ECE 473/573 Fall 2023 - Project 3
Distributed Database Systems

Samman Chouhan
A20561414

II. Cassandra Cluster Management

Question 1: What happens if one misconfigured the CASSANDRA_CLUSTER_NAME variable in docker-compose.yml?

Misconfiguring variables in docker-compose.yml can cause complications in Cassandra cluster setup. The field names the Cassandra cluster to which each node belongs. Misconfiguring this variable can cause nodes to join several clusters, leading to data inconsistency and communication issues within the cluster. Ensure that all Cassandra nodes share the same cluster name.

Question 2: What is the purpose of the CASSANDRA_SEEDS variable in docker-compose.yml? Is it necessary to list all the servers?

The seed variable in docker-compose.yml provides the seed nodes in the Cassandra cluster. Seed nodes control cluster membership and facilitate node discovery during bootstrapping. It's important to include a few reliable servers in the variable, rather than listing all of them.
Seed nodes initiate clusters and serve as interaction points for new nodes. Having a subset of nodes as seed nodes improves both stability and performance. It's advised to have two or three seed nodes, however this can vary based on the cluster's size and requirements.

Question 3: What does the UN mean for the first column of the output from nodetool status?

The "UN" in the nodetool status report indicates "Up and Normal". This status confirms that the Cassandra node is operational inside the cluster. Nodes designated as "UN" are actively participating in the cluster and can handle read and write requests. A healthy Cassandra cluster typically has nodes with the "UN" status. This indicates that they are operational and contribute to data storage and processing duties within the cluster.

Question 4: In our cluster consisting of containers, where does Cassandra store data?

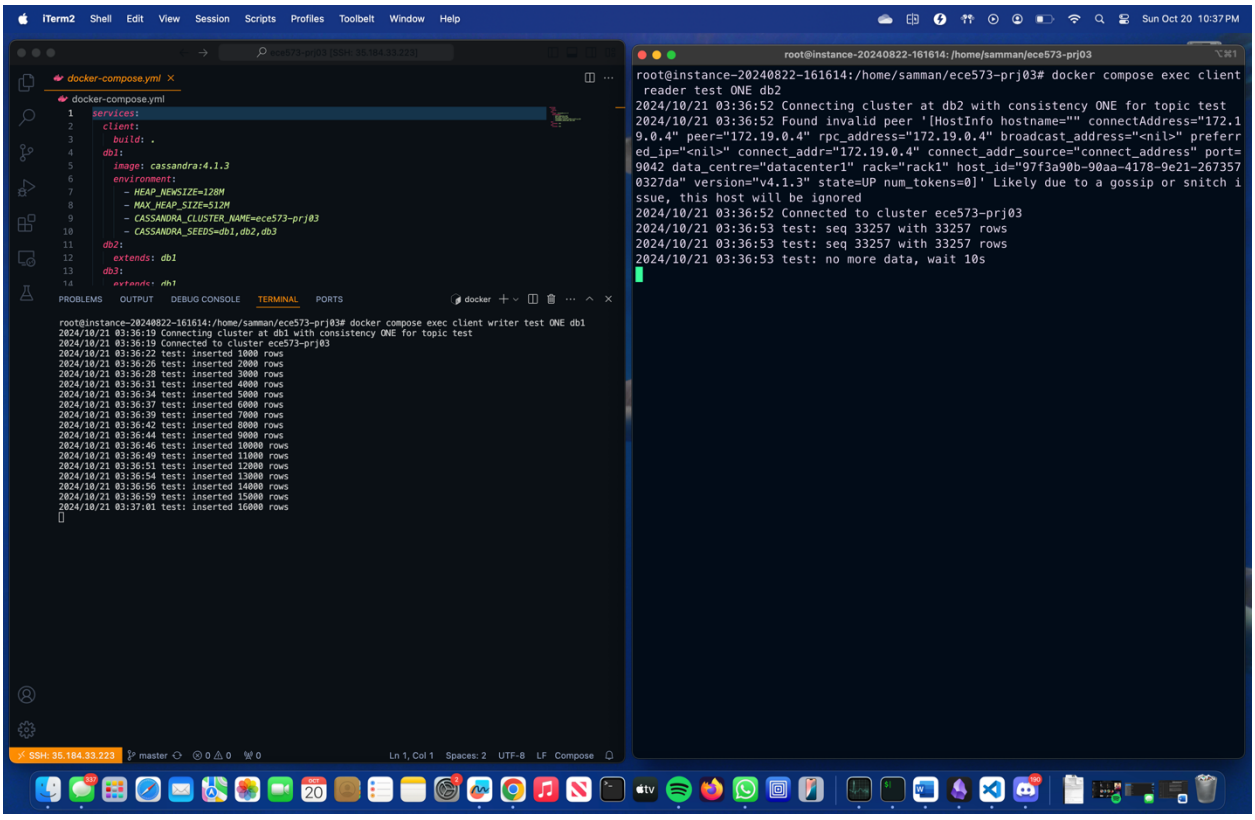
Cassandra stores data in node-specific directories within a container cluster. Cassandra nodes run in independent containers (e.g., db1, db2, db3), each with its own data directory for managing keyspaces and tables.
Cassandra's data directory is commonly set in its configuration files (cassandra.yaml). In a Dockerized Cassandra cluster, data directories are often located on the container's filesystem. When using containers, make sure the data directories are properly mapped to the host machine or external storage to preserve data even if the container is terminated or deleted. This enables data to survive container restarts or re-creations.

