17CS352:Cloud Computing

Class Project: Rideshare

Date of evaluation: 19-05-2020
Evaluator(s): K Srinivas
Submission ID: 1433
Automated submission score: 10

SNo	Name	USN	Class/Section
1	Sruthi Madineni	PES1201700051	E
2.	Meghana Nayak	PES1201701339	E
3.	Rachana HS	PES1201701726	E

Introduction

In this project we are building a fault tolerant, highly available database as a service for the RideShare application. We used Amazon EC2 instances. We used our users and rides VM, their containers, and the load balancer for the application. In addition we enhanced our existing DB APIs to provide this service. The db read/write APIs written is used as endpoints for this DBaaS orchestrator. The same db read/write APIs was exposed by the orchestrator. The users and rides microservices no longer be use their own databases, they will instead use the "DBaaS service". Here instead of calling the db read and write APIs on localhost, those APIs are called on the IP address of the database orchestrator. We implemented a custom database orchestrator engine that will listen to incoming HTTP requests from users and rides microservices and perform the database read and write according to the given specifications.

Related work

The document shared on piazza for project and the tutorials mentioned in it.

ALGORITHM/DESIGN

Rabbitmq-

Here we have four queues which are readq,writeq ,syncq and responseq. The callback function is the one which uses writeq everytime a worker consumes something from the writeq. on_request function is the one which uses readq,Everytime a worker consumes from the read queue it takes care of what should be done with the messages.Only master uses writeq and slave uses readq. So when we do a write,write api puts message in the writeq,worker i.e master consumes from the writeq and callback function is called.Callback function then writes it to the masters db.Then on writing same message is out into the syncq by the write api.On receving this message from the syncq "sync" function is called in slaves which writes the message data into the slaves db.Hence synced.

Docker SDK-

Here we have used orchestrator to create master and slave containers.Initially orchestrator creates one master and one slave.Here both of them will run the same worker code.Depending on the number of requests being sent we increase or decrease the number of slaves.Every time a slave is created it has to get a copy of the db from the master.So as and when we get any requests we store those messages in a global lists.When new slave is created we call the get_db api which copies the db to the new slave.We have a job function which is used for auto scaling.This function is called by the schedular library every two minutes.This schedular checks the number of requests every two minutes and scales up or down depending on the requests.Crash api is used to crash the slave with max pid.It returns the pid of the slave which is crashed.

Zookeeper-

Zookeper is used to maintain high availability.Here we use it to watch on slaves and if one of the slaves crash,a slave watch event is triggered and it starts a new slave and data is asynchronously copied onto the new slave.

TESTING

-Had problem in auto scaling during the submission.Slaves created where not getting killed during scale down.Had to change the logic during testing.

CHALLENGES

-Understanding the concepts and tutorials in the starting stage of the project.

-Port issues while running locally.Hence had to switch to instance to test out everything.

-problem in understanding why master and slaves were getting exited while running sometimes

Contributions

Since we are three in a team each one of us took one part of rabbitmq,docker sdk and zookeeper.Sruthi(rabbitmq),Meghana(docker sdk),Rachana(Zookeper)

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