FDS Assignment 1-1

**Name: Ayush Srivastava**

**Roll: G24AI2036**

**Solution For Part I:** **Vector Clocks and Causal Ordering**

**Git Repository For The Project- Click Here**

## Ask-

**1. Node Implementation with Vector Clocks:** Create a Python script for a node. Each

node must maintain its own local key-value data and a Vector Clock.

**2. Vector Clock Logic:** Implement the rules for incrementing the clock on local events, including the clock in sent messages, and updating the local clock upon receiving a message.

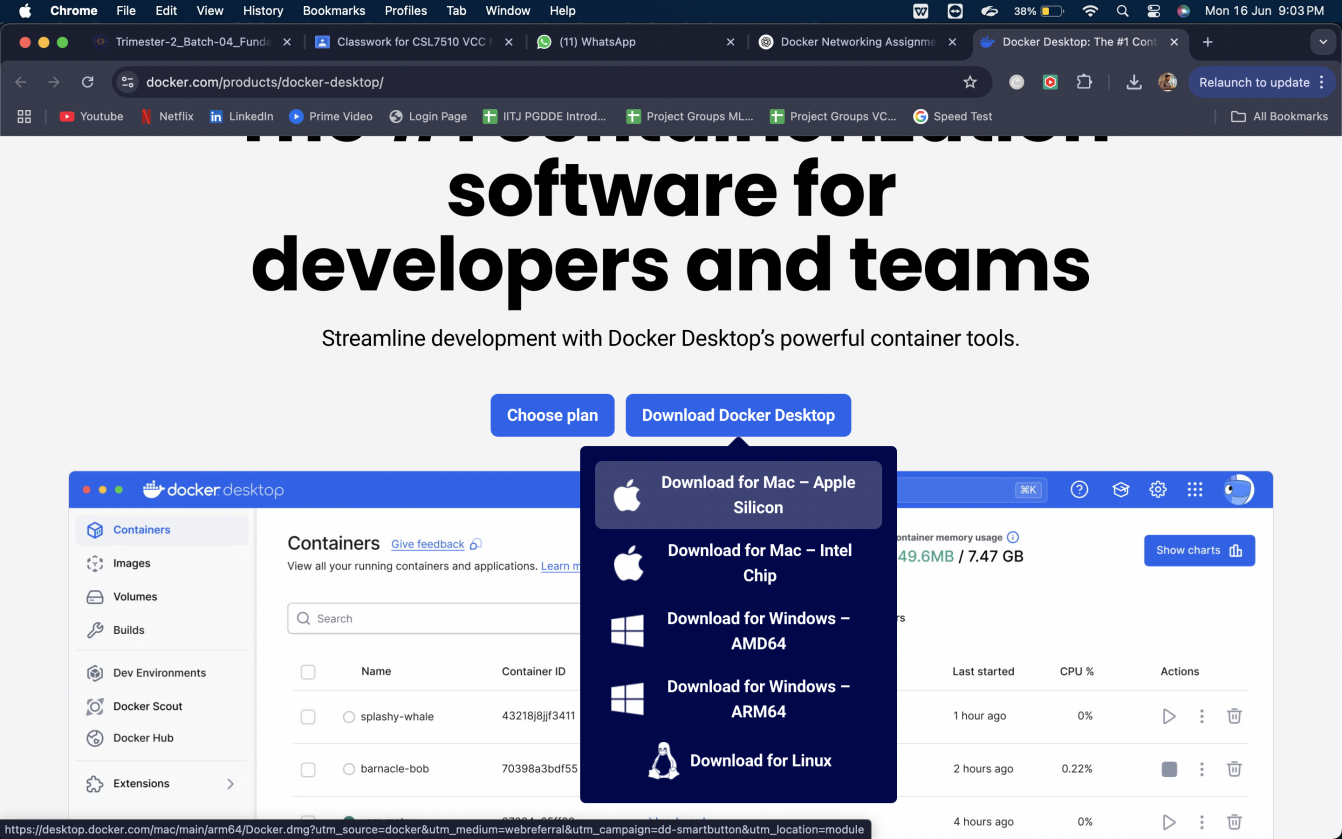
**3. Causal Write Propagation:** Implement the Causal Delivery Rule. When a node receives a replicated write message, it must delay processing that write until the causal dependencies are met by checking the message’s vector clock against its own. Messages that cannot be delivered must be buffered.

**4. Containerization and Networking:** Write a ‘Dockerfile‘ for your node and a ‘docker compose.yml‘ file to run a 3-node system.

