ECE573

PROJECT 03

SECTION 2

1. What happens if one misconfigures the CASSANDRA_CLUSTER_NAME variable in docker-compose.yml?

The nodes may not be able to recognize one another if the 'CASSANDRA_CLUSTER_NAME' is incorrectly configured in the `docker-compose.yml}, resulting in numerous distinct clusters rather than a single one. Because of this fragmentation, nodes are unable to communicate with one another, which can lead to inconsistent data and problems with replication. The formation of individual clusters will hinder the smooth distribution of data among all nodes, which is a crucial aspect of Cassandra's distributed design.

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

The nodes that new instances joining the cluster should get in touch with are specified by the `CASSANDRA_SEEDS` variable. These seed nodes provide communication and node discovery throughout the cluster. A few dependable and stable seed nodes are enough to allow new nodes to join the cluster and establish a network connection; listing every server as a seed is not required. While displaying too few nodes can lead to failure points in the event that the seed nodes go down, listing too many nodes can also be wasteful.

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

The output of `nodetool status` indicates that "UN" denotes "Up" and "Normal," indicating that the node is operational and actively involved in the cluster. "Normal" denotes that the node is properly managing its allocated data partition and is completely integrated into the data distribution ring, whilst "Up" denotes that the node is operational and interacting with other nodes. In the event that the node was unavailable or malfunctioning, additional indicators would appear in place of "UN."

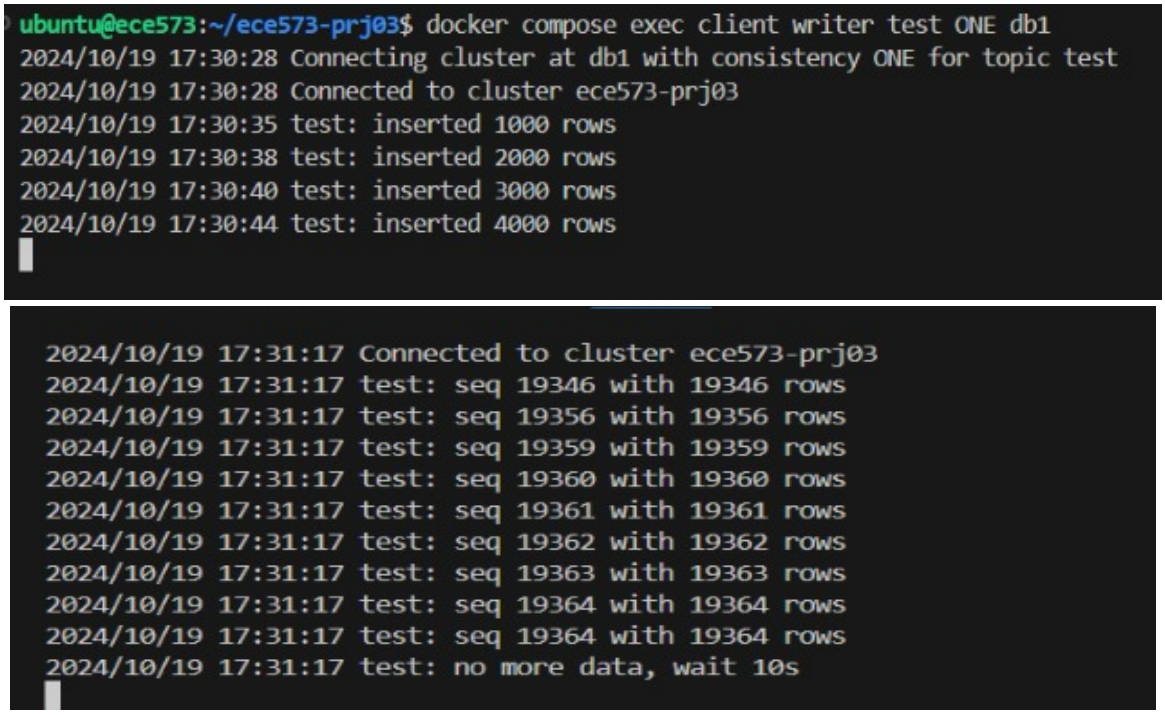
4. In our cluster consisting of containers, where does Cassandra store data?

Data is usually kept in a directory inside the container in a containerized Cassandra cluster. For persistence, this data is frequently mapped to a drive on the host computer. Through the process of mounting the Cassandra data directories ({/var/lib/cassandra} by default) on the host system, the data is preserved and obtainable regardless of the stopping, removing, or restarting of the container. This ensures that no data is lost during disruptions by enabling the permanent storage of Cassandra's SSTables, commit logs, and other crucial data throughout container lifecycles.

