CAB432 Assignment 2 Individual Report

Name : Yonathan Cahyadi

Student Number : 10149953

Partner : Yanmei Zeng (10307389)

Statelessness, Persistence, and Scaling

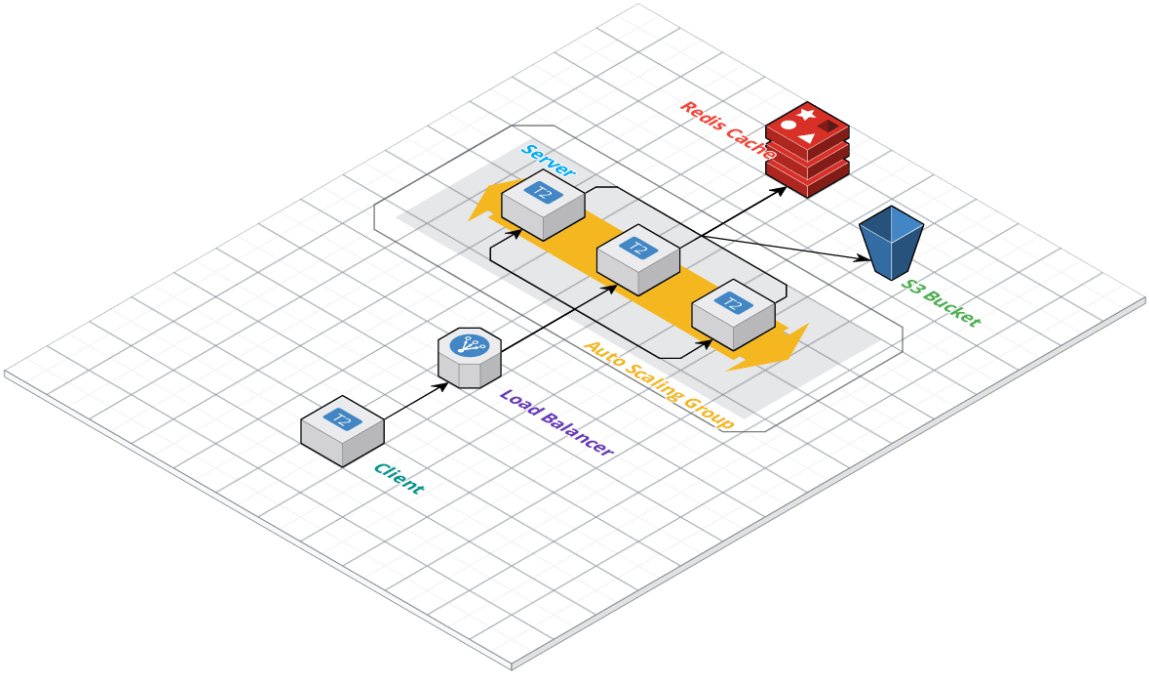

Figure 1: Architecture Diagram

Mechanisms for handling application state and their robustness when instances fail

As for handling the application state in this project we use Redis Cache and AWS S3 Bucket separated from our server, therefore our server did not store any data making it a stateless server. For our Redis Cache we have 3 Node available, 1 is the Primary Node and the other 2 acts as Replica/Backup in case our Primary Node is corrupted or fails. Another example would be, in case our AWS S3 bucket got corrupted or failed and we have data loss, we can recover our data by using the Redis Cache, assuming that the data that we loss is still recent and still stored in Redis Cache.

Compromises from true statelessness and the reasons for these choices

For this project we compromise we make is happened on the client side. Our client side need to keep user access token in its session storage. The reason is, so that when the user refreshes the pages they don't need to login back again, because the access token received after login is kept in the session

storage. As for our server, we manage to achieve true statelessness as our server did not keep track of user details, neither its received data nor processed data. All the processed data is stored in the Redis Cache and AWS S3 Bucket.

Persistence choice and their suitability for this application

For the persistence aspect of this project, we are using Redis Cache for short term storage and AWS S3 bucket for long term storage. We think that it's a good idea to store our processed data in the Redis Cache and AWS S3 bucket, the reason is being the data that the server need to process and received from the APIs is large, therefore in case the data is not stored in Redis Cache or AWS S3 bucket, our server response time is slow ±5000ms. By saving the processed data in the Redis Cache and AWS S3 Bucket we manage to achieve response time of ±2ms. By saving our processed data in Redis Cache and AWS S3 bucket, we also increase our level of persistence, for example in case of failure we still have data stored in either AWS S3 bucket or Redis Cache.