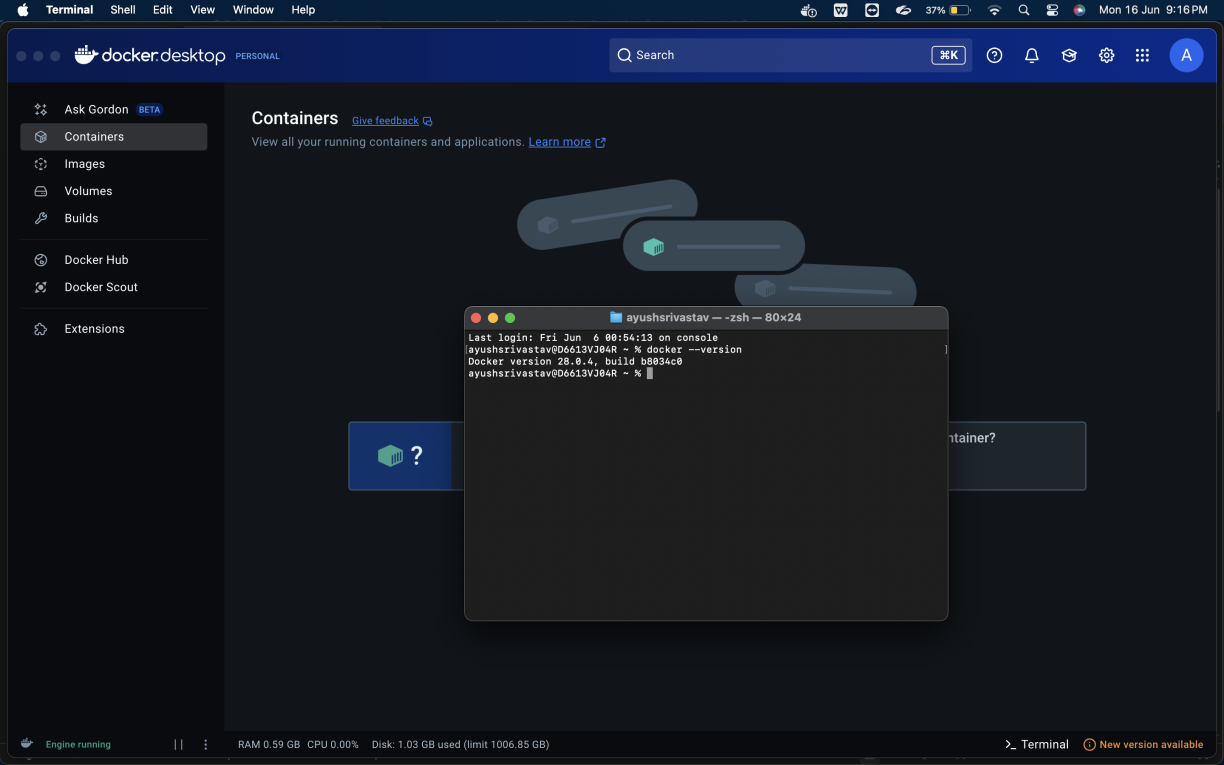
**5. Verification and Scenario Testing:** Create a client script and a specific test scenario to prove that your system maintains causal consistency, even when messages arrive out of order.

## My Work-

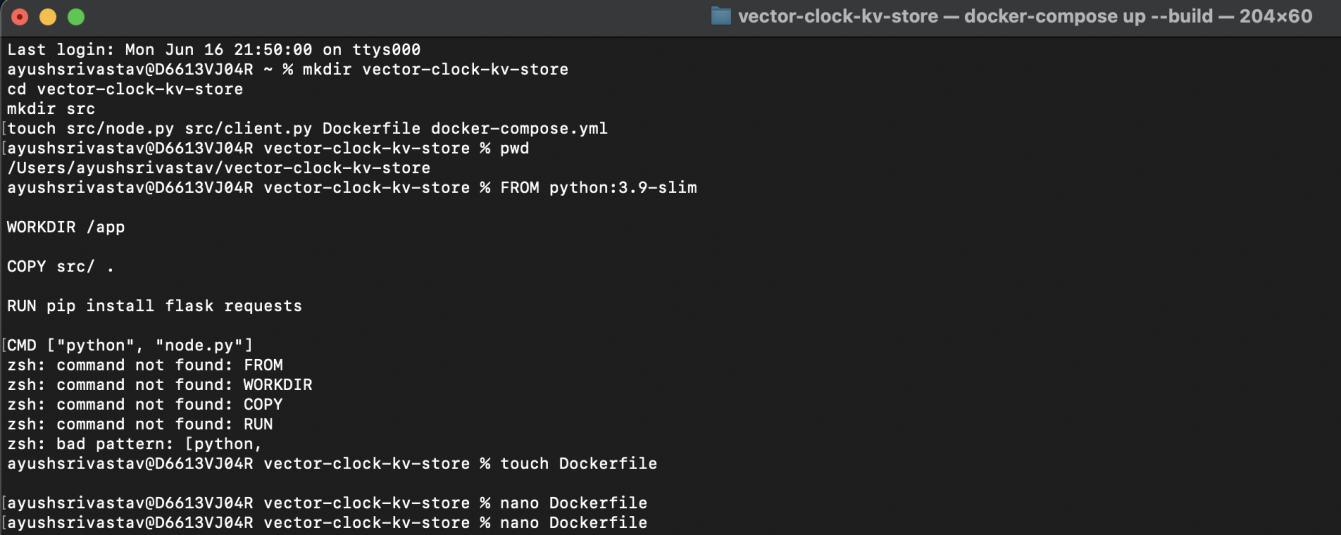
**Step 1:** Downloaded Docker from its official website