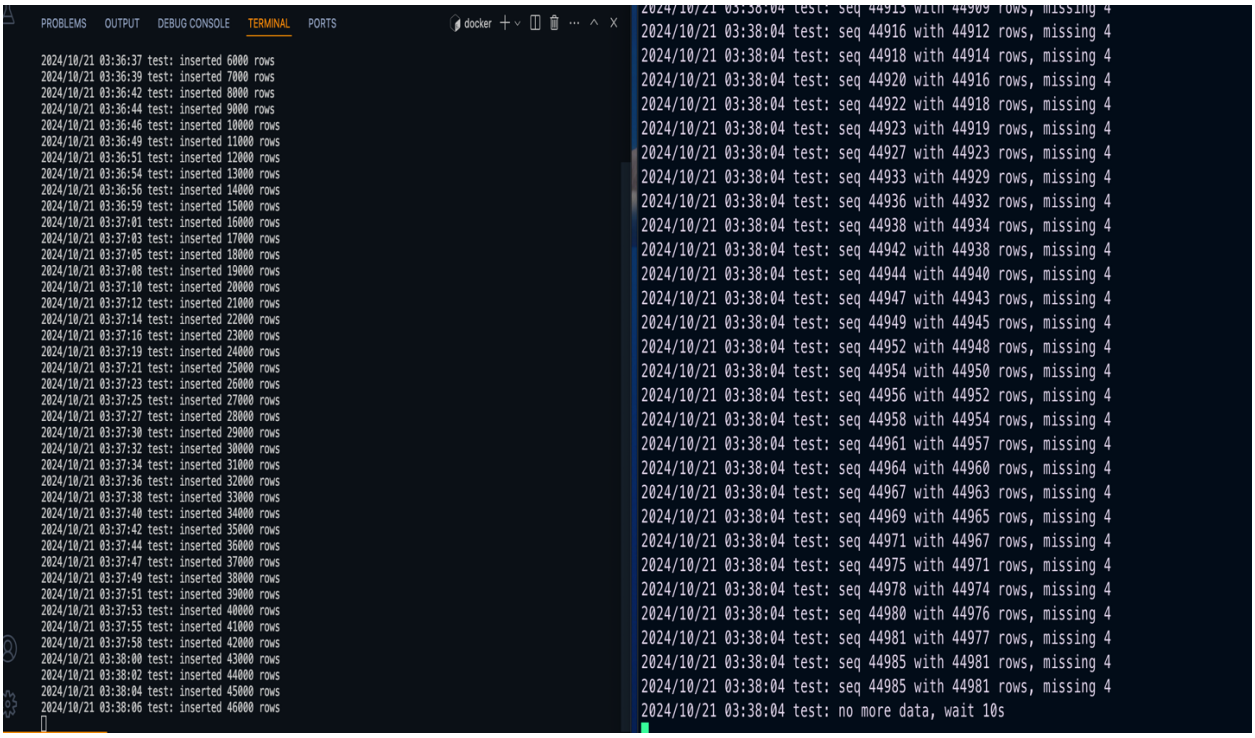
III. Tunable Consistency

Keep both reader and writer running. In a few minutes you should be able to see that the reader reports missing data. Provide screenshots and briefly explain what is happening.

Using the same consistency level (ONE) for both reading and writing can result in missing data. Cassandra uses the consistency level to determine the number of replicas required for successful write or read operations. Cassandra uses a consistency level of ONE, requiring only one replica to acknowledge the operation.
If a writer inserts data and acknowledges it, and a reader reads from a different replica (e.g., db2), data may not be propagated to that replica immediately. As a result, the reader may not find the expected data while querying the database.

