17CS352: Cloud Computing

Class Project: Rideshare

Date of Evaluation: 18/05/2020
Evaluator(s): Spurthi N Anjan, Dilip Gurumurthy
Submission ID: 585
Automated submission score: 10

SNo	Name	USN	Class/Section
1	Rithvik Chandan	PES1201700014	H
2	Shreya Maheshwari	PES1201700025	A
3	Rahul Koushik	PES1201700121	F
4	Vishal Sathyanarayana	PES1201700183	A

Introduction

- The RideShare application provides functionality to add a new user, delete an existing user, create a new ride, search for an existing ride between a given source and destination, join an existing ride and delete a ride.
- This project revolves around creating a fault tolerant, highly available database as a service on an AWS EC2 instance for the RideShare application.
- This service has been implemented using master-slave architecture along with Zookeeper (for fault tolerance), RabbitMQ (for communication between workers) within Docker containers.

Related work

- Zookeeper: http://zookeeper.apache.org/, https://www.allprogrammingtutorials.com/tutorials/leader-election-using-apache-zookeeper.php
- RabbitMQ: https://www.rabbitmq.com/getstarted.html, https://www.rabbitmq.com/channels.html
- Python binding for Zookeeper: https://kazoo.readthedocs.io/en/latest/
- Images of RabbitMQ and Zookeper: https://hub.docker.com
- APScheduler: https://apscheduler.readthedocs.io/en/stable/

ALGORITHM/DESIGN

- Set up 3 AWS instances – users, rides and orchestrator.
- Set up containers for users, rides and orchestrator in their respective instances.
- Set up and testing of load balancer.
- All the incoming requests are received by the load balancer and routed to the user's or ride's instance depending on the request.
- The APIs present in users and rides handles all incoming requests. In case of any database operations, a request is sent to the orchestrator instance over port 80.
- The orchestrator instance initially consists of 5 containers – RabbitMQ, Zookeeper, Orchestrator, Master and Slave.
- Eventual consistency is ensured by making use of Advanced Message Queue Protocol implemented using RabbitMQ.
- All read requests are published to the read queue.
- All write requests are published to the write queue.
- Once a write request is processed by the master, a subsequent request is published to the sync queue, in order to maintain consistency between the master and slave workers.
- Once any request is processed, the response is published to the respective response queue.

- A file is maintained which is common to the orchestrator and workers. This file keeps track of all the write and delete requests to aid in the creation of a new worker.
- Auto scaling is managed based on the number of incoming requests to the database service.
- For every additional 20 requests, a new container is spawned. The count of additional requests is reset every 2 minutes.
- If the number of requests reduces, scale down occurs at the same rate every 2 minutes.
- The above task of auto scaling is achieved by using APScheduler – which runs a background thread which calls a job to check the number of requests every 2 minutes.
- Fault tolerance is taken care of by making use of Zookeeper – which keeps a watch on each worker. If a slave unexpectedly stops functioning, this slave is killed and a new slave is spawned. If the master unexpectedly stops functioning, then the current master is killed, the slave with the highest pid is elected as the master and a new slave to replace it is created.

TESTING

- The specification sheet did not mention that we had to implement cleardb API in DBaaS. We overcame this challenge by the discussions on Piazza.
- Created a Postman collection of our own set of requests for testing.

CHALLENGES

- The format conversion while communicating over RPC client in RabbitMQ.
- Ensuring the functionality of queues.
- Small bug which caused ConnectionRefused error in Zookeeper.

Contributions

1. Rithvik: RabbitMQ set up + queue configuration, working of master, scale in/out, integration + testing
2. Shreya: Data replication + sync, scale in/out, APIs in orchestrator
3. Rahul: Fault tolerance using Zookeeper, APIs in orchestrator, load balancer
4. Vishal: RabbitMQ set up + queue configuration, working of slave, scale in/out, integration + testing

CHECKLIST

SNo	Item	Status
1.	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