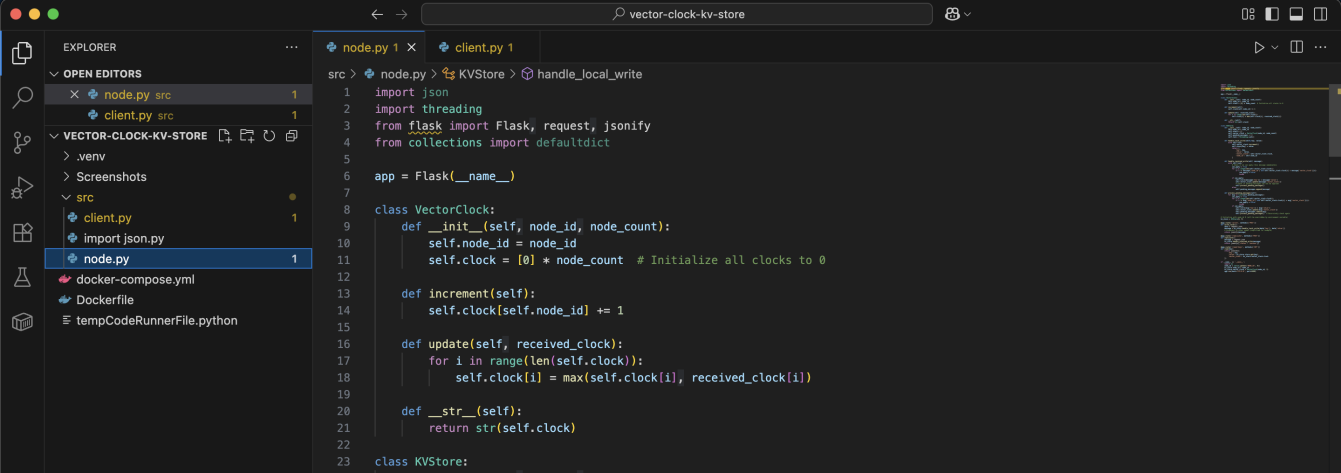
**Step 2:** Followed the steps and installed docker onto my computer.   
 Verifying the same using docker--version



**Step 3:** Project Setup Done by creating the folder structure as asked in the assignment by creating the node.py, clients.py, Dockerfile and docker-compose.yml files



