

Project Report

Vector Clocks and Causal Ordering in Distributed Key- Value Store

Fundamental Distributed Systems

Vector Clocks and Causal Consistency in a Multi-Node Key-Value Store Technologies Used:

Python, Docker, Docker Compose

Submitted by: Rimpay Sharma

Roll No: G24AI2102

Date: 13/07/25

Github Repo: <https://github.com/Rimpay-sharma/fds>

1. Project Objective

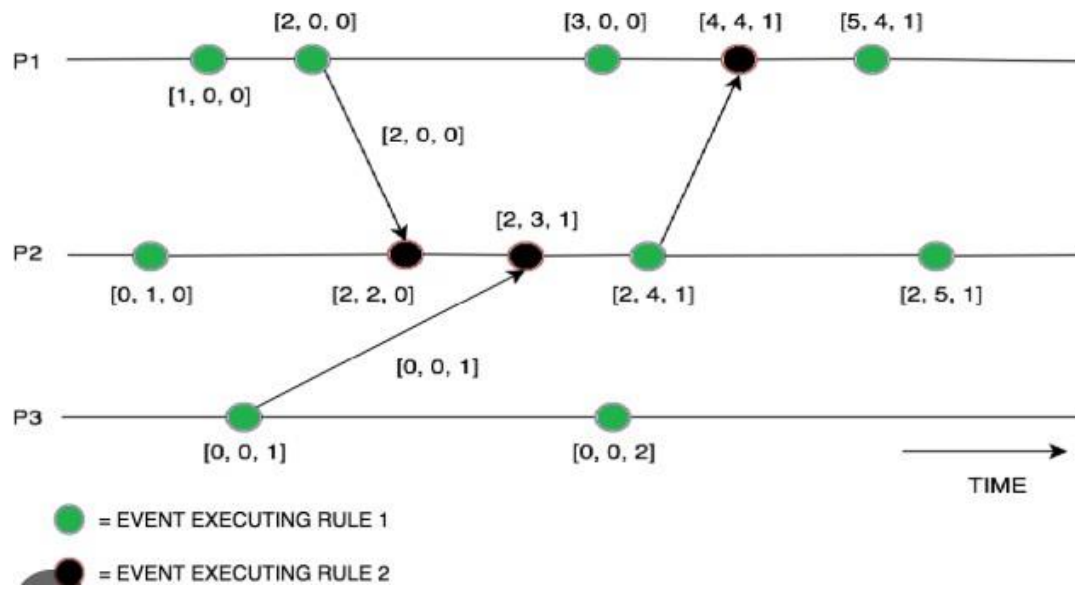
The goal of this project is to build a distributed key-value store with causal consistency using Vector Clocks. This ensures that operations across nodes respect causal relationships, preventing scenarios where dependent events are processed out of order.

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:



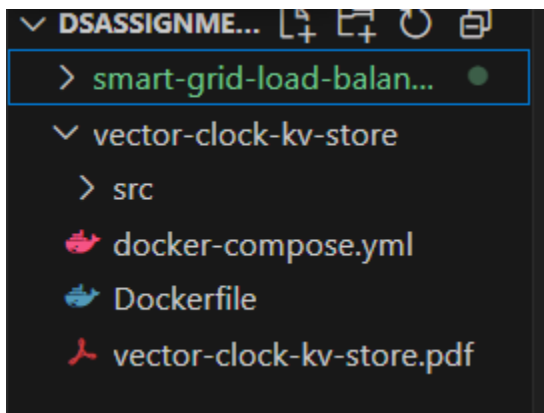
3. Technology Stack

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



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.

