UE17CS352: Cloud Computing

Class Project: Rideshare

Date of Evaluation: 17th May,2020
Evaluator(s): Srinivas KS
Submission ID: 649
Automated submission score: 10

SNo	Name	USN	Class/Section
1	GR Dheemanth	PES1201700229	G
2	Skanda VC	PES1201700987	G
3	Mehul Thakral	PES1201701122	G

Introduction

In today's systems, high availability is a must for applications. Even a few minutes of downtime can mean a serious loss of business. The main bottleneck for high availability many of the times is the database service. So our project was based on building a fault-tolerant database with high availability. This was demonstrated on the rideshare application which was built in the previous assignments.

Related work

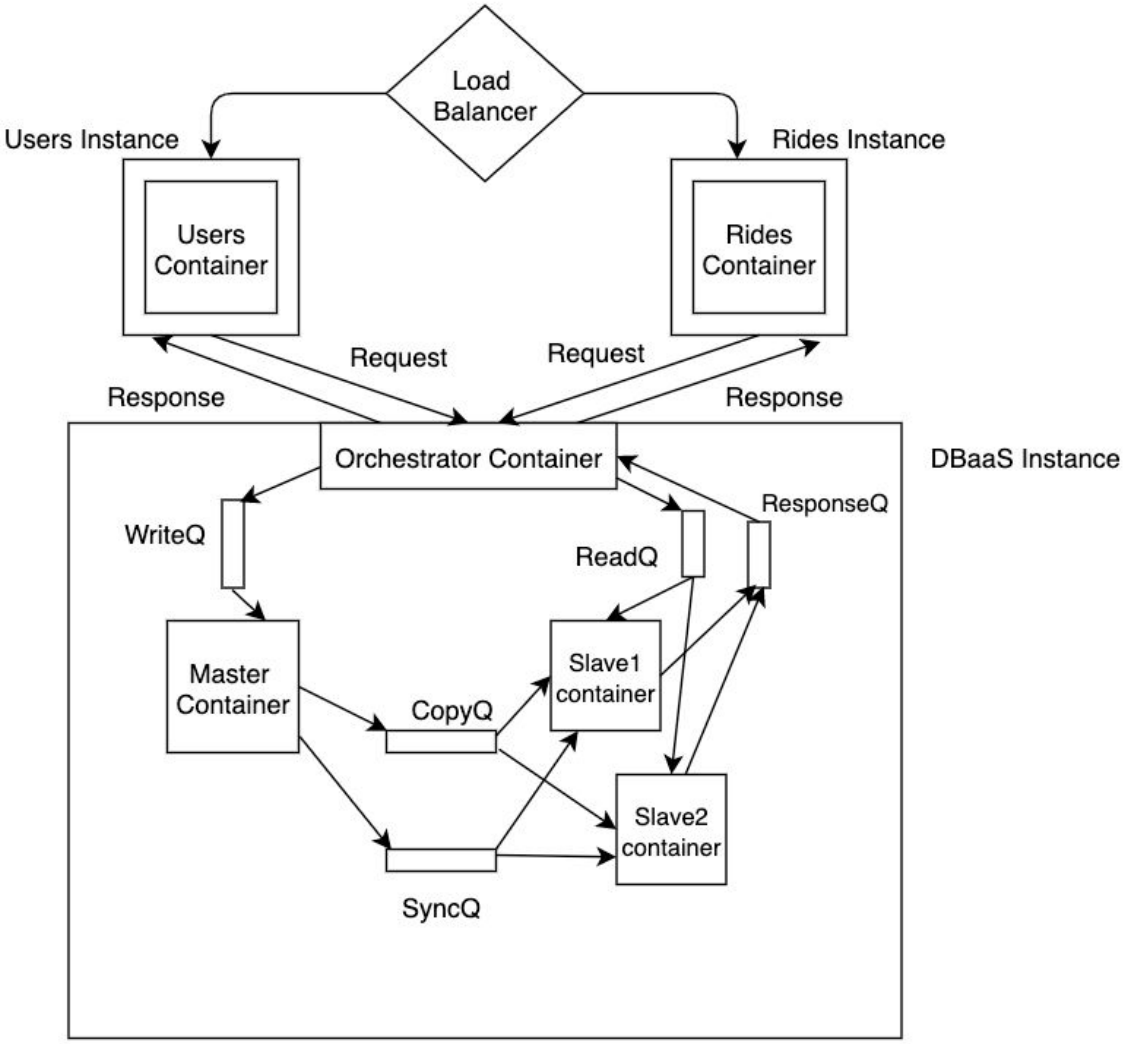
We studied about Zookeeper, docker-sdk and rabbitmq. Docker SDK was used for programmatically starting and stopping containers. We studied how a new container can be created from the orchestrator. We also studied about rabbitmq and the process of setting up queues for communication between the master and slave nodes. We also studied about zookeeper and znodes and how they can be used to ensure high availability.

Design

In order to provide high reliability, the following system architecture was used. It is given in the below flow diagram. Whenever a read request has to be made, it is redirected to the slave nodes. The slave nodes/containers only service read requests. The read request is forwarded by the orchestrator to the readQ. Orchestrator is like an endpoint for all the database requests. The data from the database is forwarded back to the orchestrator using ResponseQ.

The SyncQ is used for synchronization between the master and slave nodes. Since each container has a separate database, it becomes necessary to synchronize whenever data is written to the database by the master. So whenever data is written to master, it is also written to SyncQ. The data is then read from SyncQ by the slave nodes and written to their respective databases. A CopyQ is also maintained to write all the data from the database of the master to a new slave which is spawned. A new slave spawned initially has an empty database.

Then to implement fault tolerance, zookeeper is used. Each slave has a respective znode attached to it. The znode used is an ephemeral node which means that when the slave crashes for some reason, the znode also gets deleted. This is notified to the zookeeper. This mechanism can be used to spawn a new slave. Also to implement scale-in/scale-out functionality, the count of the requests is maintained in a database in the orchestrator. Whenever the count exceeds a threshold, a new slave is spawned.



Overall System Architecture

Testing

What were the testing challenges

- Due to a lot of parallel components within the application it had to be made sure that perfect *unit testing* is practised in order to ensure that on integration everything works as expected.
- Also due to the extent of the parallel components within the application finding out the source of some errors became an arduous task which actually took more time than to solve that particular error.

Explain how you fixed the issues on automated submission

- There were actually no issues faced at the time of automated submission due to the testing strategy which was put in place and used.

Challenges

1) Finding out a way to run a timer which will invoke a callback function for scaling was a challenge in terms of running such a timer process within the multi-threaded flask server in which worker processes would keep dying after API call ends and this timer had to run irrespective of their starting and stopping.

Contributions

SNo	Name	Contribution
1	GR Dheemanth	Master/slave, Replication and Synchronization
2	Skanda VC	Orchestrator and RabbitMQ client
3	Mehul Thakral	Scaling and Fault Tolerance (Zookeeper)

CHECKLIST

SNo	Item	Status
1	Source code documented	done
2	Source code uploaded to the private GitHub repository	done
3	Instructions for building and running the code. Your code must be usable out of the box.	done