CAB432 Assignment 2 Individual Report

Name: Joshua Paterson
SN: n10193197
Partner: Benjamin Semple N10533885

Statelessness, Persistence and Scaling

The mechanism for handing the applications states is the auto scaling feature seen in figure 2 below that directly assigns EC2 instances to the images or AMI's that the server is currently employing to satisfy user requests. Using autoscaling to handle the EC2 instances increases the robustness of the system as the autoscaling has a default feature that performs EC2 status checks. This will increase the robustness of a system as if an instance fails a status check the auto scaling will consider a EC2 instance unhealthy and replace it. This increases robustness as the auto scaling has a pool of instances to draw from and having a single broken instance will not prevent the function of the server.

The application isn't truly stateless as it does retain information from previous user interactions with the application however this information has no connection to individual users or will not cause a different response to users based on it. As seen in figure 1 the application can avoid doing any processing by checking for a s3 bucket for the desired information to reduce processing time. This was done to improve the overall responsiveness of the server.

As said in the previous paragraph some data is stored on the server in the form of a s3 bucket. If a query is not stored in memory the server will process the query, then store it for 60 seconds. This was done as having already prepossessed data can reduce the application processing time and utilization drastically. The reason for the minimal amount of time was to increase response time of popular/in demand query tags while not having the potential to use all memory on the server as well as have current information to users.

Alternatives which could provide better results are not possible to the fact this application does not need persistence to function and the use of persistence for a long period would prevent the ability for live and current data. The inclusion of local storage on user's systems such as Redis could provide better response times however the benefit was so minimal as after a short amount of time the data would need to be updated. It was chosen to not be implemented within this application.

When flooding the server with queries using the application Postman it was discovered that there was an increase in CPU utilization from the server and therefore a possible metric to scale the application by able to utilize all instances available within the auto scaling group.

Possible alternatives to CPU usage include network in utilization and disk reads. Using network in would have been a very effective possible scaling metric as it was discovered that there was very little CPU utilization this is likely due to the limit on tweets from the twitter API and the use of storing already process queries however it was seen that the network usage of the server would vary dramatically and therefore the metric used could be used to scale the application. Doing this allowed for more queries to be processed during times of high traffic therefore increasing the responsiveness of the application. The second option of disk read speed as during the higher traffic periods the s3 bucket would need to be access more often which could lead to increased performance from the server.