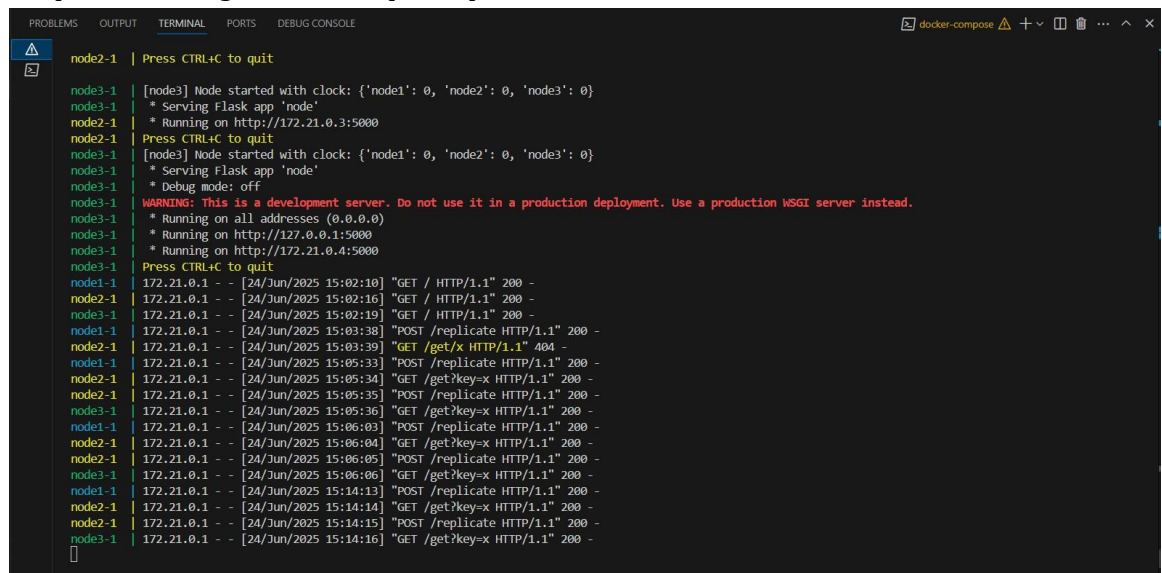
Client Script

- 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.