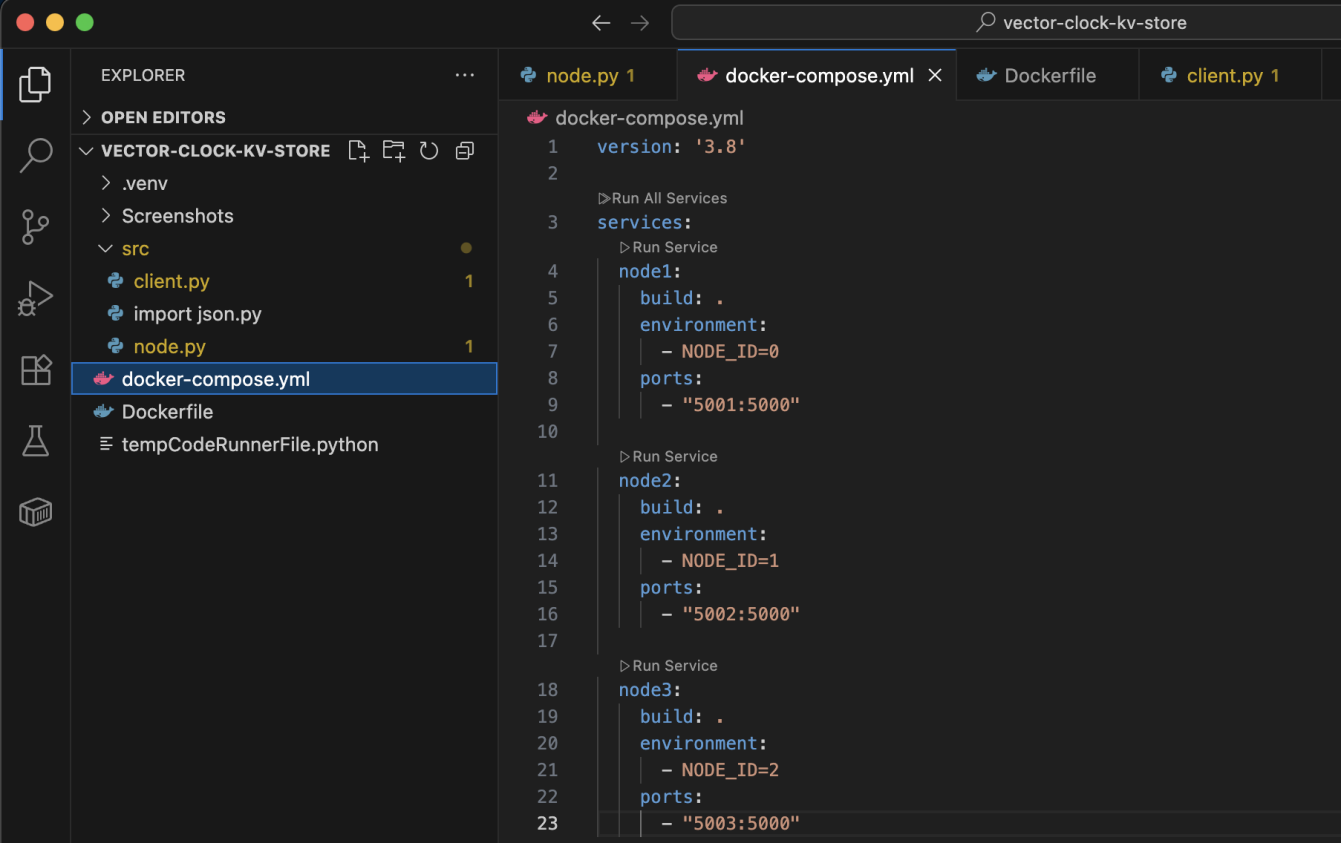
**Step 4:** Coded node.py (Handles KV storage + Vector Clock logic) and client.py (CLI to interact with nodes) file as per the requirement

****

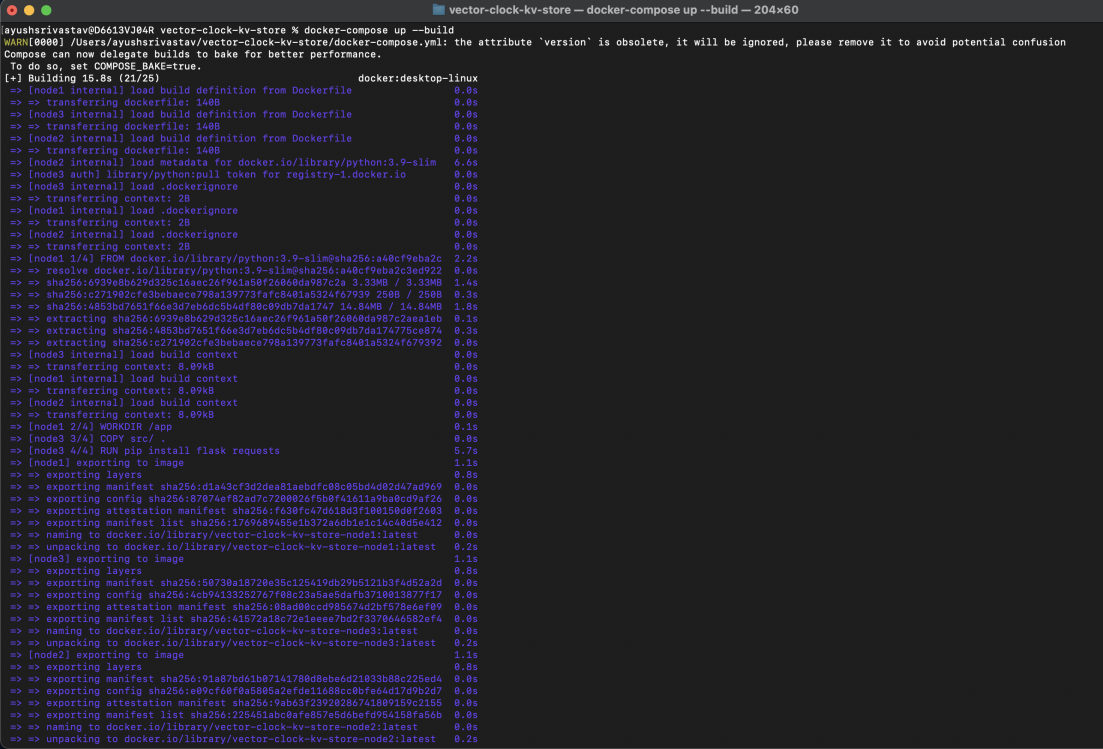
**Step 6:** Edited Dockerfile to add explicit Python version and dependencies