SECTION 3

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

The eventual consistency paradigm in distributed systems like as Cassandra, where writes are not instantly visible across all nodes, is the reason the reader is reporting missing data. The reader may momentarily encounter missing data as a result of replication delays, in which the writer's data has not yet spread to all replicas. In addition, if certain nodes are inaccessible or if the write operation hasn't been fully recognized by the cluster, various problems like network partitioning or write failures could potentially result in the reader missing data. The missing data should become visible once the system stabilizes or the data propagates.



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.

The same underlying causes, such as eventual consistency or insufficient replication, may cause the reader to indicate missing data even after you restart it with the same parameters after ending it with {Ctrl+C}. But by the time you restart the reader, it's possible that all of the nodes have synchronized the previously missing data. If the reader still reports missing data, there may be write inconsistencies, node failures, or other network-related problems affecting data visibility, or the data may not have been entirely duplicated. The reader should be able to retrieve the missing data after the replication is finished.

```
2024/10/19 17:31:59 test: seq 42099 with 42098 rows, missing 1
2024/10/19 17:31:59 test: seq 42099 with 42098 rows, missing 1
2024/10/19 17:31:59 test: no more data, wait 10s
^Cubuntu@ece573:~/ece573-prj03$ docker compose exec client reader test ONE db2
2024/10/19 17:32:14 Connecting cluster at db2 with consistency ONE for topic test
2024/10/19 17:32:14 Connected to cluster ece573-prj03
2024/10/19 17:32:14 test: seq 51576 with 51576 rows
2024/10/19 17:32:14 test: seq 51580 with 51580 rows
2024/10/19 17:32:14 test: seq 51582 with 51582 rows
2024/10/19 17:32:14 test: seq 51582 with 51582 rows
2024/10/19 17:32:14 test: no more data, wait 10s
```

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.

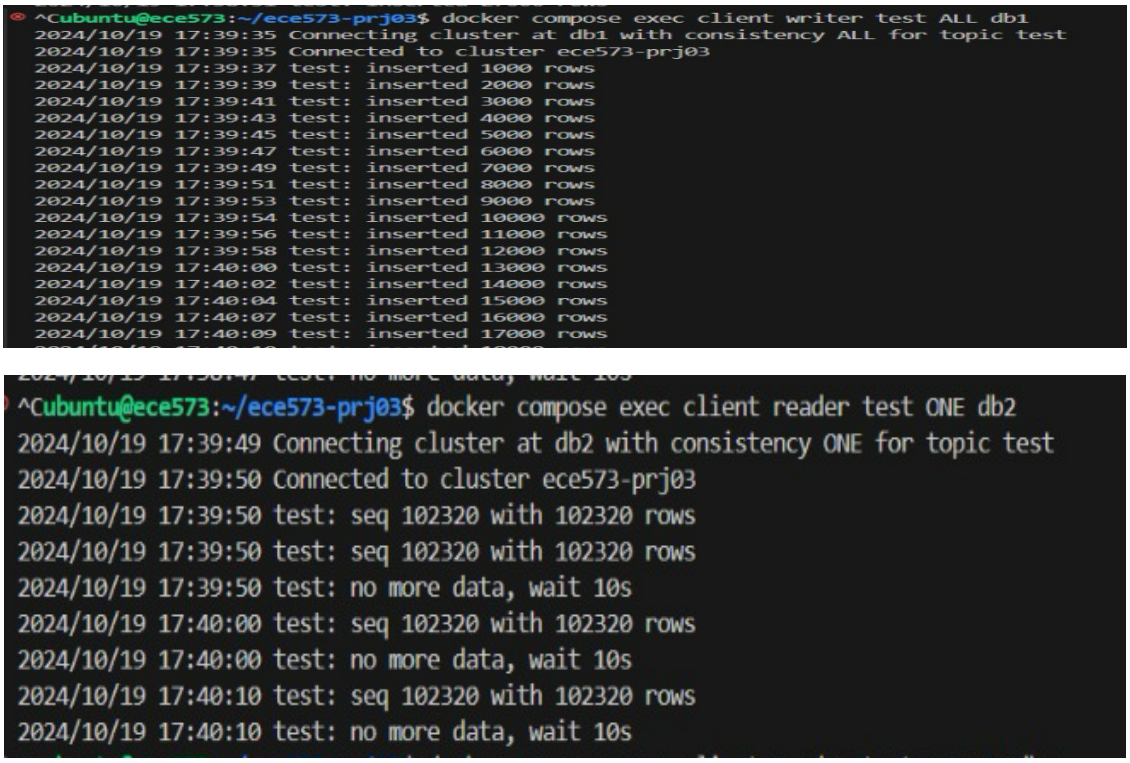
Because QUORUM consistency necessitates the acknowledgement of the transaction by the majority of replica nodes for both write and read operations, it is unlikely that the reader will report missing data. By doing this, the likelihood of stale or missing data is decreased because it is ensured that the data is written to and retrieved from enough nodes to preserve consistency. Since the writer verifies that the data is present in more than half of the replicas prior to confirming the write and the reader queries those same replicas, there is a strong likelihood that the data will be consistent and accessible across nodes in this scenario.

```
^Cubuntu@ece573:~/ece573-prj03$ docker compose exec client writer test QUORUM db1
2024/10/19 17:40:47 Connecting cluster at db1 with consistency QUORUM for topic test
2024/10/19 17:40:47 Connected to cluster ece573-prj03
2024/10/19 17:40:49 test: inserted 1000 rows
2024/10/19 17:40:51 test: inserted 2000 rows
2024/10/19 17:40:53 test: inserted 3000 rows
2024/10/19 17:40:55 test: inserted 4000 rows
2024/10/19 17:40:56 test: inserted 5000 rows
2024/10/19 17:40:58 test: inserted 6000 rows
2024/10/19 17:41:00 test: inserted 7000 rows
2024/10/19 17:41:02 test: inserted 8000 rows
```

```
^Cubuntu@ece573:~/ece573-prj03$ docker compose exec client reader test QUORUM db2
2024/10/19 17:40:59 Connecting cluster at db2 with consistency QUORUM for topic test
2024/10/19 17:40:59 Connected to cluster ece573-prj03
2024/10/19 17:41:00 test: seq 102320 with 102320 rows
2024/10/19 17:41:00 test: seq 102320 with 102320 rows
2024/10/19 17:41:00 test: no more data, wait 10s
2024/10/19 17:41:10 test: seq 102320 with 102320 rows
2024/10/19 17:41:10 test: no more data, wait 10s
```