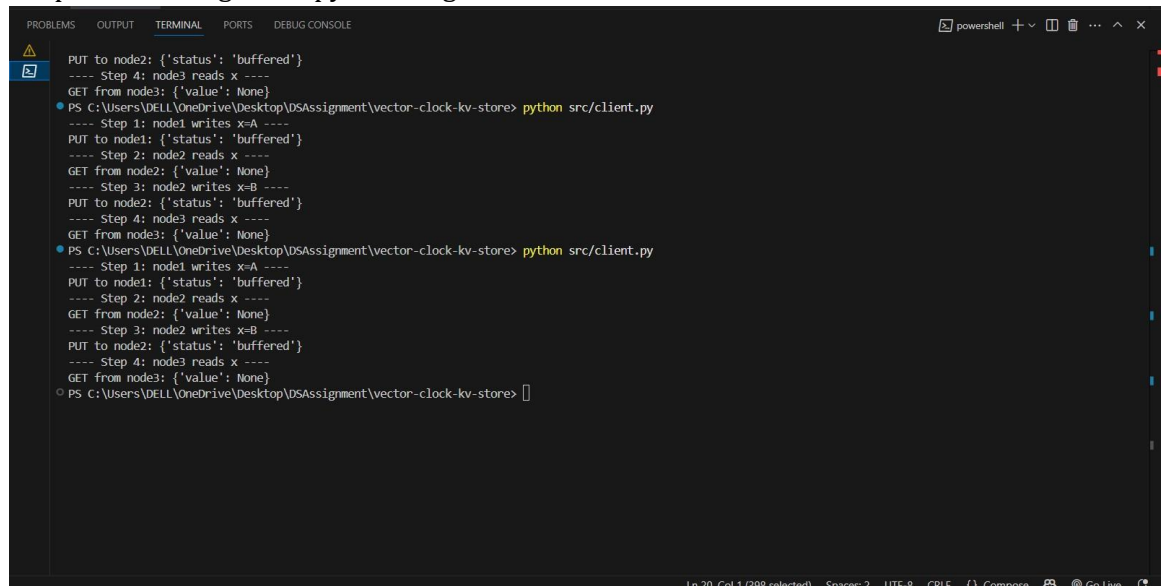
6. Screenshots

- Output of running docker-compose up



```
node2-1 | Press CTRL+C to quit
node3-1 | [node3] Node started with clock: {'node1': 0, 'node2': 0, 'node3': 0}
node3-1 | * Serving Flask app 'node'
node2-1 | * Running on http://172.21.0.3:5000
node2-1 | Press CTRL+C to quit
node3-1 | [node3] Node started with clock: {'node1': 0, 'node2': 0, 'node3': 0}
node3-1 | * Serving Flask app 'node'
node3-1 | * Debug mode: off
node3-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node3-1 | * Running on all addresses (0.0.0.0)
node3-1 | * Running on http://127.0.0.1:5000
node3-1 | * Running on http://172.21.0.4:5000
node3-1 | Press CTRL+C to quit
node1-1 | 172.21.0.1 - - [24/Jun/2025 15:02:10] "GET / HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:02:16] "GET / HTTP/1.1" 200 -
node3-1 | 172.21.0.1 - - [24/Jun/2025 15:02:19] "GET / HTTP/1.1" 200 -
node1-1 | 172.21.0.1 - - [24/Jun/2025 15:03:38] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:03:39] "GET /get/x HTTP/1.1" 404 -
node1-1 | 172.21.0.1 - - [24/Jun/2025 15:05:33] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:05:34] "GET /get?key=x HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:05:35] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.21.0.1 - - [24/Jun/2025 15:05:36] "GET /get?key=x HTTP/1.1" 200 -
node1-1 | 172.21.0.1 - - [24/Jun/2025 15:06:03] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:06:04] "GET /get?key=x HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:06:05] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.21.0.1 - - [24/Jun/2025 15:06:06] "GET /get?key=x HTTP/1.1" 200 -
node1-1 | 172.21.0.1 - - [24/Jun/2025 15:14:13] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:14:14] "GET /get?key=x HTTP/1.1" 200 -
node2-1 | 172.21.0.1 - - [24/Jun/2025 15:14:15] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.21.0.1 - - [24/Jun/2025 15:14:16] "GET /get?key=x HTTP/1.1" 200 -
```

- Output of running client.py showing causal correctness



```
PUT to node2: {'status': 'buffered'}
---- Step 4: node3 reads x ----
GET from node3: {'value': None}
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}
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}
PS C:\Users\DELL\OneDrive\Desktop\DSAssignment\vector-clock-kv-store>
```

- Node.py file

```
src > node.py > VectorClock > is_causally_ready
1 from flask import Flask, request
2 import threading
3 import time
4 import sys
5
6 # ----- VectorClock Class -----
7
8 class VectorClock:
9     def __init__(self, node_id, all_nodes):
10         self.clock = {nid: 0 for nid in all_nodes}
11         self.node_id = node_id
12
13     def increment(self):
14         self.clock[self.node_id] += 1
15
16     def update(self, received_clock):
17         for node, val in received_clock.items():
18             self.clock[node] = max(self.clock.get(node, 0), val)
19
20     def is_causally_ready(self, received_clock, sender_id):
21         for node in self.clock:
22             if node == sender_id:
23                 if received_clock[node] != self.clock[node] + 1:
24                     return False
25             else:
26                 if received_clock[node] > self.clock[node]:
27                     return False
28         return True
29
30     def get_clock(self):
31         return self.clock.copy()
32
33 # ----- Globals -----
34
35 app = Flask(__name__)
36 store = {} # key-value data store
37 buffer = [] # buffer for causally premature messages
```