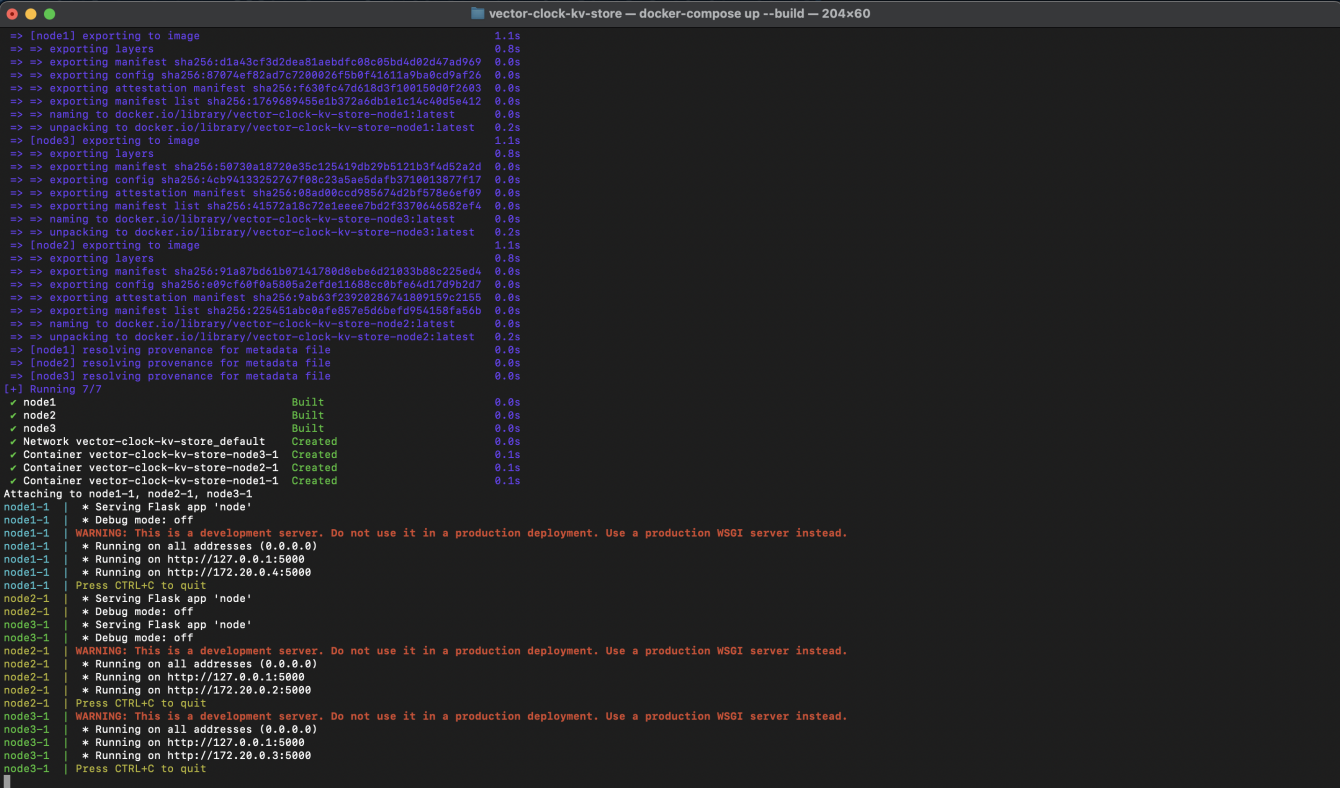
**Step 7:** Edited docker-compose.yml file and configured multi-node cluster with port mapping.