Screen shot of reader db2 and writer db1 and missing data

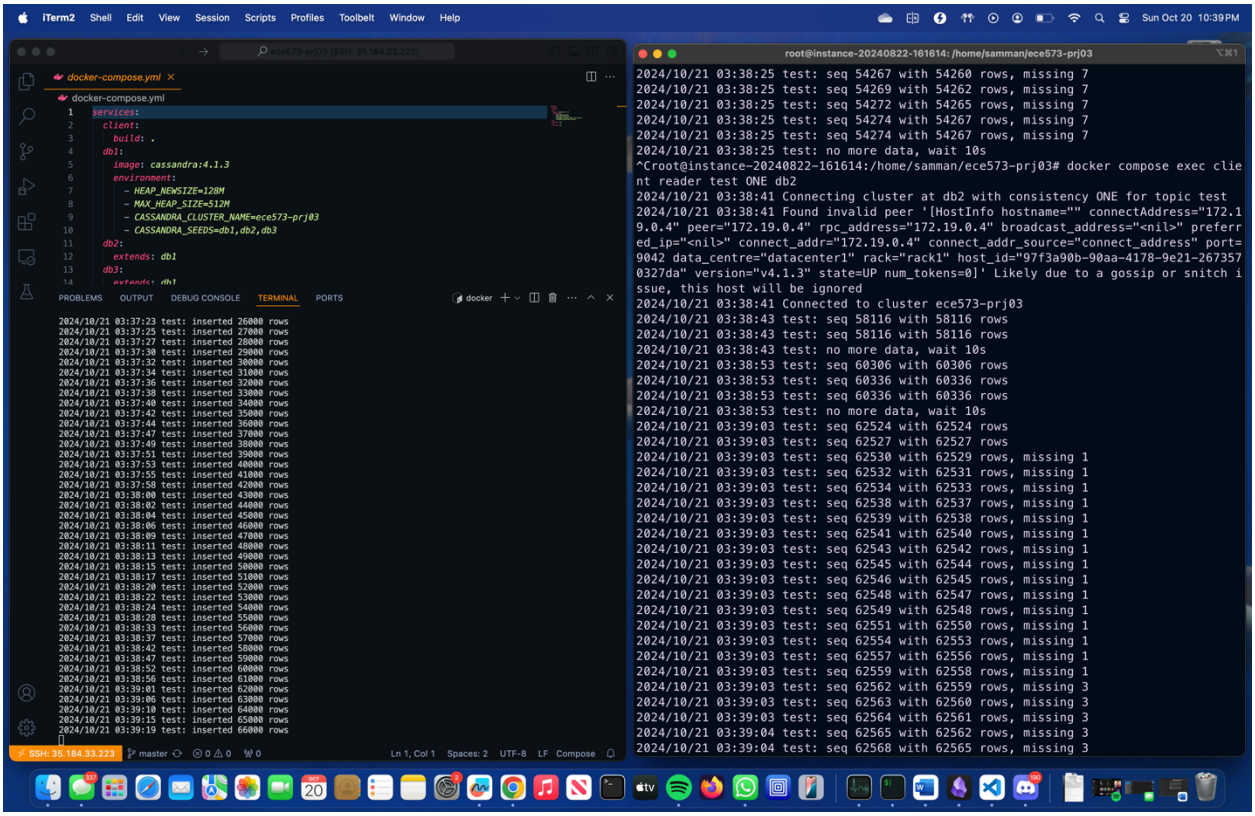




You can terminate the reader by Ctrl+C now and then restart it with the same arguments. Does it report any missing data? Provide screenshots and briefly explain what is happening.

Restarting the reader with the same parameters may result in same lost data as in the prior scenario. Cassandra replication may not have fully caught up with write operations during the short interruption while stopping and restarting the reader.
Missing data in Cassandra may be caused by asynchronous data propagation among nodes, leading to eventual consistency. When the reader restarts, it may read from a replica (db2) that has not yet received the most recent data from the writer, resulting in missing data.

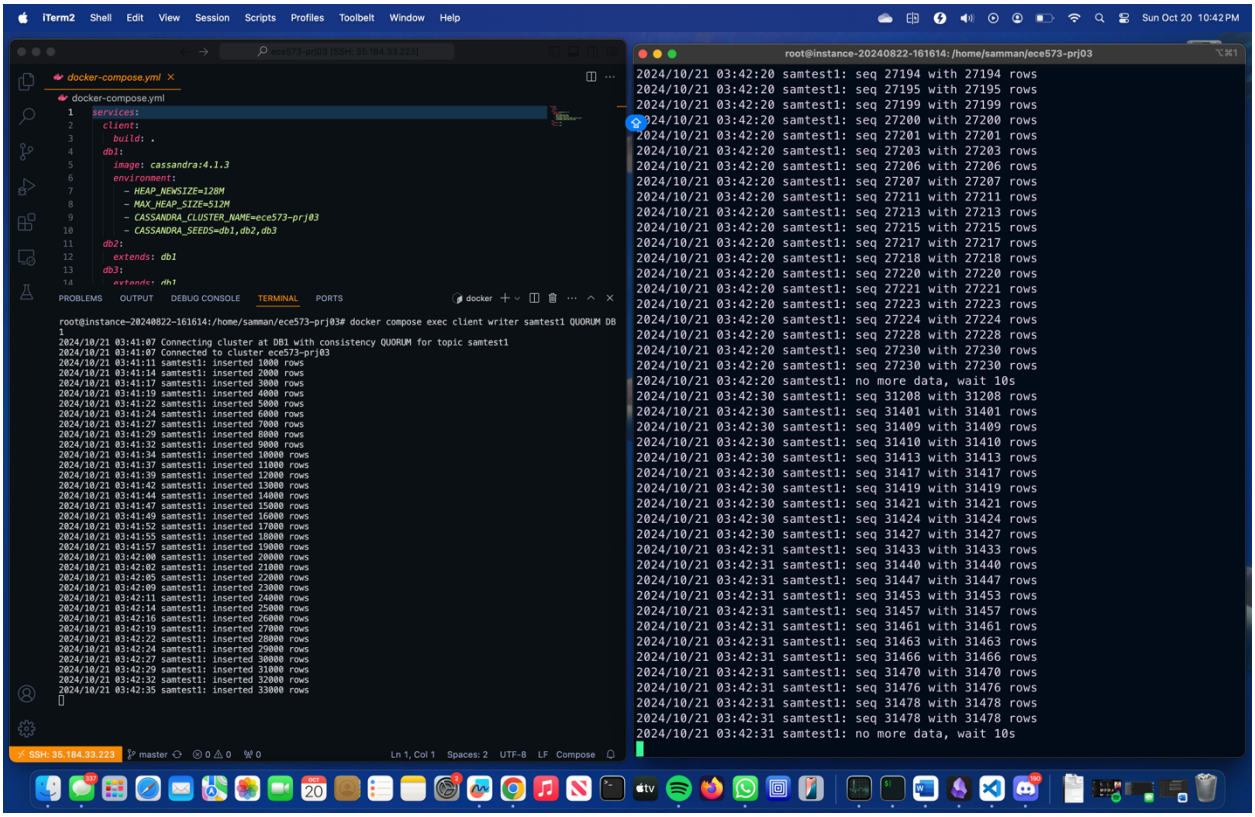
Screenshot missing data again



Stop both the reader and writer and restart them with a different topic and the QUORUM consistency. Let them run for a few minutes. Is the reader reporting any missing data and does that match your expectation? Provide screenshots and briefly explain why.