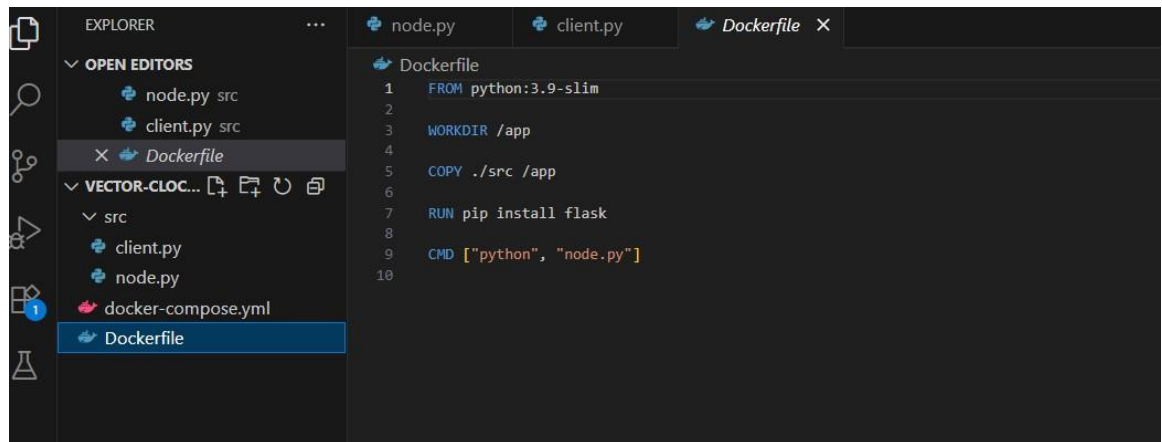
```
src > node.py > VectorClock > is_causally_ready
34
35 app = Flask(__name__)
36 store = {} # key-value data store
37 buffer = [] # buffer for causally premature messages
38 node_id = None # Current node's ID
39 vector_clock = None # VectorClock instance
40 all_nodes = [] # All node IDs
41
42 # ----- Flask Endpoints -----
43
44 @app.route('/')
45 def index():
46     return f'Node {node_id} is running with clock: {vector_clock.clock}', 200
47
48 @app.route('/put', methods=['POST'])
49 def put():
50     global store, vector_clock
51     data = request.get_json()
52     key = data['key']
53     value = data['value']
54     received_clock = data['clock']
55     sender_id = data['sender']
56
57     if vector_clock.is_causally_ready(received_clock, sender_id):
58         store[key] = value
59         vector_clock.update(received_clock)
60         print(f'[{node_id}] Applied write: {key}={value}, clock={vector_clock.clock}')
61         return {'status': 'applied'}
62     else:
63         buffer.append(data)
64         print(f'[{node_id}] Buffered write: {key}={value} from {sender_id}')
65         return {'status': 'buffered'}
66
67 @app.route('/get', methods=['GET'])
68 def get():
69     key = request.args.get('key')
70     value = store.get(key, None)
```

```
66 @app.route('/get', methods=['GET'])
67 def get():
68     key = request.args.get('key')
69     value = store.get(key, None)
70     print(f"[{node_id}] get: (key)={value}")
71     return {'value': value}
72
73 @app.route('/replicate', methods=['POST'])
74 def replicate():
75     return put()
76
77 # ----- Background Buffer Processing -----
78
79 def process_buffer():
80     global buffer
81     while True:
82         time.sleep(0.5)
83         for entry in buffer[:]: # Iterate over a copy
84             if vector_clock.is_causally_ready(entry['clock'], entry['sender']):
85                 store[entry['key']] = entry['value']
86                 vector_clock.update(entry['clock'])
87                 buffer.remove(entry)
88                 print(f"[{node_id}] Buffered write applied: (entry['key'])={entry['value']}")
89
90 # ----- Node Initialization -----
91
92 def start_node(my_id, node_list):
93     global node_id, all_nodes, vector_clock
94     node_id = my_id
95     all_nodes = node_list
96     vector_clock = VectorClock(node_id, all_nodes)
97
98     print(f"[{node_id}] Node started with clock: {vector_clock.clock}")
99     threading.Thread(target=process_buffer, daemon=True).start()
100     app.run(host='0.0.0.0', port=5000)
101
102 # ----- Entry Point -----
103
104 if __name__ == '__main__':
105     my_node_id = sys.argv[1] # e.g., "node1"
106     node_ids = sys.argv[2].split(',') # e.g., "node1,node2,node3"
107     start_node(my_node_id, node_ids)
```