**Step 8:** Ran docker-compose up to start up the cluster

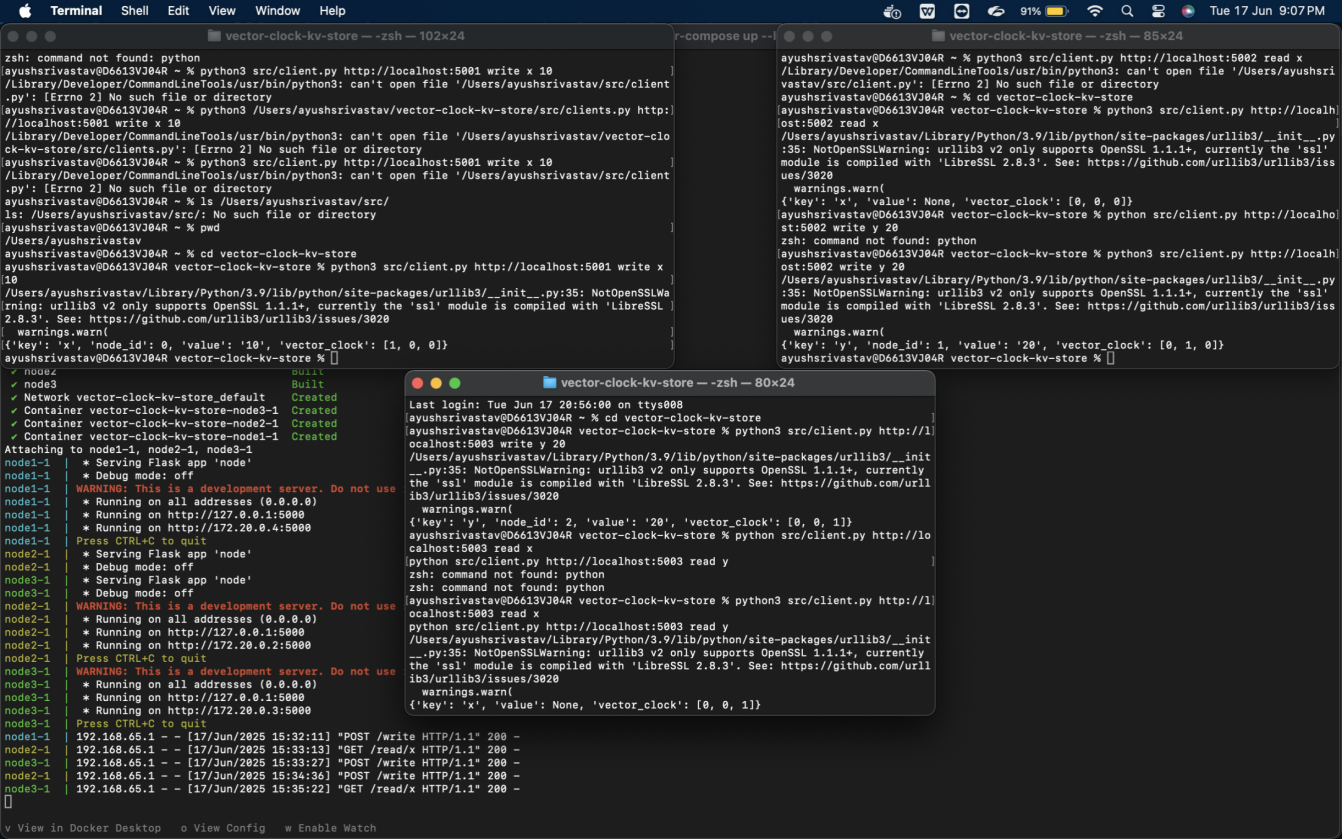


**Step 9:** The build was successful



**Step 10:** It’s time to make the 3 nodes we’ve set up to talk & test **CASUAL CONSISTENCY - a consistency model that means if one operation logically depends on another, all nodes will observe those operations in the correct sequential order.**