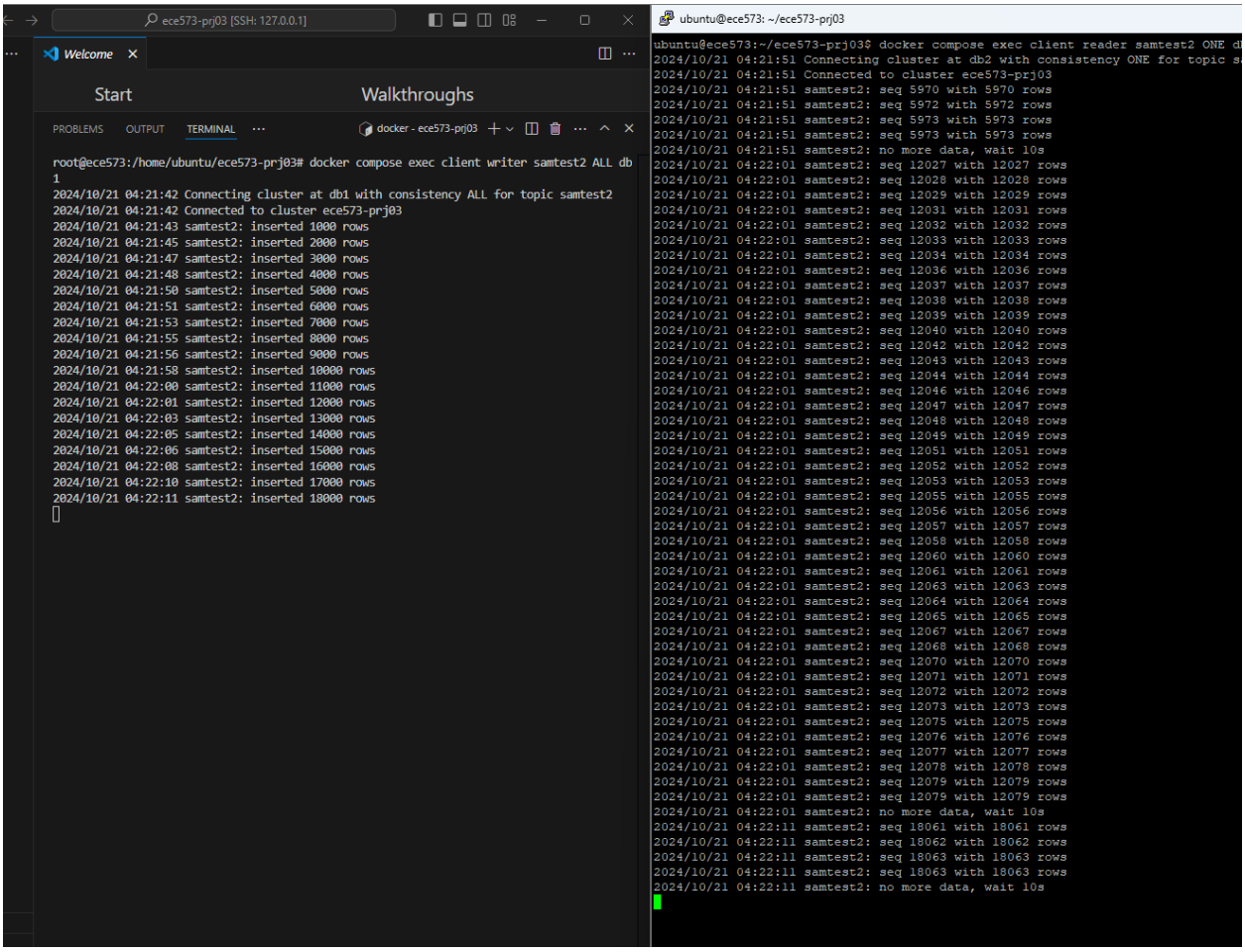
Using a QUORUM consistency level that requires more than half of replicas to acknowledge the operation reduces the likelihood of missing data.
QUORUM improves consistency and minimizes data discrepancies among replication.
The reader is less likely to report missing data in this case, so the findings should align with this assumption. QUORUM consistency improves data integrity and consistency compared to ONE.

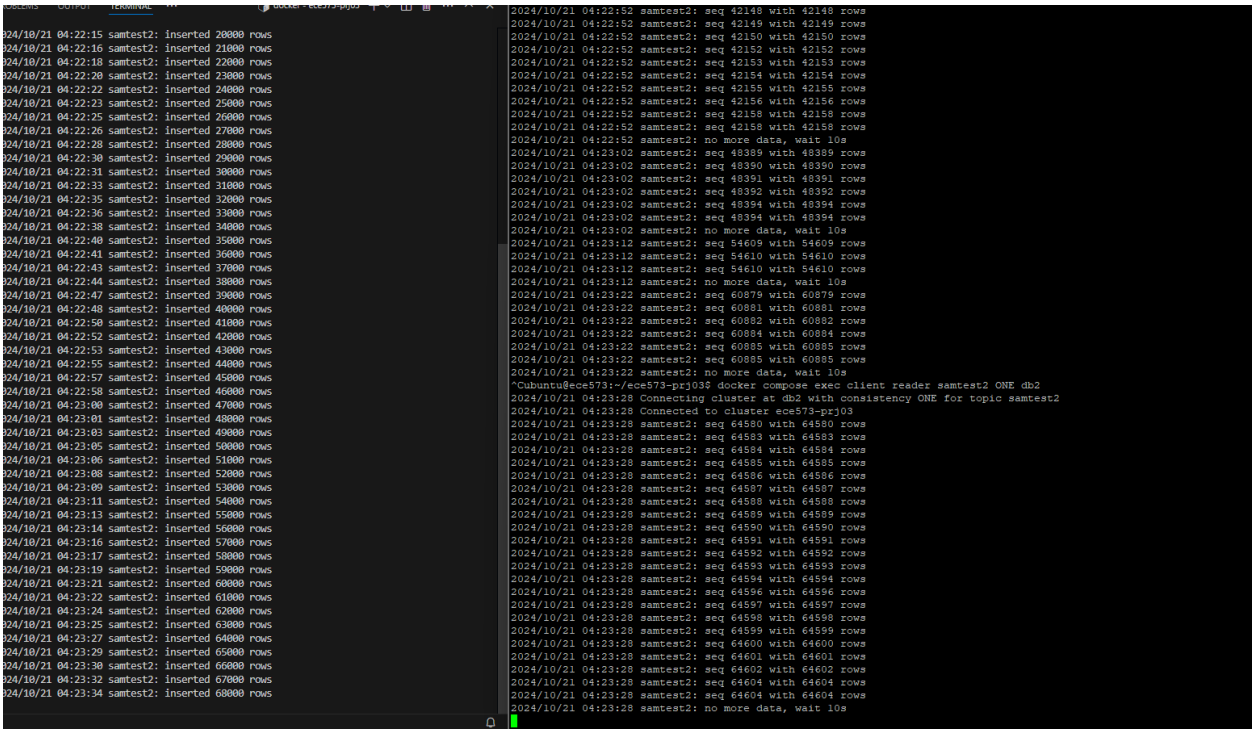
Screenshots of Quoram no loss with topic samtest1



