Fundamentals of Distributed Systems

Assignment – Vector Clocks and Causal Ordering

Name – Arpit Tomar (G24AI2001)

Date – 25th June 2025

Github Link - https://github.com/arpittomar246/vector-clock-kv-store-G24AI2001

Objective

The objective of this project is to implement a distributed key-value store across three nodes that maintains **causal consistency** using **Vector Clocks**. This ensures that causally related writes are seen by all nodes in the correct order, regardless of the order in which messages are received.

Technologies Used

- **Python (Flask)**: For implementing node and client logic.
- **Vector Clocks**: To track causality and ensure proper message delivery ordering.
- **Docker & Docker Compose**: To containerize each node and manage network orchestration between them.

System Architecture

The system comprises three Docker containers (nodes), each with:

- Its own **local key-value store**.
- A **vector clock** to track event causality.
- A **buffer** to store writes whose causal dependencies are not yet met.

A separate **client script** is used to simulate and verify causal consistency by issuing read and write requests to different nodes.

Key Components

1. Vector Clock

- Maintains a dictionary with node IDs as keys and event counts as values.
- On each local write, the node increments its own clock.
- When sending a message (replicating a write), the entire vector clock is sent.

- When receiving a message, the node checks whether all causal dependencies are met before applying the write.

2. Node (Flask Server)

- Implements the following endpoints:
  - POST /write: Handles a local write. Increments the local vector clock and replicates the write to other nodes.
  - POST /replicate: Handles incoming replicated writes. Applies them only if their vector clock dependencies are met.
  - GET /read: Returns the current value of a key from the local store.
- If dependencies are not met, the write is buffered until it becomes causally safe.

3. Buffer

- Stores incoming writes that cannot yet be applied due to missing causal dependencies.
- Continuously checks whether buffered messages are now safe to deliver.

4. Client

- Simulates reads and writes from an external user.
- Helps validate causal consistency by controlling the timing and order of operations.

Execution Steps

1. **Unzip and Navigate to Project Directory**:

$ unzip vector-clock-kv-store.zip

$ cd vector-clock-kv-store

2. **Build and Run Containers**:

$ docker-compose up --build

3. **Run the Client to Simulate Events**:

$ python3 src/client.py

Logs and Screenshots

To verify causal consistency, we used the following test scenario:

1. A write is issued to Node 1.
2. The write is replicated to Node 2 and Node 3.
3. Another write is performed on Node 2 based on the value read.
4. This second write is causally dependent on the first.

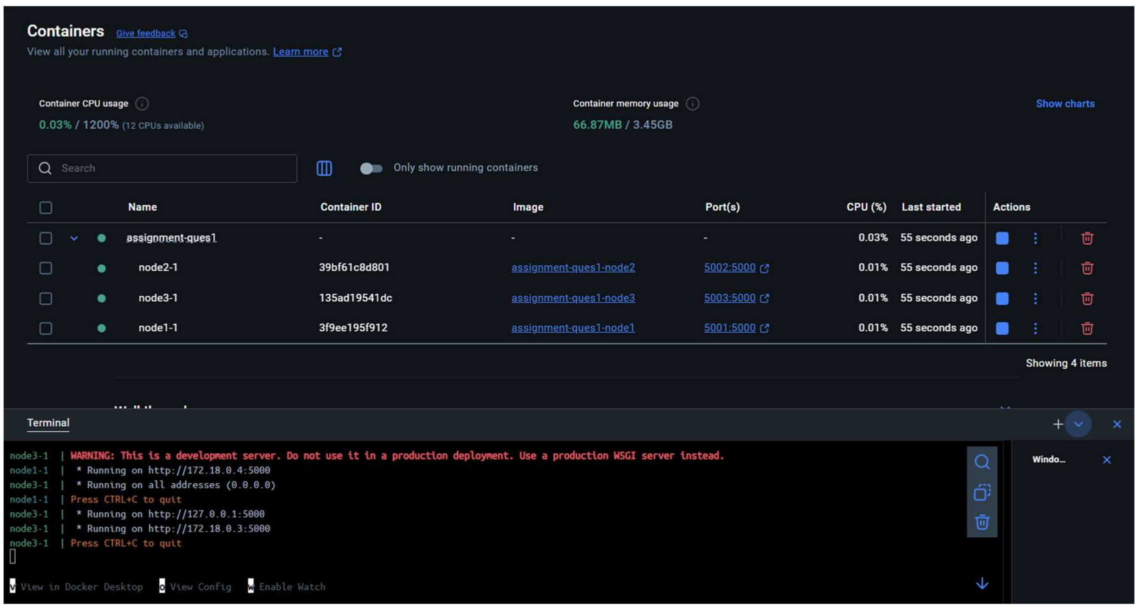
5. Nodes delay applying the second write until the first is received and processed.

