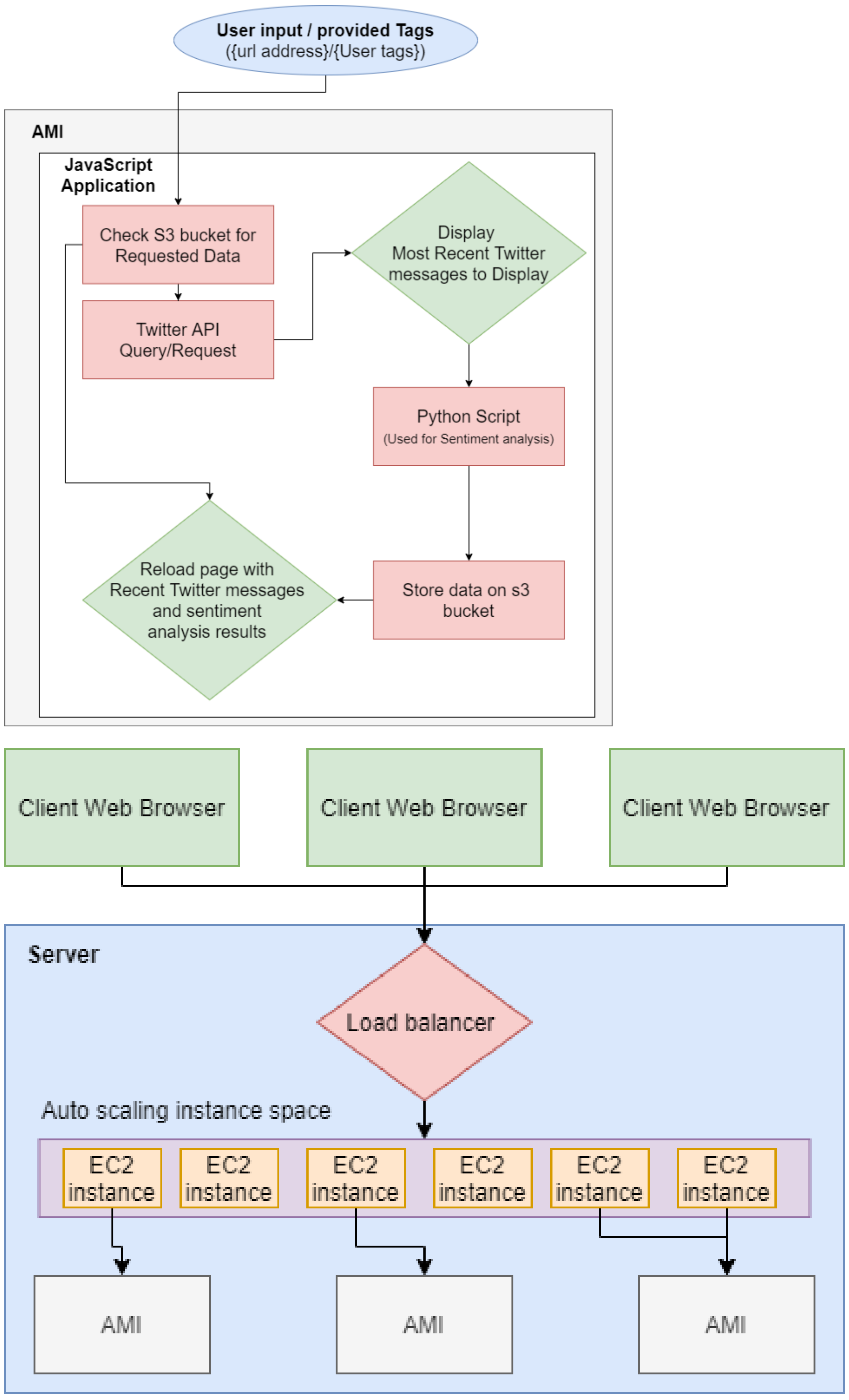
CAB432 Assignment 2 Individual Report

Name: Benjamin Semple
SN: N10533885
Partner: Joshua Paterson

Statelessness, Persistence and Scaling



User input / provided Tags
({url address}/{User tags})
AMI
JavaScript Application
Check S3 bucket for Requested Data
Twitter API Query/Request
Display Most Recent Twitter messages to Display
Python Script (Used for Sentiment analysis)
Store data on s3 bucket
Reload page with Recent Twitter messages and sentiment analysis results
Client Web Browser
Client Web Browser
Client Web Browser
Server
Load balancer
Auto scaling instance space
EC2 instance
EC2 instance
EC2 instance
EC2 instance
EC2 instance
EC2 instance
AMI
AMI
AMI

