Cluster and Cloud Computing Assignment 2
City Analytic on the Cloud

Zhaofeng Qiu 1101584

University of Melbourne — May 25, 2020

1 System Design and Architecture

The architecture of our system is in well designed, which is aimed at providing both a beautiful front-end web application and a RestFul API server with high availability, high scalability and fault tolerance. The overall architecture is shown in Figure 1.

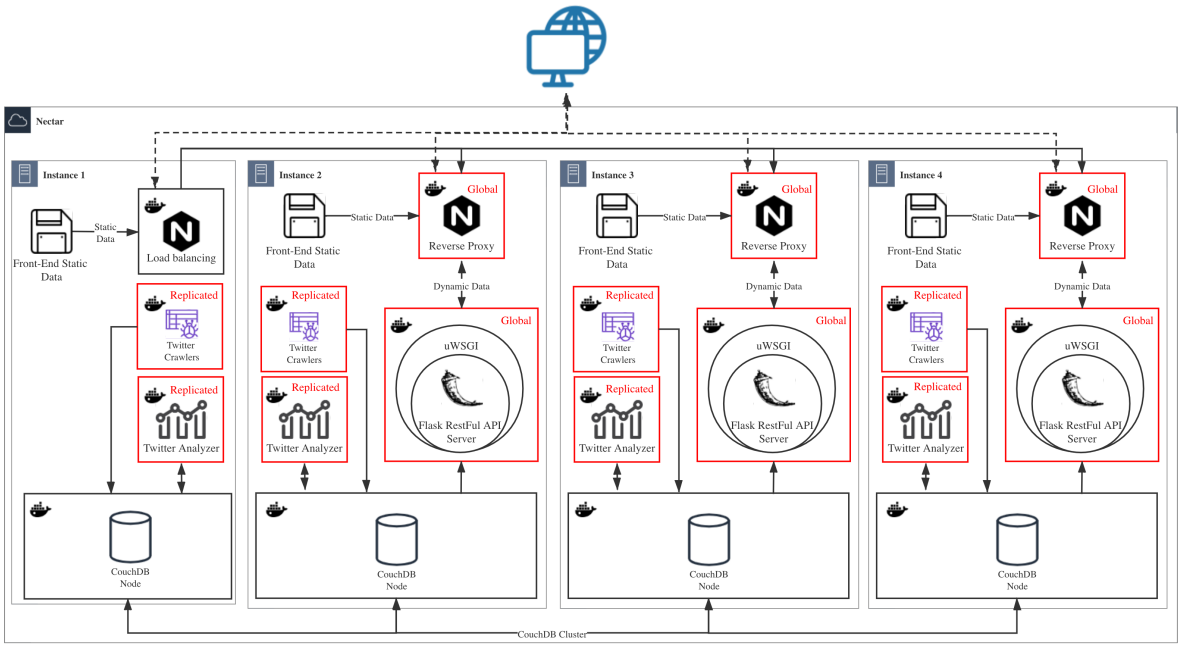

Figure 1: Velocity Re-estimation

1.1 High Availability

Through in-depth thinking and repeated practice, we continue to optimize our architecture to achieve a high availability with limited resources. We mainly use the following method to improve the availability of our system.

Firstly, we run a NginX server in our first instance to provide Load balancing. For requests that need to access the dynamic data in our database, it would pass the request to the other three server averagely. In this way, we can efficiently distribute the incoming network traffic to our backend servers in the cluster. When it comes to the situation that the first instance may crash, users can still access our application using the IPs of the other three nodes. But this solution is not good enough and is contradict to high avilability. We had tried to use two NginX server with a software called Keepalived to provide a more fault-tolerant system. However, we found out that we are not allow to use floating IP in Nectar. Since we cannot set a

virtual ip for our NginX servers, the only way to handle this issue is to let the other three nodes to keep the full functionality of our application. User can access our application through any IP of our instances, but only the IP of the first instance would provide load banlancing control.

Secondly, we use docker swarm to manage part of our services. We set three swarm manager nodes in our cluster. Any instance with a ID bigger than 3 would be set to be a worker node. A single leader is elected from the three swarm manager nodes to conduct our orchestration tasks. Our manager would also run the services as worker nodes. According to [1] and Swarm Admin Guide[1], the number of our manager nodes is set based on the following facts:

- More managers would resulte in a longer election for a new leader when one goes down. The swarm is in a read-only state which means that no new replica tasks can be launched and no service can be updates. No auto-recover can happen during the election.
- More managers would require tighter management of resources to prevent manager starvation.
- Less managers would increase the probability of all managers going down.
- To support manager node failures, the number of the managers should be a odd number and should bigger than one. In our cluster, we can only provide four nodes. So, setting the number of managers to be three in our system is our best choice. In this way, our system can provide fault tolerance that one of our manager node can crash.

1.2 Dynamic and Static Separation

When the first instance is alived, we

References

[1] B. Fisher, "Pros and cons of all managers as workers in swarm," https://stackoverflow.com/questions/48853473/pros-and-cons-of-running-all-docker-swarm-nodes-as-managers, 2018.

[1] https://docs.docker.com/engine/swarm/admin_guide/