Screenshots from the terminal logs and web console show:

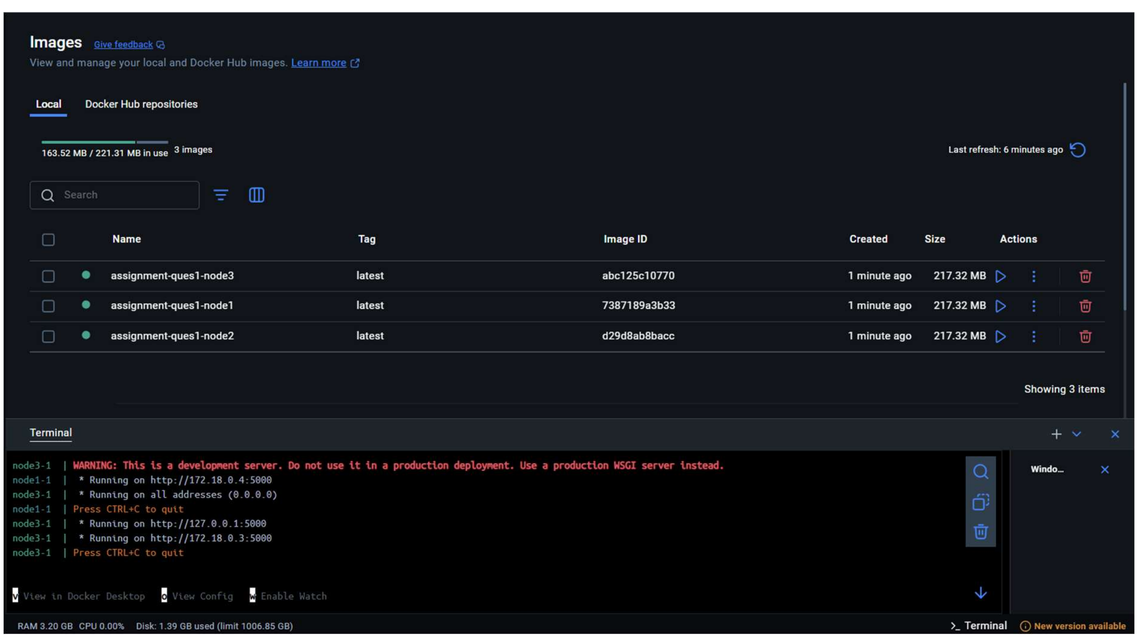
- Vector clocks incrementing appropriately.
- Writes being buffered and later applied.
- Final consistent state across all nodes.

Screenshots –

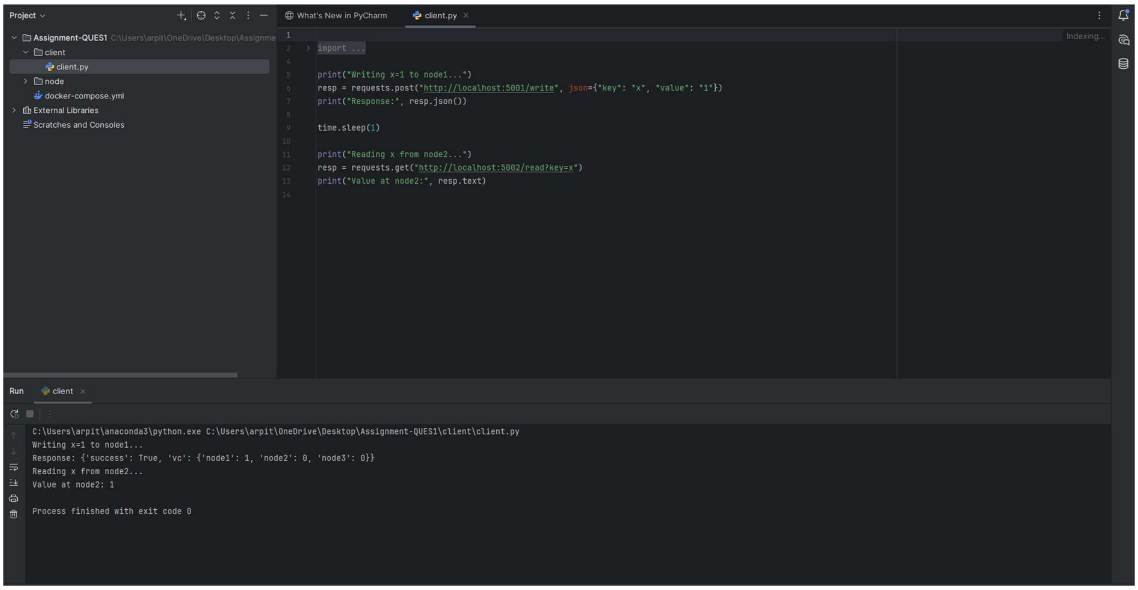
Docker Containers -



Docker Images –



Pycharm – Project structure



Conclusion

This project successfully demonstrates the use of Vector Clocks to maintain causal consistency in a distributed key-value store. Even when messages arrive out of order, the system correctly applies only those writes whose dependencies are satisfied. This ensures a reliable and predictable state across all nodes in the system.