**Project Report:**

**Vector Clock-Based Causal Key-Value Store**

**Name**: Aastha Singh  
**Roll Number**: G24AI2079

**Objective**

The goal of this project is to implement a causally consistent distributed key-value store using vector clocks. The system ensures that updates across different nodes are applied in an order that respects causal relationships between events.

**Technology Stack**

* **Language**: Python
* **Framework**: Flask (for HTTP APIs)
* **Containerization**: Docker
* **Orchestration**: Docker Compose
* **Testing**: Python client script

**System Architecture**

* The system consists of three nodes.
* Each node maintains:
  + A key-value store
  + A vector clock of size 3
  + A buffer to hold out-of-order messages
* Nodes communicate over REST APIs using Flask.
* Each node can receive read and write requests.
* On a write, the node increments its own clock, updates the value, and broadcasts the update along with its vector clock to other nodes.

**Vector Clock Logic**

* When a local write happens, the node increments its own clock index.
* Every message (replication) contains the sender's vector clock.
* Upon receiving a message:
  + The node checks if it can be safely delivered (using vector clock comparison).
  + If not, the message is buffered.
* A background thread keeps checking the buffer for deliverable messages.

**Docker & Execution**

**Folder Structure:**

vector-clock-kv-store/

├── src/

│ ├── node.py

│ └── client.py

├── Dockerfile

├── docker-compose.yml

└── project\_report.pdf

**Commands:**

1. To build and run the 3-node system:

docker-compose up --build

1. Each container asks for node ID input (0, 1, or 2).
2. In a new terminal, run:

python src/client.py

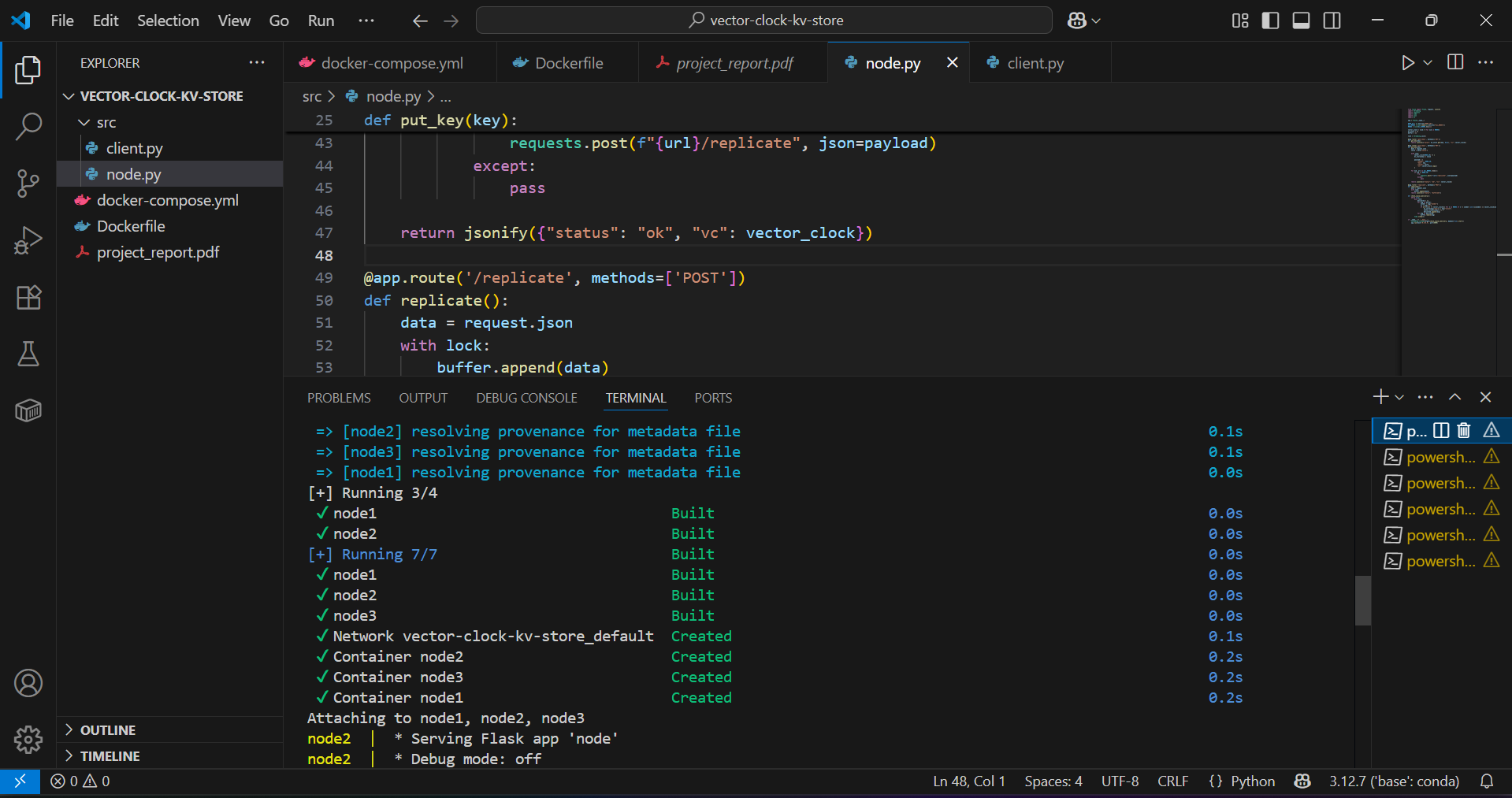
**Test Scenario**

1. Write key "x" with value "A" from Node 0.
2. Read the key "x" from Node 1.
3. Write a new value "B" to the same key from Node 1 (causally dependent).
4. On Node 2, the system ensures "A" is delivered before "B", even if "B" arrived first.

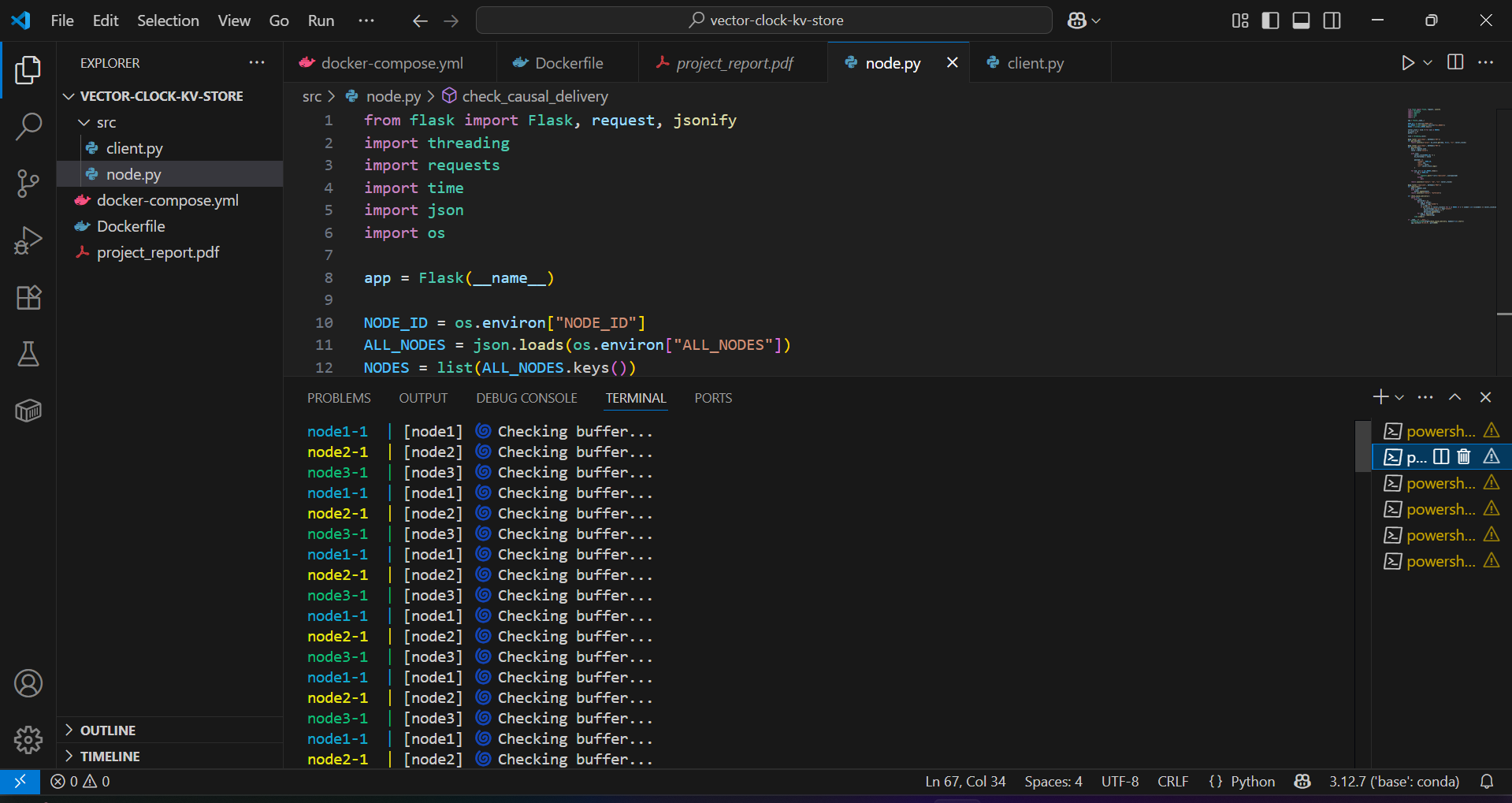
This confirms that causal consistency is maintained using vector clocks and buffered delivery.