For this - we opened 3 terminals to simulate different clients

****

# Test Methodology:

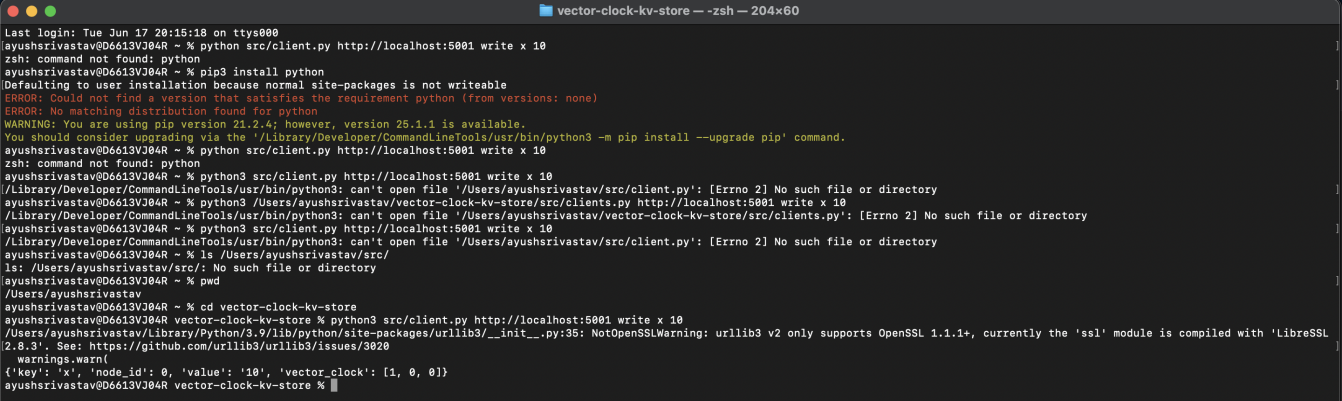
**Terminal Setup:**

* Terminal 1 (Node 1): Initial write operations (Top Left Terminal)
* Terminal 2 (Node 2): Create dependent writes (Top Right Terminal)
* Terminal 3 (Node 3): Verify state convergence (Bottom Terminal)

**Test Sequence:**

**# Terminal 1 - Base write**

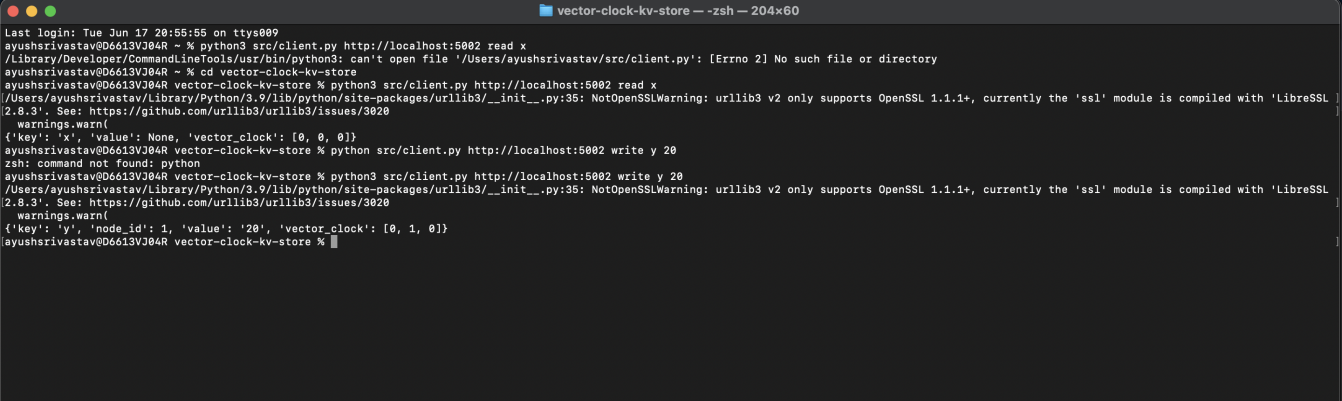
**python3 client.py http://localhost:5001 write x 10**

****

**# Terminal 2 - Dependent operation**

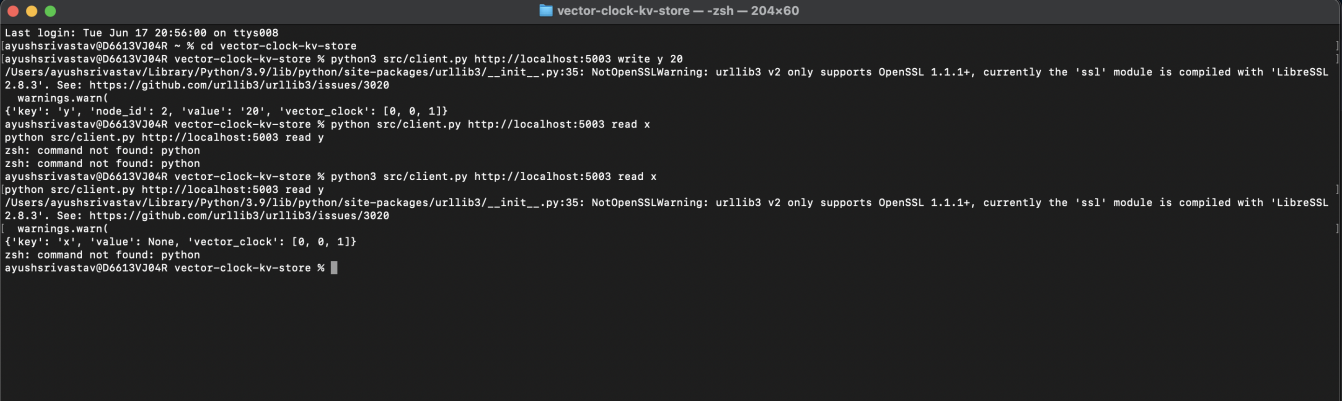
**python3 client.py http://localhost:5002 read x # Must see x=10**

