UE17CS352: Cloud Computing

Class Project: Rideshare

Date of Evaluation: 21/05/2020
Evaluator(s): Prof.Usha Devi BG and Prof.Sanjith
Submission ID: 298
Automated submission score: 10

S No.	Name	USN	Class/Section
1	Sathvik N Jois	PES1201700213	A
2	Hemanth C	PES1201701566	A
3	Rachana Jayaram	PES1201701573	A

Introduction:

The main goal of this project was to create a DBaaS(Database as a service) for RideShare app which is a fault-tolerant, highly available database as a service for the RideShare application using Amazon EC2 instances.

A Master-Slave architecture for the highly available database was implemented, so that it may replace the existing APIs for reads and writes in the Users and Rides instances. The existing interface and the implementation for reads and writes were migrated to the Master-Slave Architecture established.

Algorithm / Design:

There are 3 main aspects that have to be taken care of when creating a distributed computing system are -

- Isolating each node from another node.
- Ensuring communication between the nodes.
- Ensuring Coordination between the nodes.

The nodes were isolated using docker containers. Communication between the isolated containers was achieved through RabbitMQ and coordination between the nodes of the cluster was achieved through Zookeeper.

Design of Zookeeper:

All workers were made to have the same code, to facilitate the replacement of a failed master node with a slave node. Since all workers have the same code, a zookeeper was used to identify whether a given node is a Master Node or a Slave Node. This was done by categorizing all the workers under `/Worker Znode` and only the Master-Details are copied to the Znode path `/Master` as shown in the storage architecture in Fig 1.

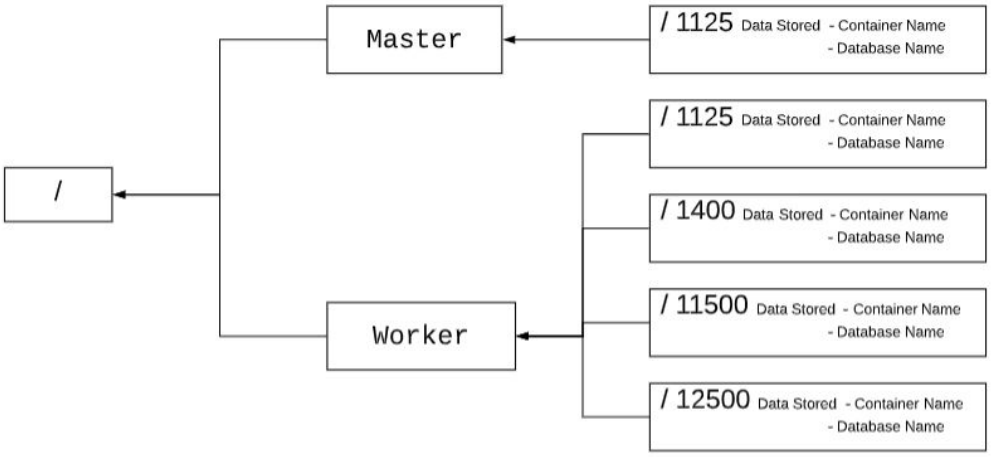

Fig. 1. Storage architecture

This implementation makes it easier to make fast inquiries about the current Master. This also allows all the workers to be classified under the Znode path `/Worker`.

Usage of RabbitMq:

As stated above each worker can be classified as a Master or a Worker. RabbitMq uses AMQP to send messages in the form of queues from a producer node to a consumer node.

There are namely 4 queues used in this implementation -

1. Write Queue
    - The orchestrator appends a write message in the Write Queue.
    - If a given worker is the master node it should process the message and upon successful execution, it should broadcast this, else return the error response back to the Orchestrator in the response queue
    - If the current node is not a master node then it should append the message back into the queue.

2. Read Queue
    - This Queue is used to process read requests from the Orchestrator.
    - If the current node is a Slave Node, the node should process the read request and send back the requested data
    - If the node is not a slave node, it should append back the message into the Read Queue.

3. Sync Queue
    - This queue is used to sync the Master DB writes into the Slave DB.
    - This is so that the eventual consistency is achieved and all the DB are in sync with the Master Node.

4. Response Queue
    - This basically contains the response from either Slave or Master node for the corresponding Read or Write requests.

TESTING:

- To test the Setup, we performed 100 sequential write requests and 100 read requests and compared it with the expected output, and hence verified whether the reads and writes are being synchronously updated.

- To make sure each request is being mapped to the correct sender, we use locks to make sure requests are being served in a sequence.

- To check fault tolerance, we stop the running process from the command prompt to check if the watches on the Znode path are being triggered or not.

CHALLENGES:

- Lack of documentation of docker sdk and zookeeper.

CONTRIBUTIONS:

Name	Contribution
Sathvik N Jois	RabbitMQ, setting up a Load balancer and target groups
Hemanth C	Setting up instances, Scaling up and down using docker sdk zookeeper and error debugging
Rachana Jayaram	Writing all base APIs, In-depth unit testing for the APIs, Planning the architecture, Code maintenance, and documentation.

CHECKLIST

S No	Item	Status
1	Source code documented	done
2	Source code uploaded to a private GitHub repository	done
3	Instructions for building and running the code. Your code must be usable out of the box.	done