17CS352:Cloud Computing

Class Project: Rideshare

Date of Evaluation:16/05/2020
Evaluator(s):
Submission ID:259
Automated submission score: 10

SNo	Name	USN	Class/Section
1	Jnanesh D	PES1201701822	B
2	Mayur RB	PES1201700714	B
3	Sujay Gad	PES1201700177	B
4	Avi	PES1201701116	B

Introduction

A fault tolerant Scalable and high available DBaas for rideshare. The db read/write APIs will now be exposed by theorchestrator. The users and rides microservices will be using DBaas Service. Instead of calling the db read and write APIs on localhost, those APIs will be called on the IP address of the database orchestrator. Zookeeper is used for High availability and for Scalability orchestrator is used. Basically orchestrator is a flask application.

Orchestrator:-
The orchestrator will be a flask application. It will be serving the endpoints 'api/v1/db/read' and 'api/v1/db/write'.

ZooKeeper:-
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications.

Scalability:-
The orchestrator has to keep a count of all the incoming HTTP requests for the db read APIs. For now, we assume that most requests are read requests and hence the master worker does not need scaling out. The auto scale timer has to begin after receiving the first request. After every two minutes, depending on how many requests were received,the orchestrator must increase/decrease the number of slave worker containers

Related work

- Rabbitmq - pika tutorial 1 , 2 , 3 , 6 https://www.rabbitmq.com/tutorials

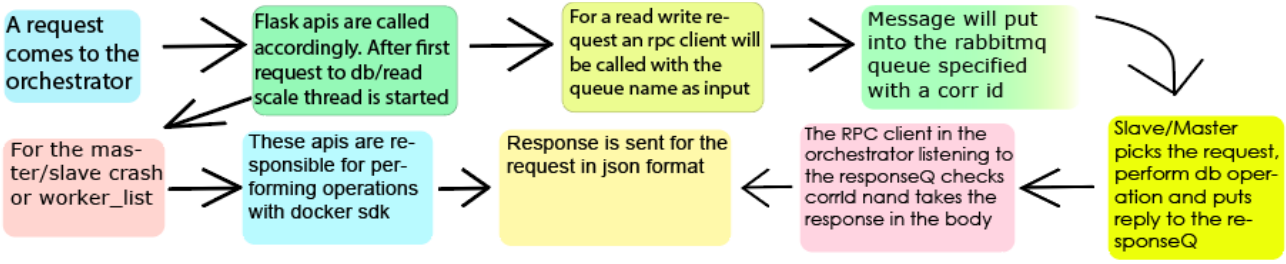
- Kazoo basic usage https://kazoo.readthedocs.io/en/latest/basic_usage.html

- Docker networks , volumes
  https://kazoo.readthedocs.io/en/latest/basic_usage.html
  https://docs.docker.com/storage/volumes/

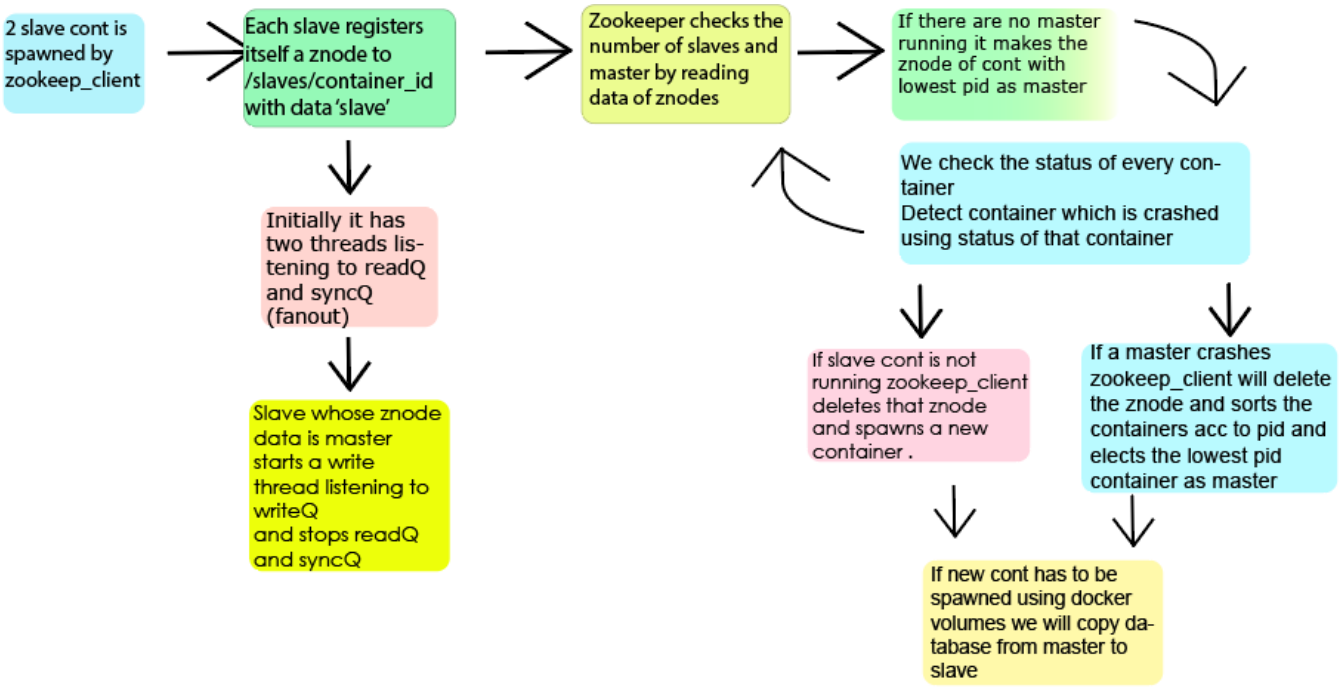
- Docker sdk - https://docker-py.readthedocs.io/en/stable/
- https://stackoverflow.com/questions/28550140/python-and-rabbitmq-best-way-to-listen-to-consume-events-from-multiple-channel

ALGORITHM/DESIGN

Our design of flow of message is illustrated below



Fault tolerant design



TESTING

Scaling of the docker-containers and the data consistency was taking time in the beginning.

Online submission we didn't had issues as we got full marks in first submission.

CHALLENGES

- Exposing the pika blocking connection inside Flask
- Listening to multiple queues in worker file
- We had issues in data replication of the new worker
- It was difficult doing leader election . Converting a slave to master
- New containers connecting to the rabbitmq and zookeeper
- Using the znodes

Contributions

Jnanesh D – Fault tolerance using zookeeper, Leader election , Data replication to new worker .

Mayur RB – Making the rpc client and server for communication through rabbitmq and AWS deployment

Sujay Gad – Scalability, setting up multiple rabbitmq queues in worker and data sync

Avi bansal – Database operations in worker, changes in users and rides, new APIs of orchestrator

CHECKLIST

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

Github Link : https://github.com/JnaneshD/DBaas (private repo)