Repeat the above experiment using writer with ALL consistency and reader with the ONE consistency. Don't forget to change to a different topic. Provide a screenshot and briefly explain your observations.

Screenshots





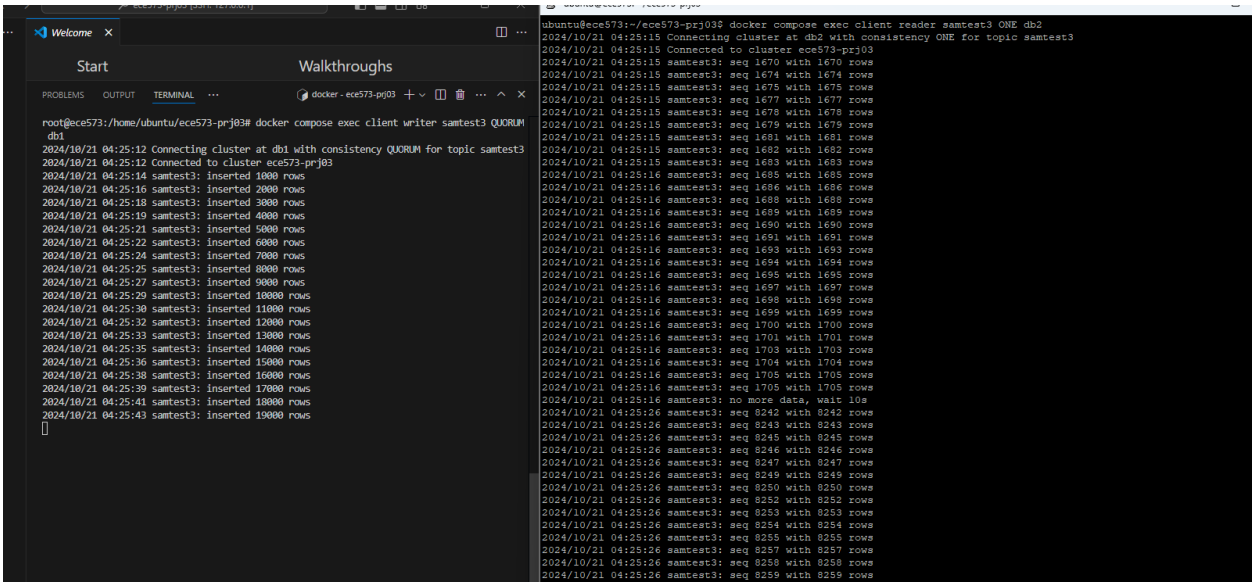
Explanation

When a writer utilizes ALL consistency and a reader uses ONE consistency, the latter is more likely to report missing data. Cassandra's ALL consistency requires all replicas to recognize an operation. If a writer acknowledges a write with ALL consistency and a reader attempts to read with ONE consistency level, it may read from a separate replica that has not yet received the data. As a result, readers may not find expected data, resulting in missing data.

What may happen if the writer uses the QUORUM consistency and the reader uses the ONE consistency?

If the writer employs QUORUM consistency and the reader utilizes ONE consistency, there may still be some situations of missing data, but not as frequently as with ALL consistency. QUORUM consistency needs a majority of replicas to recognize an operation and is more resistant to discrepancies than ONE consistency. However, it lacks the strength of ALL constancy. Even if the writer acknowledges a write using QUORUM and the reader reads with ONE, data may not fully propagate to the replica being read from, leading to missing data.

Screenshots



Since the source code of both writer and reader are available in our VM, can we run them from the VM directly without using the client container? Briefly explain why or why not.

You can run the writer and reader applications directly from the VM, without utilizing the client container. The source code for both the writer and reader is accessible, allowing for direct execution from the virtual machine. To ensure proper application execution, make sure the VM has all essential dependencies and configurations. Running them in containers offers a more controlled environment, but they can also be run directly on a virtual machine with the necessary software and libraries installed.

IV . Availability

● Modify writer.go to retry for a failed write operation. Use a retry policy to wait 10 seconds before retrying but allow an infinite number of retries.

