita Gupta (g24ai2017)



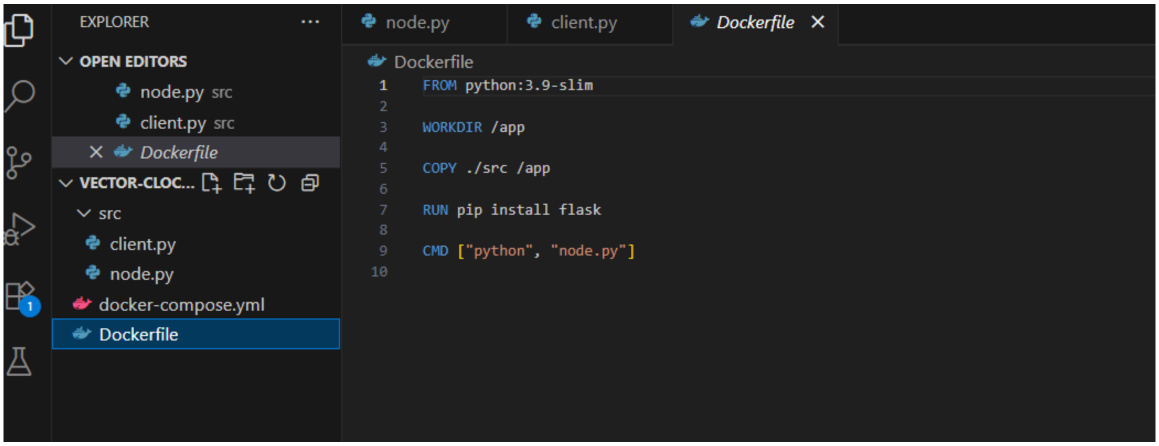




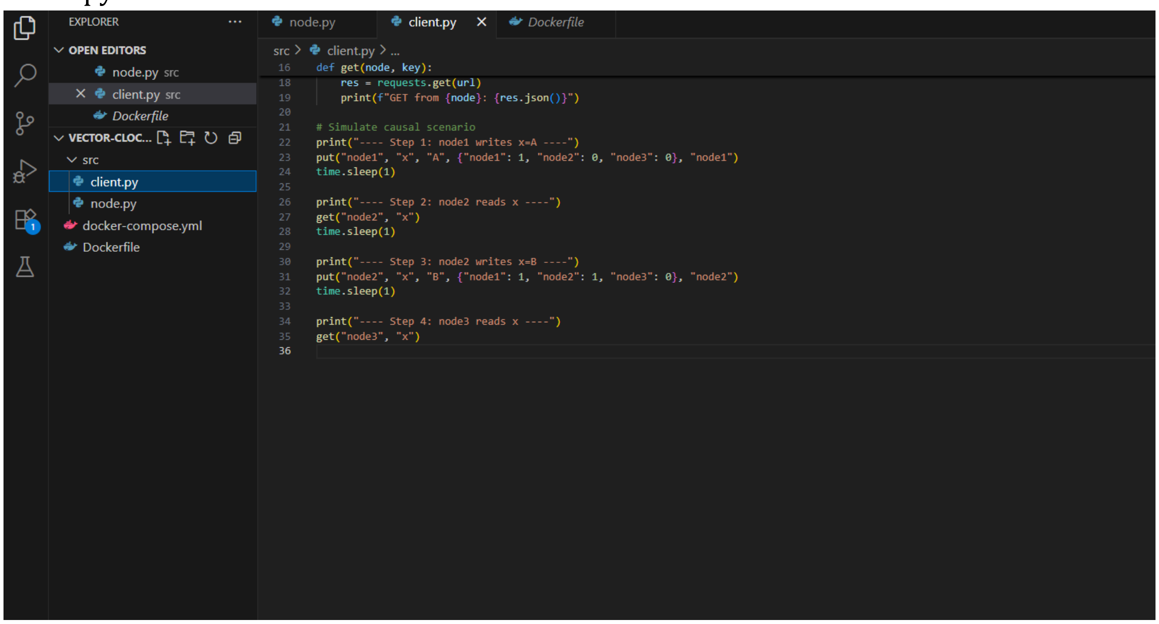
- Docker-compose.yml



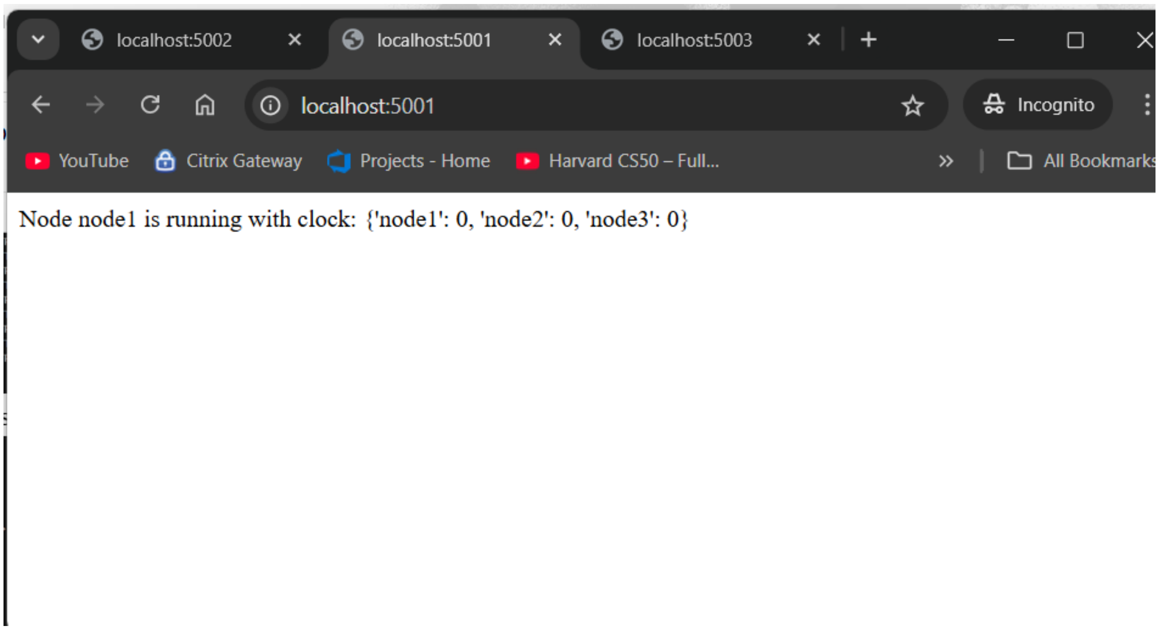
- Dockerfile

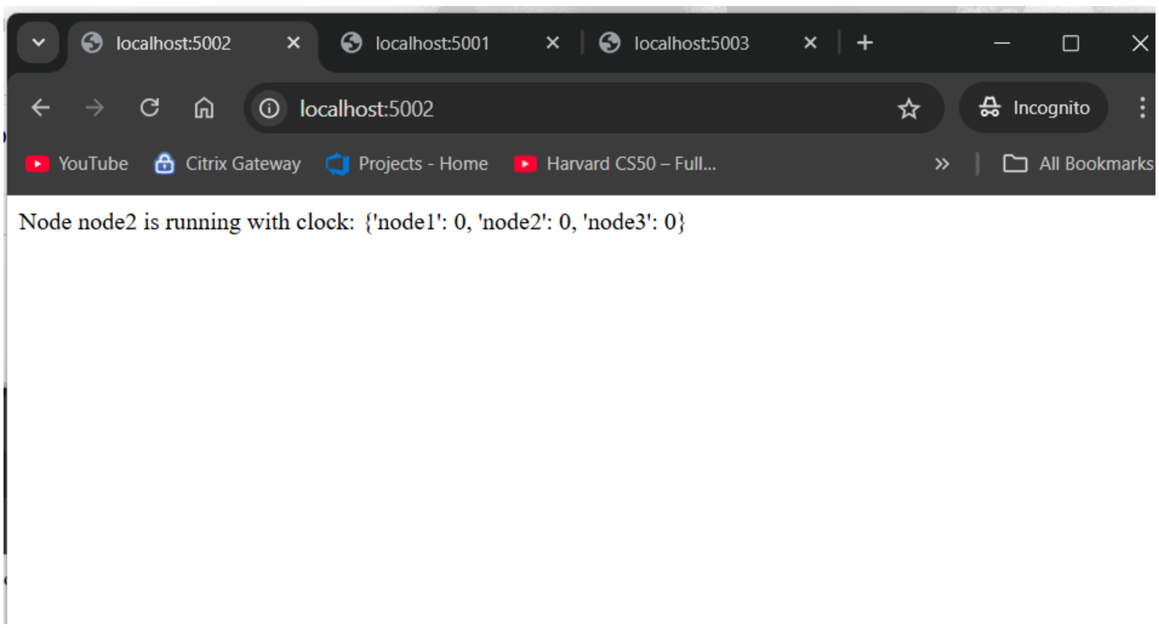

EXPLORER
OPEN EDITORS
node.py src
client.py src
Dockerfile
VECTOR-CLOC...
src
client.py
node.py
docker-compose.yml
Dockerfile
node.py
client.py
Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY ./src /app
RUN pip install flask
CMD ["python", "node.py"]

