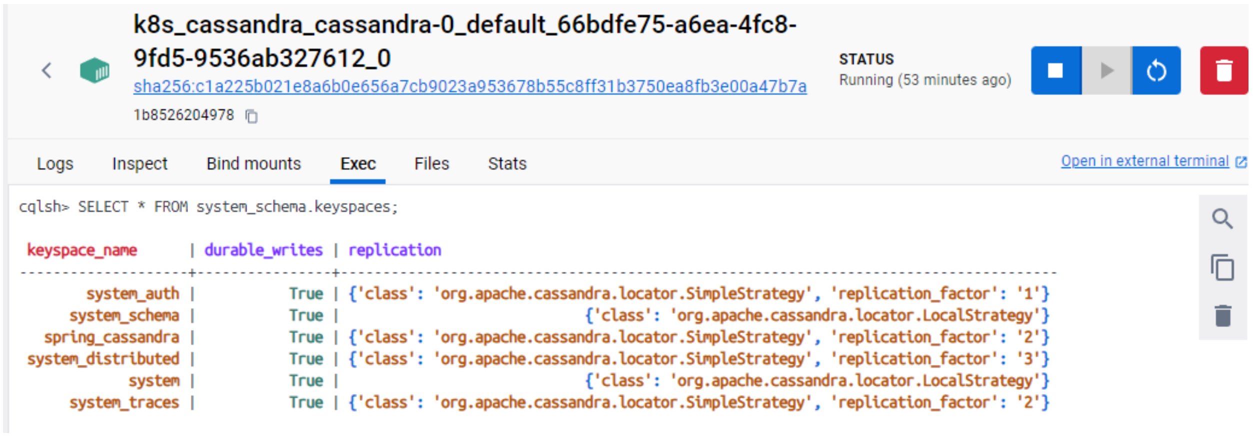
Cassandra Deployment Report

Introduction

This report is created to show the details into the setup of the Cassandra database used in my project. The configuration is used to manage data related to storing information about movies, user preferences, and watch parties. I will give more detailed explanation about the setup, the rationale behind my choices and the possible future plans for scalability.

Initialization and keyspace creation

I have deployed the Cassandra keyspace `spring_cassandra` using `SimpleStrategy` with a replication factor of 2 through a Kubernetes Job, that I deploy once the main Cassandra instance is loaded and stable. I have decided to use a replication factor of 2 for the development stage of the project. This way I am protecting against the failure of a single node, while at the same time minimizing storage and operational overhead.


k8s_cassandra_cassandra-0_default_66bdfe75-a6ea-4fc8-9fd5-9536ab327612_0
sha256:c1a225b021e8a6b0e656a7cb9023a953678b55c8ff31b3750ea8fb3e00a47b7a
1b8526204978
STATUS
Running (53 minutes ago)
Logs
Inspect
Bind mounts
Exec
Files
Stats
Open in external terminal

```
cqlsh> SELECT * FROM system_schema.keyspaces;

 keyspace_name      | durable_writes | replication
--------------------+----------------+-------------------------------------------------------------------------------------
        system_auth |           True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '1'}
      system_schema |           True |                             {'class': 'org.apache.cassandra.locator.LocalStrategy'}
   spring_cassandra |           True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '2'}
 system_distributed |           True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
             system |           True |                             {'class': 'org.apache.cassandra.locator.LocalStrategy'}
      system_traces |           True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '2'}
```

Screenshots for the rest of the setup can be found in the portfolio evidence folder.

A replication factor of 3 or 5 could be a better option for important data, however, my database will be holding not so critical data in the form of movie information and watchparties, hence the focus on the minimal storage. The benefits of a higher replication factor can be seen in the cases of 2 node failures. With a replication factor of 3 or 5, the requirements fir storage capacity are 3x and 5x, which significantly increase the storage costs. This may not be such a problem if the database stores critical data, but this is not the case for me. Another thing to consider is the replication factor for even and uneven numbers. Cassandra often uses quorum reads and writes to ensure consistency and it is defined as **(RF/2) + 1 nodes**. With even replication factor achieving a majority for quorum can be more challenging and less efficient. For instance, with RF=2, quorum is 2 nodes, meaning all nodes must agree, which negates the benefit of redundancy in some failure scenarios.

Odd numbers provide better fault tolerance characteristics. For instance, RF=3 can tolerate one node failure while still maintaining quorum, whereas RF=2 can’t tolerate any node failure without losing quorum. While this setup means that I will not lose the data if ONLY one node fails, the data will be lost if the remaining node fails before the first one has recovered. Furthermore, the consistency of the read and write will be lost as the data read may be stale if the remaining node has not updated the data.

Cassandra uses mechanisms like read repair to eventually synchronize the nodes, but during the failure, consistency issues can arise.

Since the future of the application is not limited to the market of one country, the use of `NetworkTopologyStrategy` is something to consider as it would offer optimized data replication and reduced latency across potentially multiple data centers.

To determine if and when to think about changes in the setup a continuous monitoring after the launch of the application will need to be done and the decisions based on that. The transition strategy would be focused on maintaining high availability and minimizing disruption during the migration.

Persistent data storage

The system includes a PersistentVolumeClaim that provides 1 GiB of storage, which is very important for maintaining data integrity and availability across pod restarts and failures. This initial capacity supports the current workload but is scalable to accommodate future increases.

Deployment configuration

The deployments of the Cassandra pods is done by StatefulSet with an initial single replica. Together with that the setup includes a configuration of the ports in the network of the Kubernetes cluster, health checks. While a single initial replica of the pods is enough for now, since the load is not that high and the limiting factors are the other services, a good future consideration scaling the number of the replicas based on the data derived from the monitoring tool Grafana that I have integrated into the Kubernetes cluster. This, combined with the metrics server, responsible for the autoscaling will ensure that the performance of the application will be sufficient at all loads.