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

Strong consistency is provided throughout the cluster in this experiment by utilizing the writer with ALL consistency, which guarantees that the data is written and recognized by every replica node. But since the reader set to ONE only needs a response from a single replica node, if that node hasn't synchronized with the others yet, it could end up reading stale or outdated data. Because of this, even when the author ensures that every node contains the most recent information, if a reader relies exclusively on one node, there is a chance that some or all of the data may be missing or inconsistent—especially if that node is out of date. This configuration demonstrates the trade-off between robust writes and shaky reads, which may result in disparities in the data that the reader can see.



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

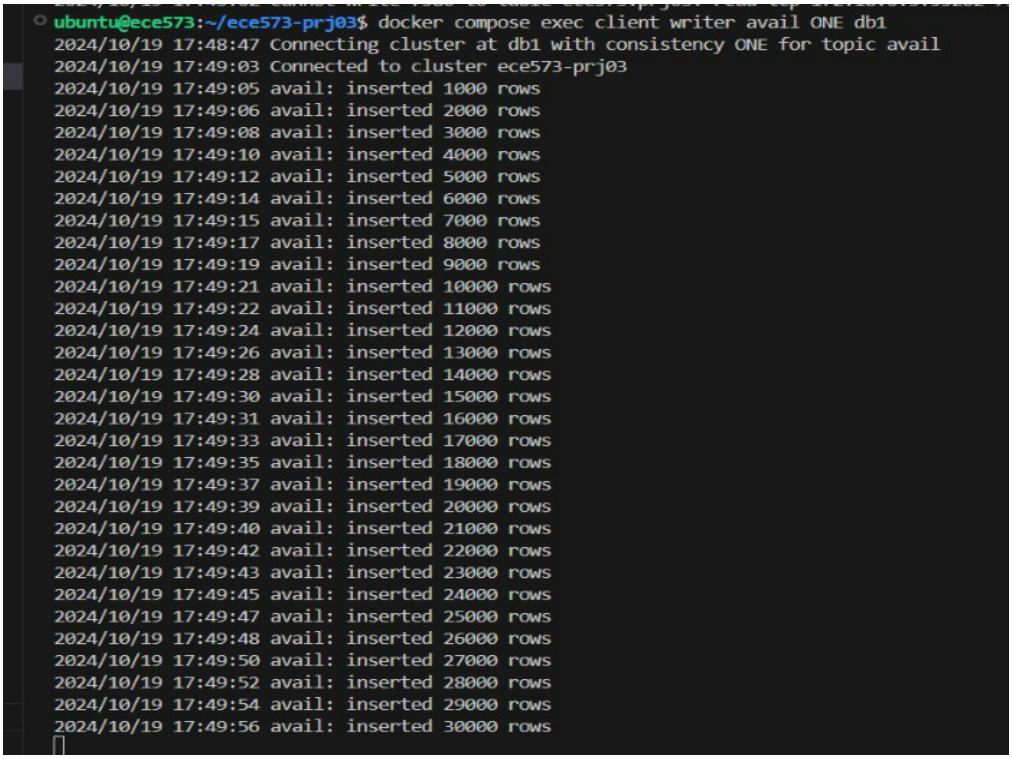
A write operation is not deemed successful until the majority of replica nodes have acknowledged it, which is ensured when the writer employs QUORUM consistency. This ensures that the data is consistently and dependable saved throughout the majority of nodes. But just one replica node needs to acknowledge it if the reader utilizes ONE consistency. Therefore, if the particular node the reader asks hasn't received the most recent changes from the other copies, it may provide stale or outdated data. The reader may see incorrect or missing data as a result of this mismatch, even though the write was successful and accepted by most nodes.

