

17CS352: Cloud Computing

Class Project: Rideshare

Date of Evaluation:18/05/2020

Evaluator(s):

Submission ID:558

Automated submission score: 10

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | USN | Class/Section |
| 1 | Archana C | PES1201701384 | G |
| 2 | Keerthi Priya P | PES1201701643 | G |
| 3 | Thanusha Melavoy Sai | PES1201701457 | G |
| 4 | BhanuPriya C G | PES1201701742 | D |

# Introduction

RideShare is an application built on Flask, REST API’s and docker and is a platform to pool rides. It is a smaller version of cab booking applications like Ola and Uber. Here we are creating a backend for the application. The application built is fault tolerant, Highly available and auto scalable. The Application has a Load balancer, Users and Rides microservice, An Orchestrator providing Database as an Application. The Application is built on REST API and Flask for backend and Docker to maintain containers, Zookeeper and Kazoo for High Availability, Rabbitmq for syncing data between workers.

## Related work

* https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce
* https://hackernoon.com/what-is-amazon-elastic-load-balancer-elb-16cdcedbd485
* ​<https://www.rabbitmq.com/getstarted.html>
* ​<https://kazoo.readthedocs.io/en/latest/>
* ​<https://docker-py.readthedocs.io/en/stable/>
* stackoverflow

## ALGORITHM/DESIGN

The application has two microservices namely User and Rides present on two different AWS instances where User microservice is responsible for creating users and Rides microservice is responsible for creating rides and joining rides, etc. There is an AWS Application load balancer which will distribute incoming HTTP requests based on urls . There is an Orchestrator which is responsible for providing Database as an Application. It will listen to incoming HTTP requests from users and rides microservices and perform the database read and write accordingly. Here, we are using Zookeeper to maintain high availability of workers. The orchestrator is responsible for auto scaling based on an incoming HTTP requests and the Auto-scale runs as a thread and starts when the first read request is received .The orchestrator maintains master-slave model where master is responsible database writes and slave is responsible for database reads .We are using Rabbitmq which uses Advanced Message Queue Protocol so that database writes by the master are in synchronous with slave’s database as each worker have their own database. Using Rabbitmq, we have used READQ and RESPONSEQ for receiving http requests from the orchestrator and sending the response back through the RESPONSEQ. We have maintained WRITEQ to write into the database from the orchestrator and SYNCQ to sync master’s database with the slave’s database and we have used fanout exchange to implement SYNCQ. We have used another Queue namely COPYQ to replicate database when a new slave starts so that it is similar to that of master’s database. In this Queue the master would send its database as JSON and the slave would commit the database. We have used kazoo Zookeeper to maintain high availability and fault tolerance. The kazoo creates a path namely slave in the orchestrator and when a new slave starts it creates a ZNODE and attaches itself to the path and when the crash slave API is called, the slave is killed and the watch is triggered in the orchestrator. We have used Docker SDK to run and kill the containers.

## TESTING

The testing challenge was returning of workers list when a slave is crashed and we realized that it was due to the time taken to stop a container and hence we increased the timeout attribute of container\_id .stop(timeout) and then it worked properly.And we initially didn’t sort the pid of workers and later realized it.

## CHALLENGES

The major challenge here was how to replicate the master’s database when a new slave starts and we at last used Rabbitmq’s Advanced Message Queue Protocol to send the database from master to slave in the form of JSON.

## Contributions

Archana has implemented orchestrator, replicating the database to all the slaves and master/slave implementation.

Keerthi Priya has implemented auto scale and fault tolerance using Zookeeper.

Thanusha and Bhanupriya has implemented setting up the Rabbitmq’s and made the report.

## CHECKLIST

|  |  |  |
| --- | --- | --- |
| SNo | Item | Status |
|  | Source code documented | Done |
| 2 | Source code uploaded to private github repository | Done |
| 3 | Instructions for building and running the code. Your code must be usable out of the box. | Done |