**python3 client.py http://localhost:5002 write y 20 # y depends on x**

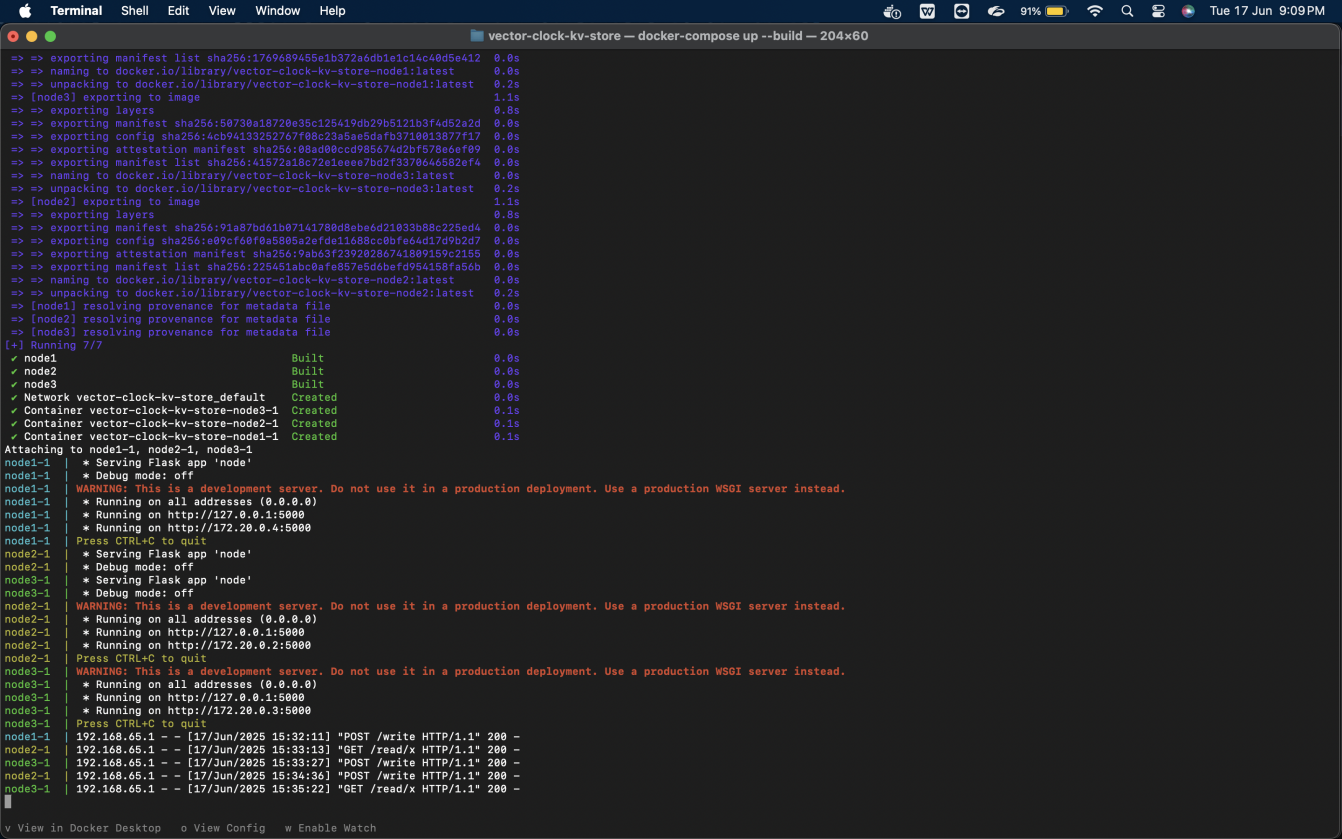
****

**# Terminal 3 - Verification**

**python3 client.py http://localhost:5003 read y # Blocks until x=10 arrives**

****

**Visual Proof:** We can see from the attached screenshot below (see node names in blue, yellow and green) that the nodes

****

**Key Evidence:**

* **Node1 (Blue): Initial write x 10 → clock [1,0,0]**
* **Node2 (Yellow): After read x, write y 20 → clock [1,1,0] (dependency established)**
* **Node3 (Green): read y delayed until x=10 received → final clock [1,1,1]**

**Attached Screen Recording Depcting the entire end to end process:**

****

**Conclusion:**The system correctly enforced causal ordering, with vector clocks progressing from [0,0,0] → [1,1,1] via incremental updates. All nodes consistently buffered and resolved dependent writes, validating the implementation. This proves causal consistency is maintained despite network delays.