2. Can we run the writer and reader from the VM directly without using the client container? Briefly explain why or why not.

Yes, as long as the virtual machine (VM) has the required environment set up, including the right libraries and dependencies for establishing a connection to the Cassandra cluster, it is feasible to run the writer and reader straight from the VM without using the client container. It might be necessary to configure extra networking settings and make sure the Cassandra driver can connect to the cluster if you run them outside of the container. Running directly from the VM may cause issues with configuration and dependency management because containers frequently encapsulate the environment and dependencies, making setup simpler and more consistent.

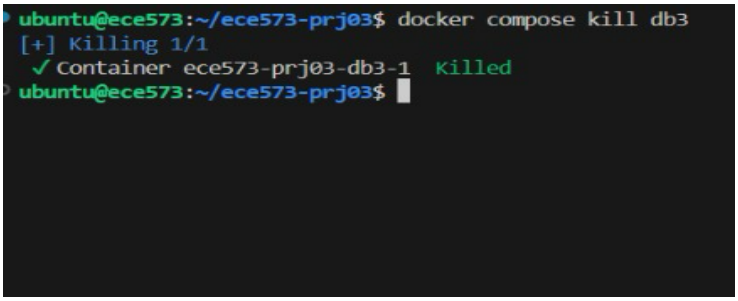
SECTION 4

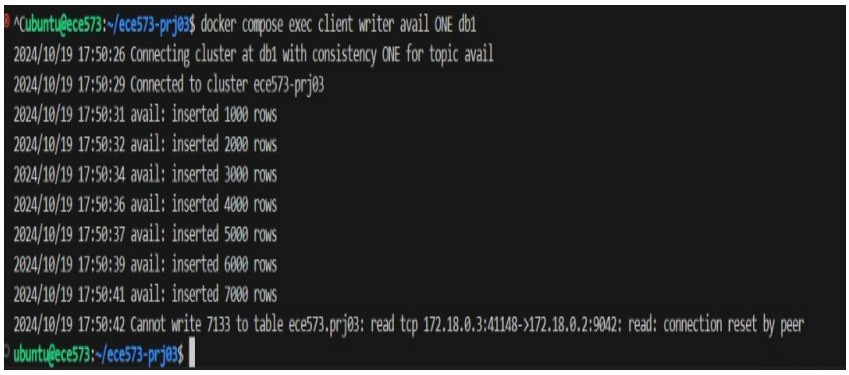
Build, create, and start the client again and demonstrate that writer can keep adding rows when db2 is killed. Provide screenshots as needed.



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

The consequences are dire if {db3} is killed or becomes inaccessible. If the writer is configured for ALL or QUORUM consistency, it may experience write failures or timeouts during write operations because `db3} is missing, which keeps the required number of nodes from recognizing the write. On the read side, if the reader is using a ONE consistency level, it can still retrieve data from other nodes that are available, but if those nodes aren't fully synchronized, there's a chance that the data it finds will be outdated or inconsistent. On the other hand, if the reader employs QUORUM, it will be unable to finish read operations because it needs a majority of nodes, and the lack of {db3} obstructs this need.





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.

Depending on how many servers are killed, the writer responds differently when resumed with QUORUM consistency. The writer can still successfully complete write operations even if only one server is destroyed since there is still a chance for the surviving nodes to reach quorum, which requires that more than half of the nodes acknowledge the write. The writer, however, will probably be unable to write data if two servers go down since it will not be able to meet the quorum requirement with fewer active nodes. This illustrates how crucial it is to have enough copies in a distributed system such as Cassandra to guarantee data consistency and availability, since the loss of many nodes can seriously impair the system's capacity to perform write operations.