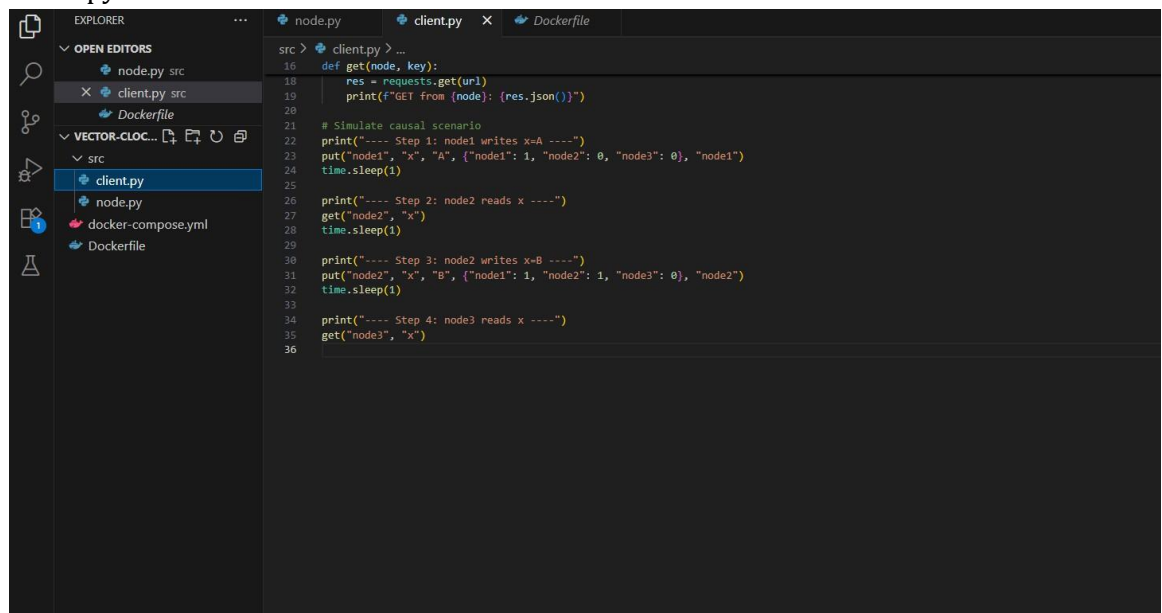
- Docker-compose.yml

```
1 version: '3'
2 services:
3   node1:
4     build: .
5     ports:
6       - "5001:5000"
7     command: ["python", "node.py", "node1", "node1,node2,node3"]
8
9   node2:
10    build: .
11    ports:
12      - "5002:5000"
13    command: ["python", "node.py", "node2", "node1,node2,node3"]
14
15   node3:
16    build: .
17    ports:
18      - "5003:5000"
19    command: ["python", "node.py", "node3", "node1,node2,node3"]
20
```