**Screenshots**:

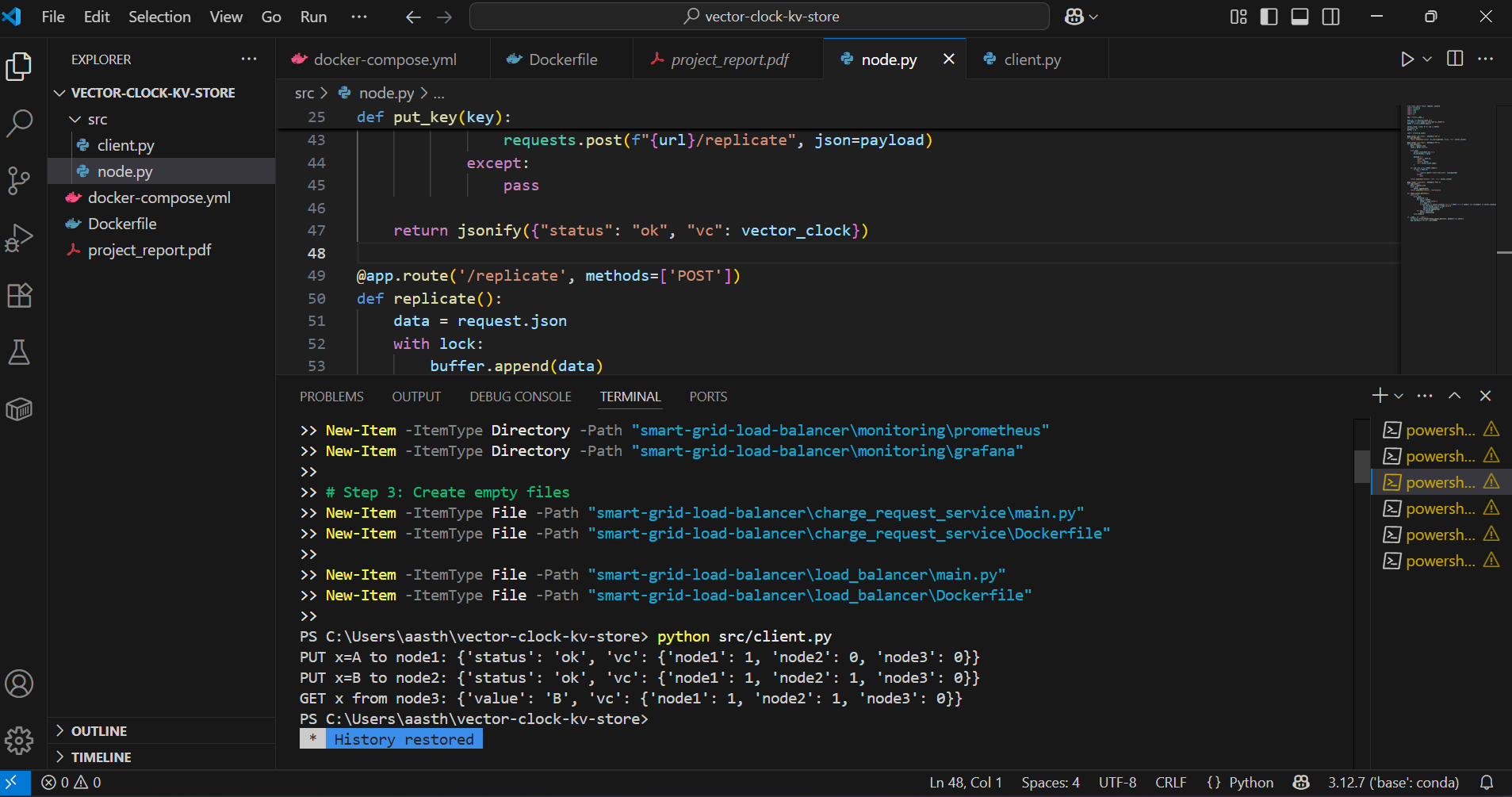
* Terminal outputs of all three nodes



* A case where a message is buffered and later delivered



* Final vector clock values for verification



**Conclusion**

This project successfully demonstrates how vector clocks can enforce causal ordering in a distributed key-value store. It gave hands-on experience with key distributed system concepts like event ordering, replication, and consistency models.