Screenshot with the modified code

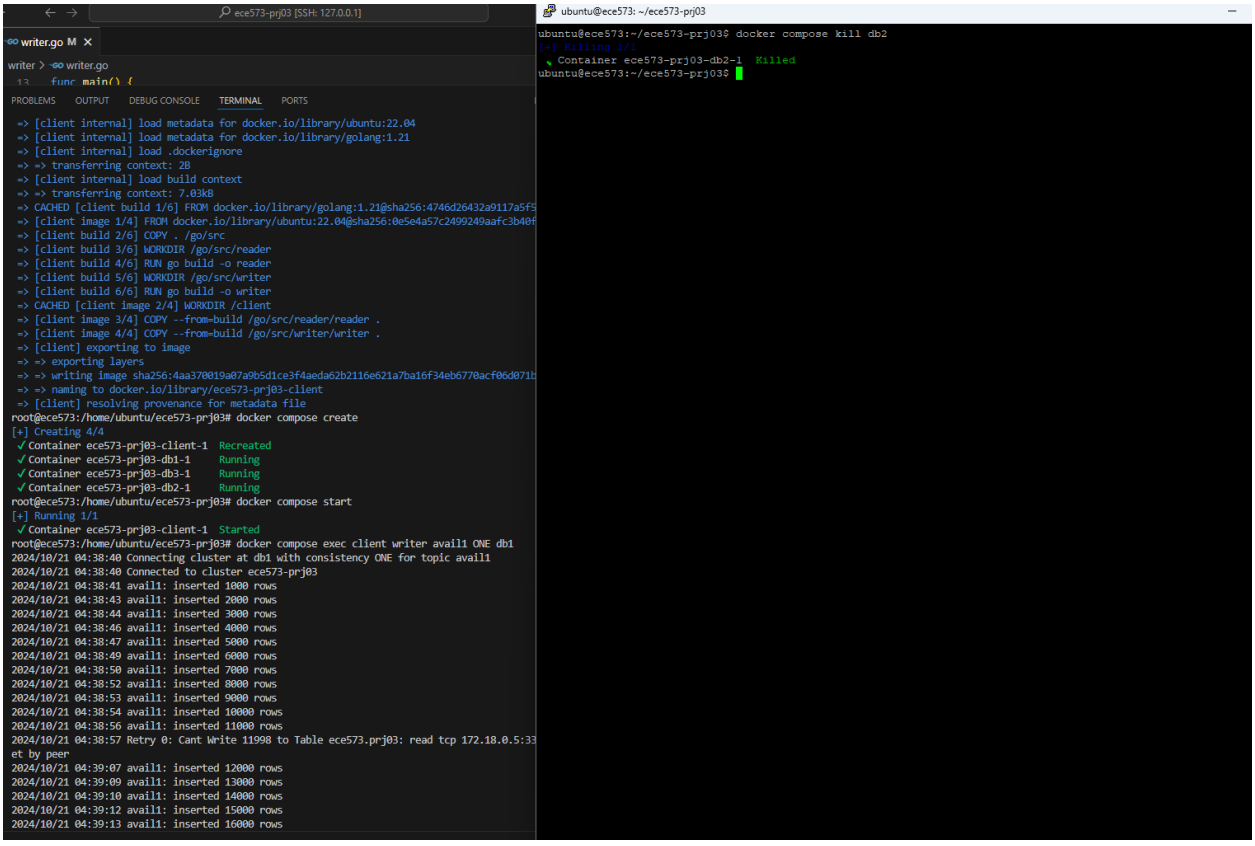
```
for seq := 1; ; seq++ {
    value := rand.Float64()
    retries := 0
    for {
        err := session.Query(
            INSERT INTO ece573.prj03 (topic, seq, value) VALUES (?, ?, ?),
            topic, seq, value).
            Exec()
        if err == nil {
            break //log.Fatalf("Cannot write %d to table ece473.prj03: %v", seq, err)

    }
    log.Printf("Retry %d: Cant Write %d to Table ece573.prj03: %v",retries, seq, err)
    time.Sleep(10 * time.Second)
    retries++
    }
    if seq%1000 == 0 {
        log.Printf("%s: inserted %d rows", topic, seq)
    }
}
```

Build, create, and start the client again and demonstrate that the writer can keep adding rows when db2 is killed. Provide screenshots as needed.

When using the modified writer with the infinite retry policy, it should continue to add rows even if db2 is terminated. The retry method attempts to write data until it is successful. No errors will be displayed, and the writer will be able to withstand temporary node failures.