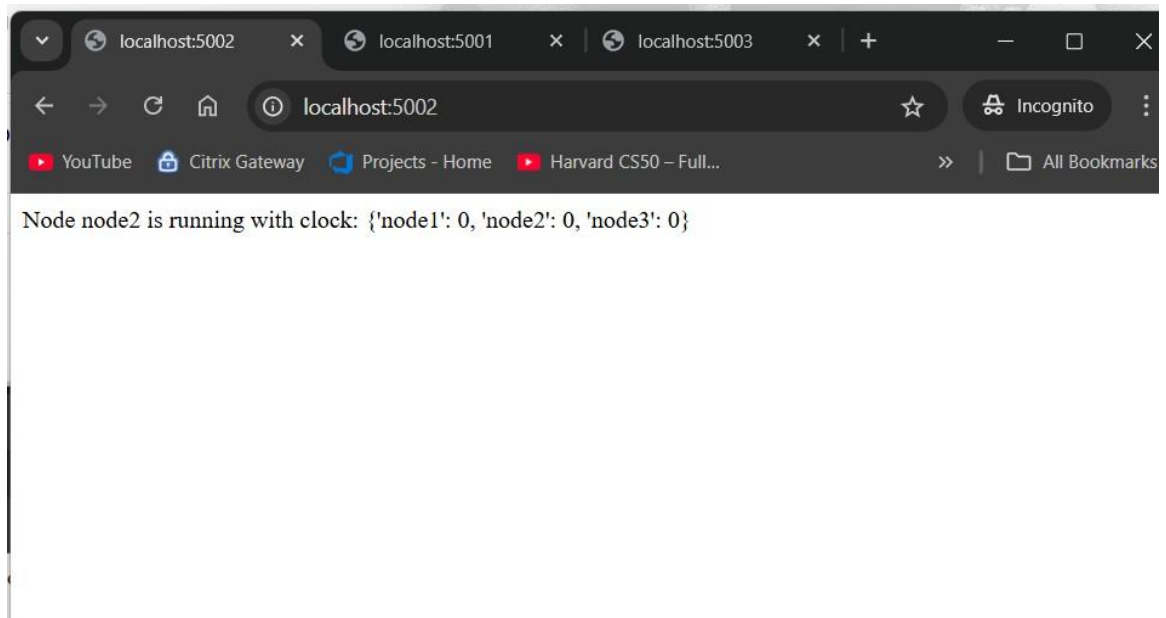
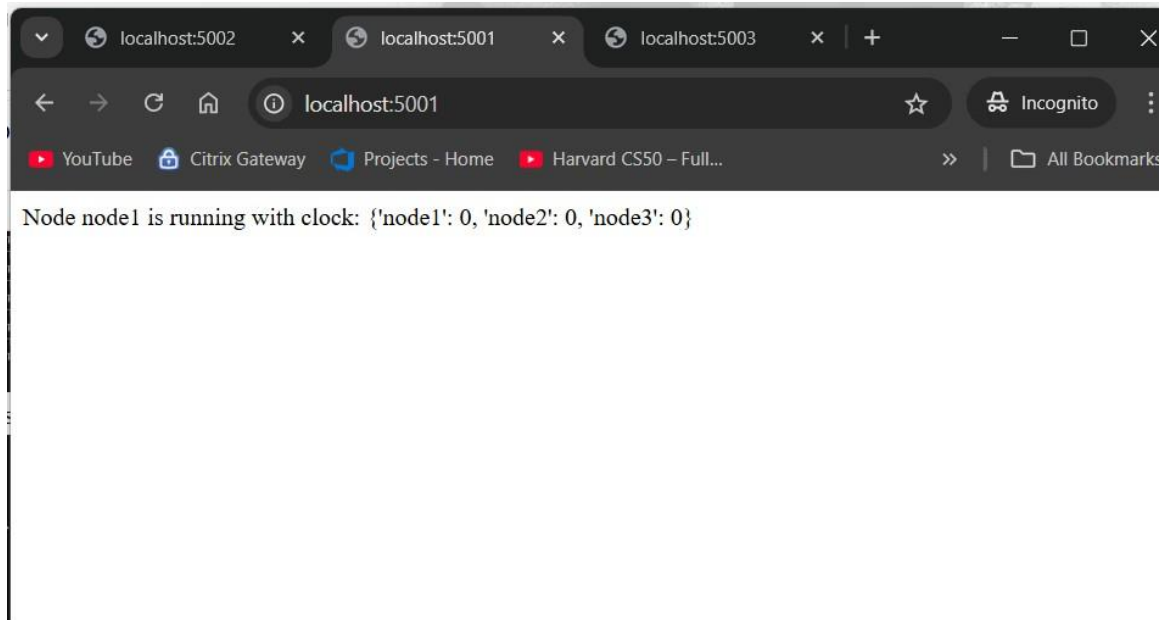
- Dockerfile

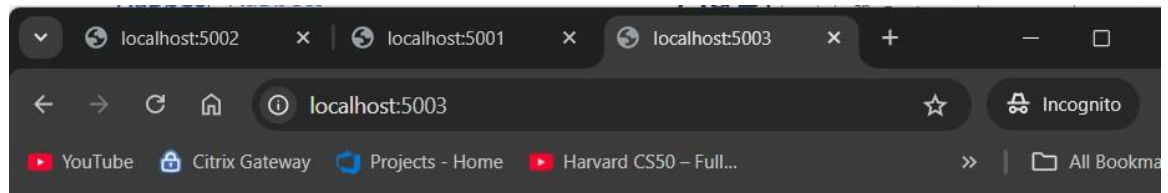


- Client.py



- browser response for node status





Node node3 is running with clock: {'node1': 0, 'node2': 0, 'node3': 0}

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. 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