

**Submitted By:**

Anusha V

G24AI2042

**1. Project Title**

Vector Clocks and Causal Ordering

**2. Objective**

To design and implement a distributed key-value store with causal consistency using vector clocks across three containerized nodes in Docker.

**3. Technologies Used**

• **Programming Language**: Python 3

• **Networking:** Flask (REST API)

• **Concurrency:** Threading for buffer monitoring

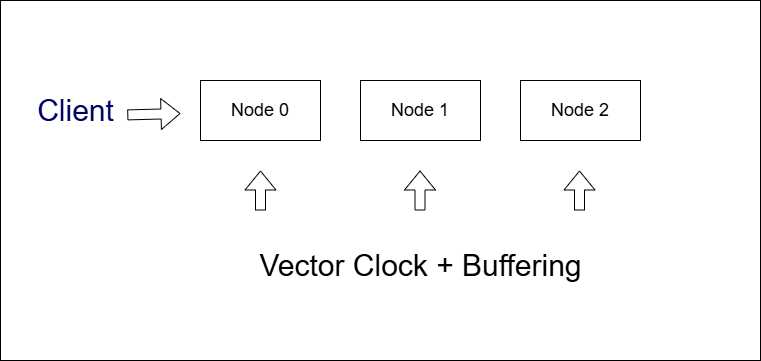
• **Containerization:** Docker, Docker Compose

**4. Architecture**

Each node does the following:

* Maintains a local key-value store
* Tracks causal history using a vector clock
* Forwards updates to peer nodes
* Buffers updates until their causal dependencies are satisfied

Diagram:



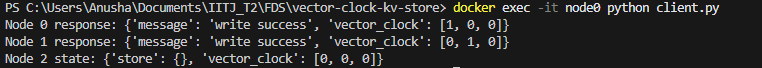
**5. Implementation Highlights**

* Vector clock is incremented on local writes
* Updates are sent to all other nodes with the current vector clock
* Nodes receiving updates compare vector clocks
* If dependencies met → apply immediately
* If not → buffer until ready

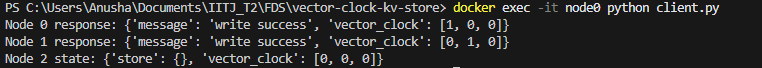
**6. Log Output and Explanation**

Example test:

* Input Command:



* Output:

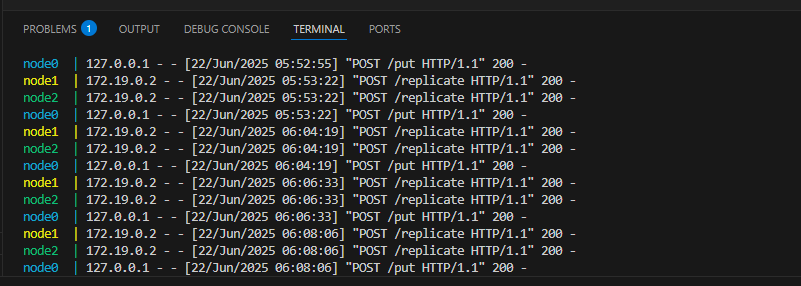


**Explanation:**

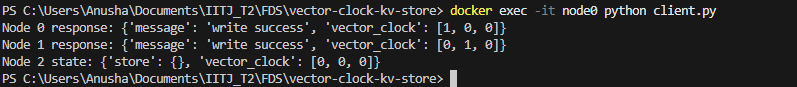
* Node 0 and 1 executed local writes
* Node 2 hasn’t applied either due to unmet causal dependencies
* This confirms buffering + vector comparison logic is correct

**7. Screenshots**

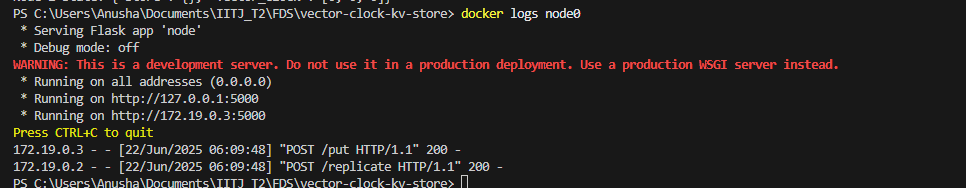
**Image 1:** All 3 Containers running successfully

****

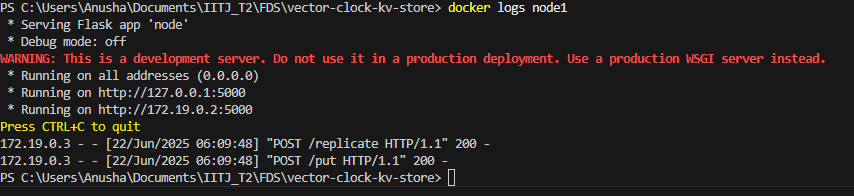
**Image 2:** Proof of buffering + vector comparison logic is correct as Node 0 and 1 executed local writes and Node 2 hasn’t applied either due to unmet causal dependencies.



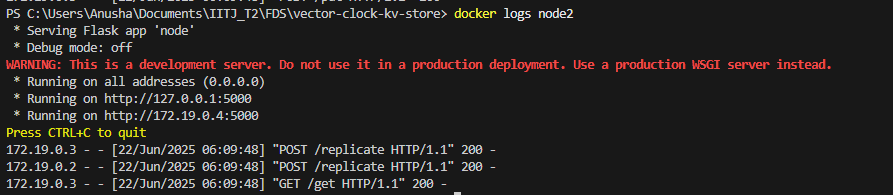
**Image 3:** Node 0 logs show: local write + replication



**Image 4:** Node 1 logs show: receiving a replicated message from node0



**Image 5:** Node 2 logs show: buffered state and eventually applied updates



**8. Demo Video Link**

<https://youtu.be/obOttZV3o_A>

**9. Conclusion**

This project successfully implements a causally consistent key-value store using vector clocks. Buffered replication ensures no event is applied until its causal history is present, demonstrating proper causal ordering in a distributed system.