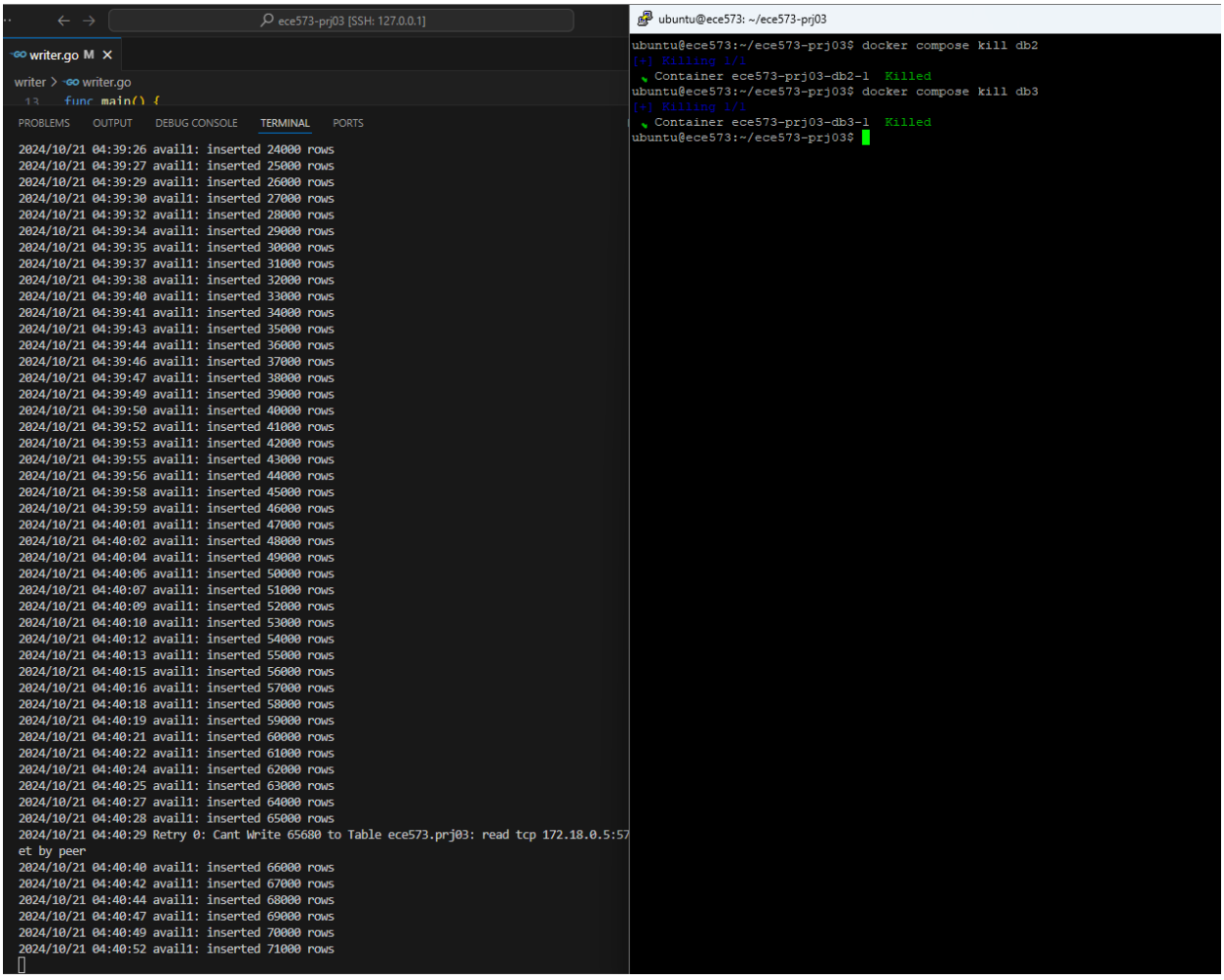
Screenshot



What if db3 is also killed? Provide screenshots and briefly explain your findings.

If both db2 and db3 are destroyed, the updated writer with infinite retry policy will continue to attempt writes notwithstanding node failures. The retry technique is particularly handy in this case. You should not observe any errors, and the writer should continue to add rows. The writer's robustness to node failures is due to the retry algorithm.
Please find the photos.

Screenshot



Restart writer with consistency QUORUM. Does the writer perform differently when one server is killed and when two servers are killed? Provide screenshots and briefly explain your findings

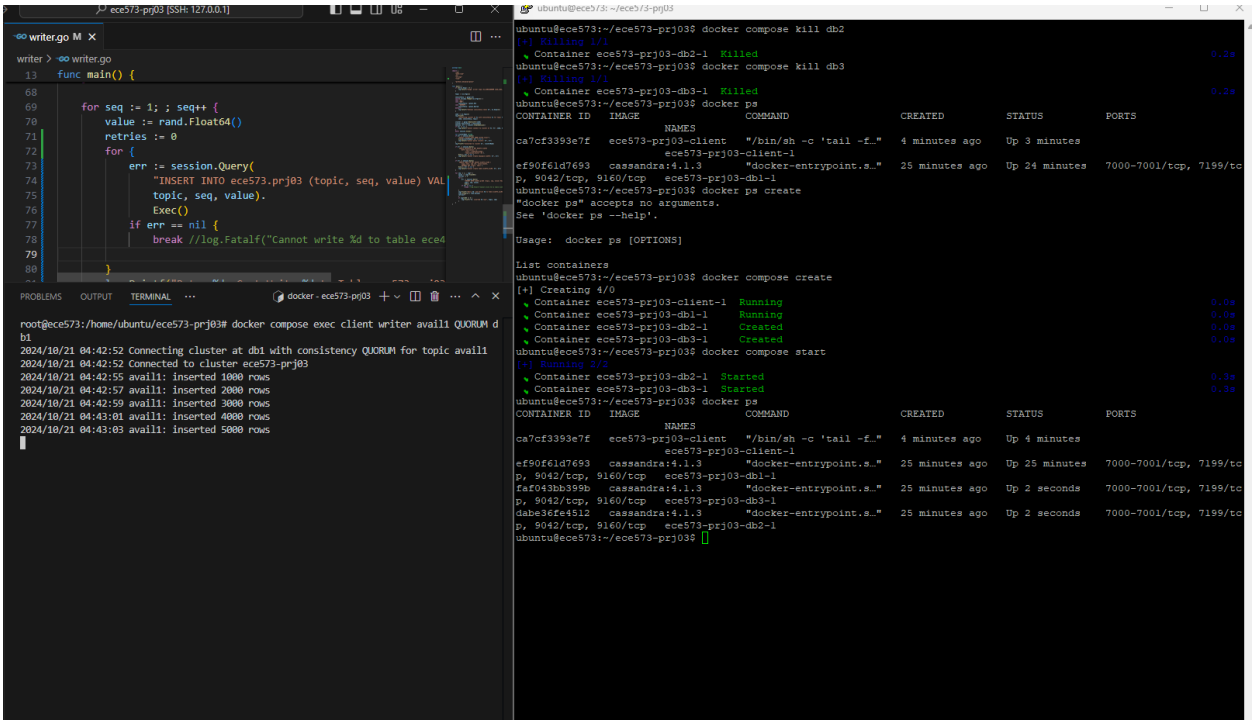
The QUORUM consistency can be investigated, for instance, from a db2 standpoint, when one server is killed. The writer will continue to work because the QUORUM consistency just requires the majority of the replicas to acknowledge the write operation. At this point, since we have a total of three nodes—db1, db2, and db3—it will work if at least two nodes are up.

- The writer can add rows, and you might have a little higher latency because of the loss of a single node. Zero errors should happen.