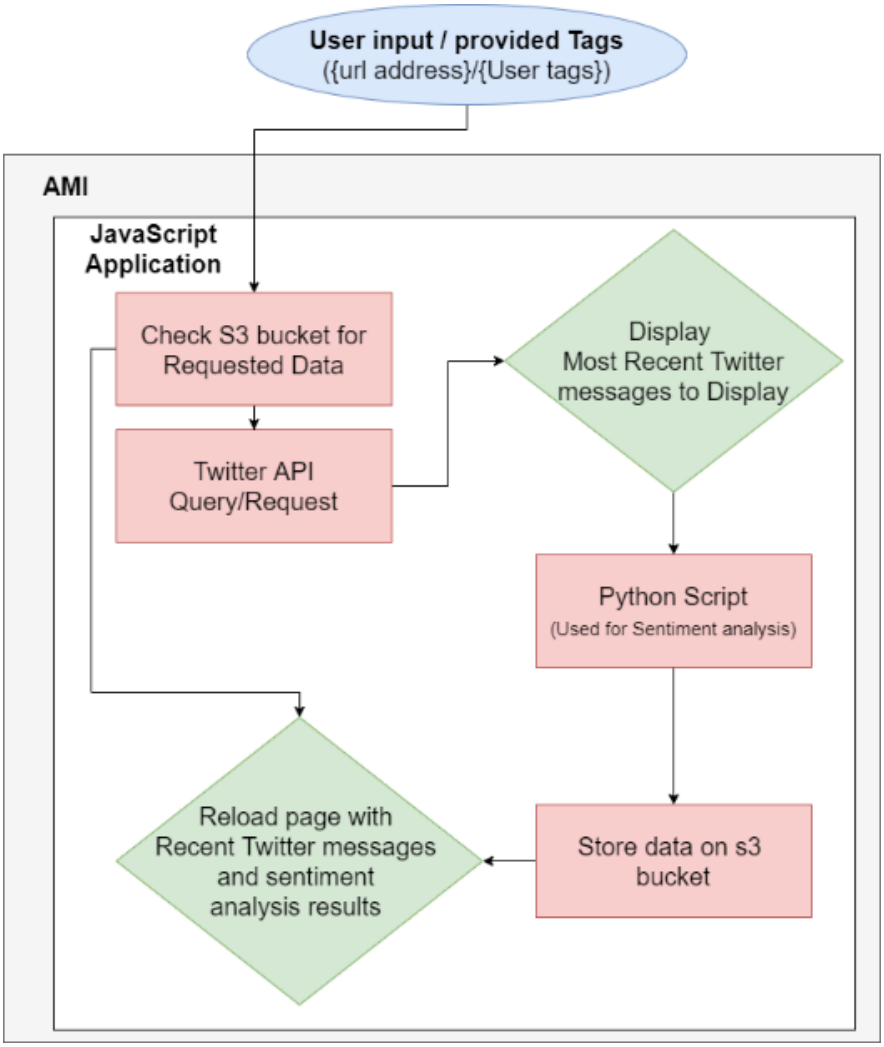

Figure 1: Application Data Flow Diagram

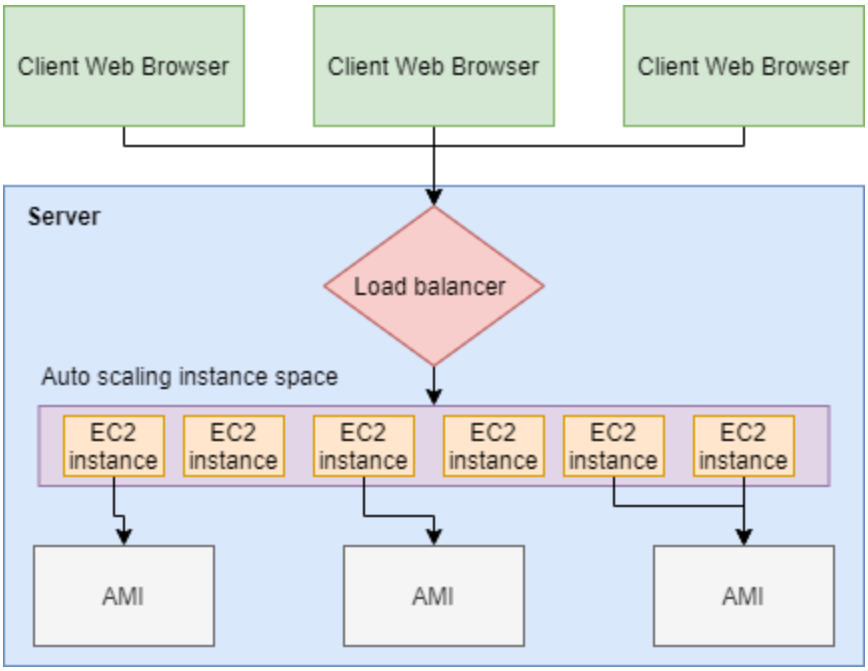

Figure 2: AWS Scaling Diagram

The Global Application

The persistence choices made for the application could have some major issues when scaled to a global level. An issue is present as storage of data on the server is determined by traffic on the server this could cause the amount of storage on the server to rapidly increase. This method of storing data would make the website highly susceptible to DDoS attacks as it could be easy to fill up the memory space on the server. However, with a large enough memory space for the server these issues would be limited in occurrence as they would only occur in periods of high traffic as storing the data requires only a small amount of space. It should be noted that this would cost more to maintain such a large memory space especially if data was kept in closer location to users and distributed to many storage sites however this issue could be mitigated by only limiting the number of stored requests on the s3 bucket, but this could limit the faster response time of some requests and limit performance. The application at high scales will maintain the same level of stateless as the original application providing the same functionality. In times of high traffic, the response time will likely be determined by the instance pool size the auto scaler has access to. If the instance space is too small there could be significant latency issues. The effect of eventual consistency could increase response times for user wherever they may be however this would increase the cost as storing the same data at multiple sites would multiply storage cost by the number of nodes. Additional cache levels would not provide any benefits at higher scale for the application. All data that is sent to a user is data that must be processed before being sent to the user and will be unique by the users own provided tags. The website doesn't have any common information being sent to all users such as images. No data can be stored closer to users that would benefit all users. The scaling metric and polices would not need to be changed but would need to be scaled up horizontally or increasing the number of instances assigned to the auto scaling group and would not work vertically due to the current version of the applications order issues and increase the instance space horizontally could be a potential fix for this issue.