Better Alternatives regarding services used

Any alternatives that we might do would be to implement MongoDB database. This database would store both the processed data and static data that we received from the APIs. Example of static data that we received would bet he song lyric, we can store the song lyric based on the song title and artist name, by doing this we could lower our Network Out and In traffic we can also get a better response time, cause we don't need to wait for the lyric from the API. Storing processed data, also can add a layer of persistence in our project in case our AWS S3 bucket failed.

Approach to application scaling, and the suitability of the metrics and the thresholds used

As for the scaling policy of this project, we decided it to be based on CPU utilization, Network In, and Network Out. For the thresholds, we decide that our server would scale in case of CPU utilization is above 50% or Network In or Out is above 10MB. The reason being, the data we need to process is large in case all of the song we received had lyric, for example the song we received is 50, first we need to send 50 request to get the lyric and if all of them had lyric, assuming that each song lyric had ±300 words our server would need to process ±15000 words for each client request, this means our server also need to receive a lot of raw data, send a lot of processed data to the client and have enough CPU power to process all of it.

The Global Application

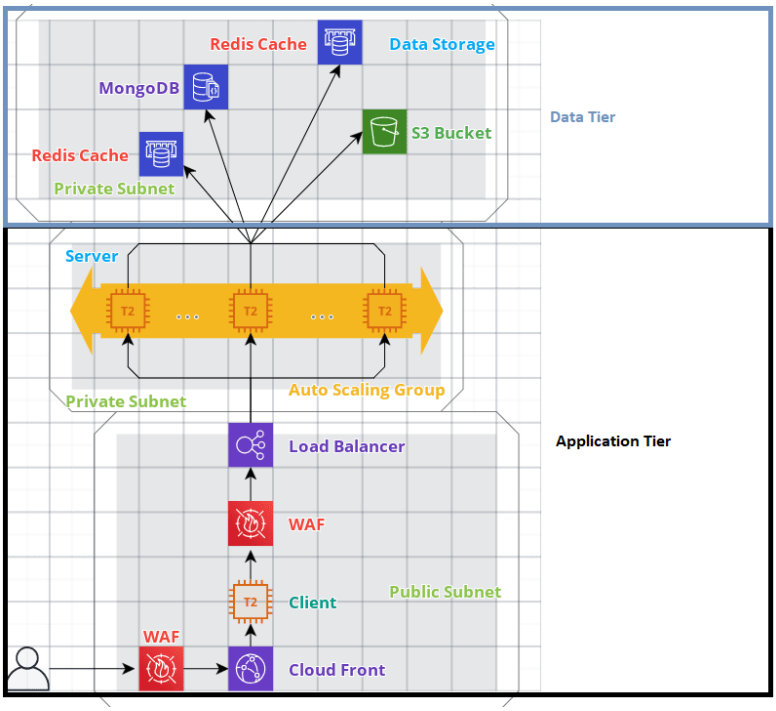

Figure 2: Revised Architecture Diagram for Global Scale

For global scenario, our app needs to add some other service for it to work efficiently. First, we need to add some firewall, such as WAF. We also need to add Cloud Front for edge caching, the edge caching will be some static resource, such as website logos. Another thing we need to do is separate our service into private and public subnet, from the diagram above, I think putting the Data and Server in Private Subnet is making our application more secure. We also added MongoDB and caching service for MongoDB, the purpose of MongoDB is for storing some information that is static or rarely change, for example in our application case it would be the song lyric. We can store the lyric of song based on the song title and artist, so if we have searched those song before we didn't need to request the API again, rather we can look at our MongoDB or the Redis Cache for MongoDB. This would speed up our server response time. We also can make our Auto Scaling Group generate more instance, currently our Auto Scaling Group is limited to 3 instances, but with enough resource we can add more instance. Another thing we need to change is our Scaling Policy, for convenience and faster response time we can lower our Scaling Policy threshold.

Cloud Security

Some security flaw that our application has currently, is all the data transfer is not encrypted (http), data such as access token can be stolen over the network. Our application also vulnerable to DDoS attack and XSS attack, the reason being currently our architecture doesn't have any firewall configured. Another flaw is our service that we used currently still configured with default port and setting, for example our Redis Cache is configured at default port which is 6379. Our Client and Server also lack logging management and configuration. This problem can be solved by installing and configuring a logging service, encryption and firewall.