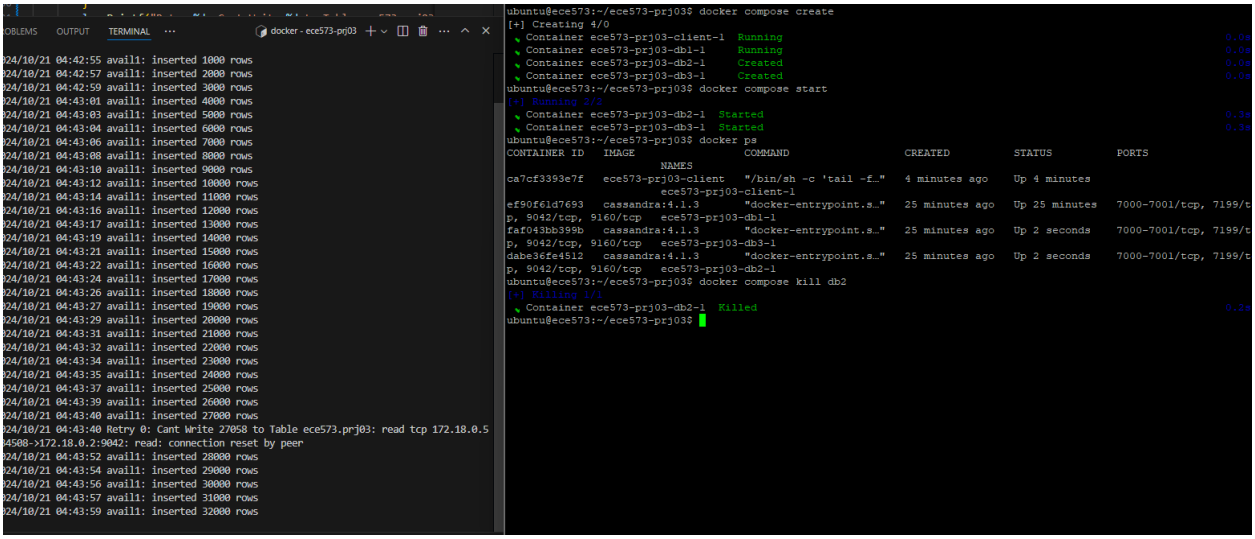
Two Servers Killed (db2, db3):

- If two servers are killed, and the consistency used by the writer is set to QUORUM, then that will never reach a quorum, which may lead to an error or sometimes failure to write. This is, in essence, because for any write, QUORUM needs at least two nodes to acknowledge the writer when three nodes are used. Now, with two of the servers killed, only one node is up and running, which cannot satisfy the achieved QUORUM. In this case, the writer is likely to encounter errors or failures to write.

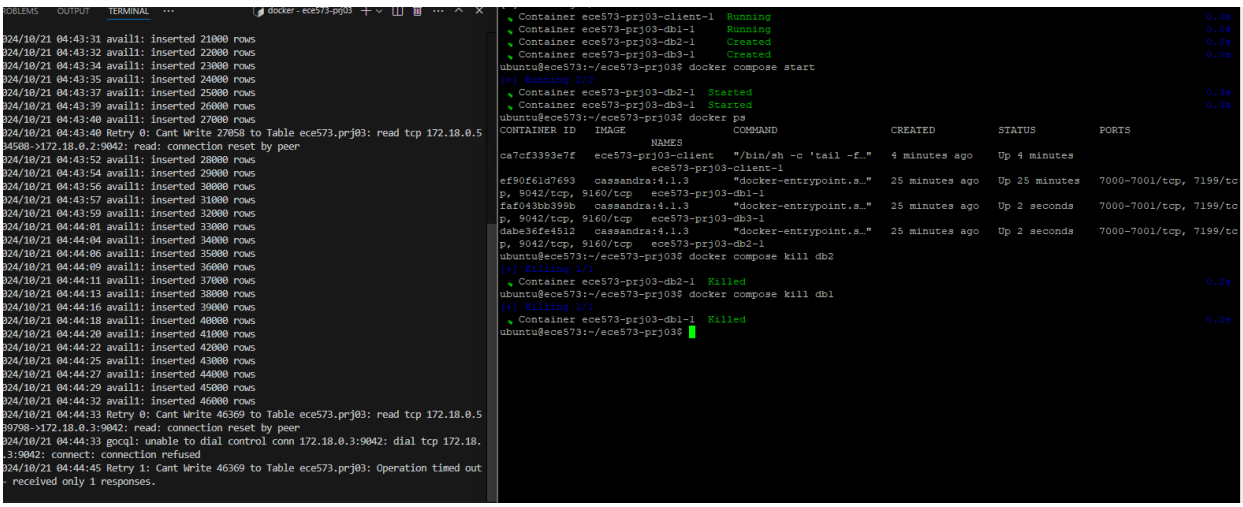
Screenshots



Only db2 is killed



Both db killed



Conclusion

In this project, we implement Cassandra Cluster Management by using Docker containers and analyze how the consistency levels of Cassandra would impact the resilience and performance of a distributed database. This project requires the simulation of server failures, configuration of consistency levels, and subsequent study of the behavior of the Cassandra cluster.

Key findings and takeaways of the project:

- **Cassandra Cluster Configuration:** Cassandra Cluster is set up using Docker containers for multiple nodes: cassandra cluster, db1, db2, db3, to simulate a distributed environment.
- **Consistency Levels:** We have tried playing with different consistency levels of ONE and QUORUM to see how it works for read and write operations in a distributed database.
- **Retrying Failed Operations:** We made several changes to the writer program in order for it to retry a failed write operation. It has been quite an important illustration of how applications of distributed systems should handle transient failures.
- **Resilience to Node Failure:** Our modified version of the writer program was able to sustain its writing of data despite the killing of individual nodes—for example, db2—thanks to the retry mechanism.
- **Quorum and Multiple Node Failures:** QUORUM consistency allows the writer to be killed and continue on when one node is killed but might not work when two nodes are killed since a quorum is required.
- **Variability Based on Configuration:** One should remember that Cassandra may have unforeseen behavior based on the replication factor, number of nodes, and special configurations in your cluster.