17CS352:Cloud Computing

Class Project: Rideshare

Date of Evaluation:20-5-2020
Evaluator(s):Usha Devi BG,Sanjith
Submission ID:1144
Automated submission score: 10.0

SNo	Name	USN	Class/Section
1	P Sudhamshu Rao	PES1201701634	B
2	S Thejas	PES1201701621	B
3	Vaibhav V Pawar	PES1201701131	B

INTRODUCTION

- Our project aims to build a Ride share application using Docker, flask,zookeeper rabbitmq,and other programs. The Rides Share application contains many rest api endpoints using which our application can be accessed. For database we use sqlite3.All the flask applications are dockerised. We hosted all our application on AWS. There are three aws instances for users, rides, and database. This project mainly focuses on database. The database component contains three main parts, orchestrator which receives all incoming requests and directs it to appropriate endpoints, slave which serves the read requests to the database, master which writes to the database. The orchestrator slave and master communicate using RabbitMQ. Zookeper is used to ensure fault tolerance. And orchestrator decides how many slaves are required (creates/deletes the slave container) based on number of incoming requests.

RELATED WORK

- https://stackoverflow.com/questions/26861761/why-use-gunicorn-with-a-reverseproxy
- https://stackoverflow.com/questions/38982807/are-a-wsgi-server-and-http-serverrequired-to-serve-a-flask-app
- https://www.javacodemonk.com/part-2-deploy-flask-api-in-production-using-wsgigunicorn-with-nginx-reverse-proxy-4cbeffdb
- https://medium.com/ymedialabs-innovation/deploy-flask-app-with-nginx-using-gu nicorn-and-supervisor-d7a93aa07c18
- https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications -with-gunicorn-and-nginx-on-ubuntu-18-04
- https://www.bogotobogo.com/python/Flask/Python_Flask_HelloWorld_App_with_ Apache_WSGI_Ubuntu14.php
- https://stackoverflow.com/questions/18048318/apache-mod-wsgi-vs-nginx-gunico rn
- https://www.peterbe.com/plog/nginx-gunicorn-vs-mod_wsgi

ALGORITHM/DESIGN

All the requests are serviced through the orchestrator. The read requests are directed to slave container whereas the write requests are directed to master container. When a worker container is started, a ephemeral znode is created which is maintained by zookeeper container.

RabbitMQ's RPC model is used for read/write request communication between orchestrator and worker containers. One RPC model maintains two queues, one for sending the request and one for receiving the response. The orchestrator writes into the request queue and listens to the response queue. The worker containers listen to the request queue and writes into the response queue.

When a worker (slave) container is created, it copies the database from the master. The updates made by master is written (updates to be done) into a sync exchange (RabbitMQ's Publish/Subscribe Model). Each of the slave container creates a sync queue and is binded to the sync exchange. Master writes the updates to database in sync exchange, which is then received by all the slave containers.

Zookeeper keeps track all the znodes (ephemeral) created by the worker containers. All znodes are created in the same path (worker creates znode in this path). Znode has name and value fields. The name and value field contains the pid of the worker container that created znode. A watcher (Zookeeper's ChildrenWatch) is added to the path where znodes are maintained and a function is associated to this watcher. The associated function is triggered when there is a change in the znode present in that path, or a znode is created or deleted. When a worker container is crashed, the ephemeral znode created by that container is deleted. This then triggers the function which starts a new container if required. In this way fault tolerance can be ensured.

TESTING

We tested scaling by sending a number of requests using postman. We tested the fault tolerance by modifying the worker code to finish execution (so that the container stops running). We tested all our rest endpoints using postman, by manually sending requests to loadbalancer. We had problems getting the loadbalancer working in automated testing.We mainly solved any problem by checking the requests sent to our VM by the automated testing program,and tried to figure it out from there.

CHALLENGES

- Understanding how to use docker sdk
- Ensuring fault tolerance of worker containers
- Deciding on model to ensure data consistency among the worker containers

CONTRIBUTIONS

Name	USN	Contribution
P Sudhamshu Rao	PES1201701634	Master/slave worker Queues Using RabbitMQ Master/slave worker Scale out/in Orchestrator Report
S Thejas	PES1201701621	Orchestrator Fault tolerance using Zookeeper Queues Using RabbitMQ Report
Vaibhav V Pawar	PES1201701131	Perform data replica/sync Master/slave worker Scale out/in Report

CHECKLIST

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