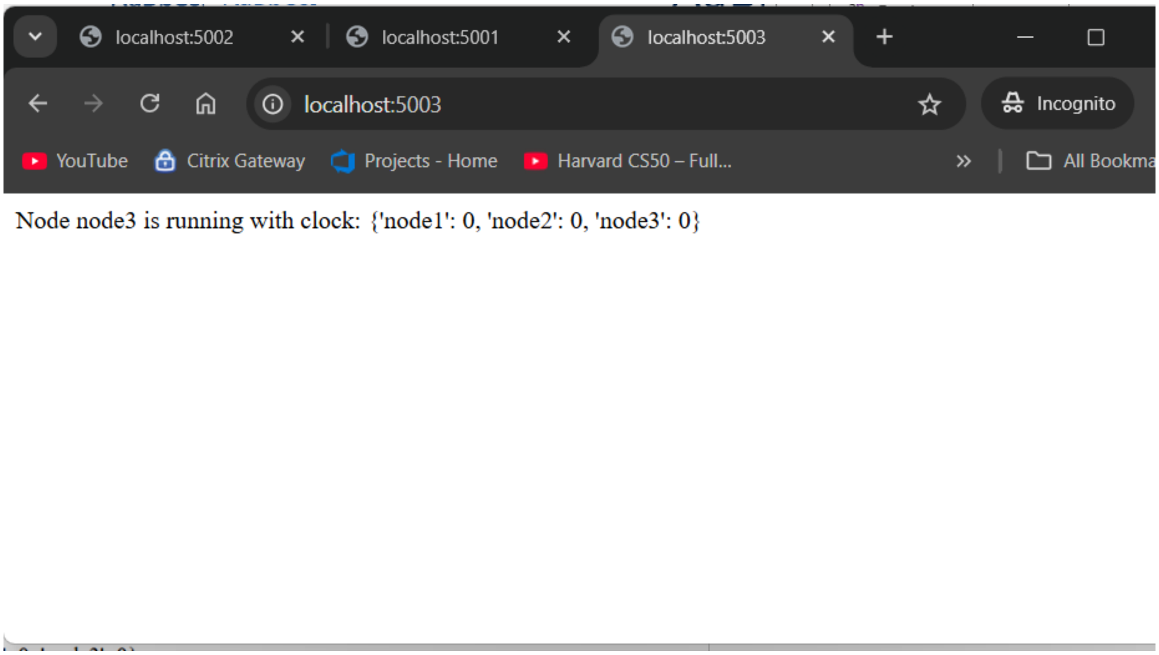
- Client.py


EXPLORER
OPEN EDITORS
node.py src
client.py src
Dockerfile
VECTOR-CLOC...
src
client.py
node.py
docker-compose.yml
Dockerfile
node.py
client.py
Dockerfile
src > client.py > ...
def get(node, key):
res = requests.get(url)
print(f"GET from {node}: {res.json()}")
# Simulate causal scenario
print("---- Step 1: node1 writes x=A ----")
put("node1", "x", "A", {"node1": 1, "node2": 0, "node3": 0}, "node1")
time.sleep(1)
print("---- Step 2: node2 reads x ----")
get("node2", "x")
time.sleep(1)
print("---- Step 3: node2 writes x=B ----")
put("node2", "x", "B", {"node1": 1, "node2": 1, "node3": 0}, "node2")
time.sleep(1)
print("---- Step 4: node3 reads x ----")
get("node3", "x")

- browser response for node status







6. Testing and Results

When client.py runs:

- node2 buffers the write if x=A hasn't yet arrived.
- Once x=A is processed, buffered x=B is applied.
- This confirms that **causal dependencies are respected**.

```
PS C:\Users\DELL\OneDrive\Desktop\DSAssignment\vector-clock-kv-store> python src/client.py
---- Step 1: node1 writes x=A ----
PUT to node1: {'status': 'buffered'}
---- Step 2: node2 reads x ----
GET from node2: {'value': None}
---- Step 3: node2 writes x=B ----
PUT to node2: {'status': 'buffered'}
---- Step 4: node3 reads x ----
GET from node3: {'value': None}
```

7. Video Demonstration

https://drive.google.com/file/d/1gIBNEF7jOfEWeRT3Ladfa_11Y24musbH/view?usp=drive_link

8. Conclusion

This project demonstrates the successful implementation of a **causally consistent distributed system** using **vector clocks**. All requirements are met:

- Vector Clock logic

- Causal write propagation and buffering
- Flask APIs
- Containerized multi-node setup
- Scenario-based validation with a client script

End of Report