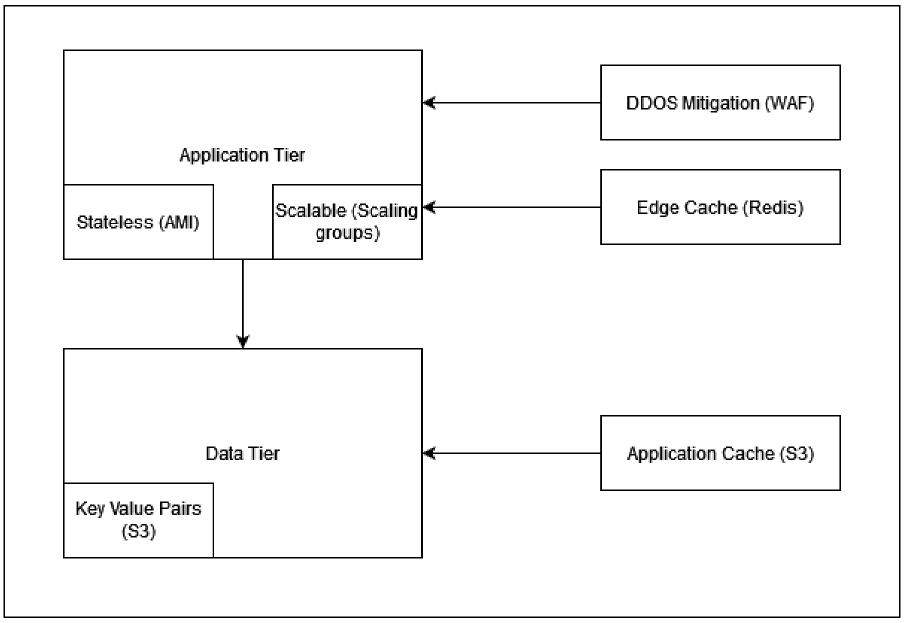
The application state is handled by the server running and also the S3 storage bucket system. The initial state which is checked for is the S3 bucket. If valid data is found in the S3 bucket then that is what is served as the state, if it is not found then it runs the computation and returns that to the user. If the S3 has a bug or fails then the error is caught and the computation occurs instead and that saves the app from a crash. If the computation fails, for some reason then the failed computation is caught by error handling and the application offers up a page showing that the results are not valid, saving the app from a crash.

The application does not achieve true statelessness. There are several places where the application is not stateless. When the application runs and another request from another source then the app serves up whatever it served up to the other source during loading, the other user only gets what they searched if they search while a different user is not also causing the app to perform computations in order to receive a page-based ion their query. Another way the application is not stateless is any data from one user that then ends up in the database is then accessed by any user who runs the same query and the data in the bucket has not expired yet.

S3 was chosen for the application to provide persistence because it is easily scalable and handles JSON inputs well. JSON is what was used to present data on the frontend so as a result using S3 makes sense to store the data. The automatic scaling of S3 also means that storage can be implemented and then no management or maintenance is required, and the app will correctly scale and store all of the data. Hosting another database within the application itself may been a better idea. A popular and preferred database system which allows for this is Redis. Redis stores JSON directly and is also automatically scalable, meaning that it has all the advantages of S3, however it would be on the application which would mean it would run faster. Ideally, we would have both Redis and S3 used to make the program run on two caches, with the Redis cache remaining stateless with a combined S3. This would make the application run more efficiently. It would mean the Redis cache could cater to a single user and provide them with fast responses, with the S3 providing a faster response to a different user who had not searched a query previously, however the query has been searched so S3 caching it may make the response faster for a different user.

The metrics which were used were CPU usage. The reasoning for this is because the CPU usage scales as the processing being performed increases. This appeared to be an adequate and suitable metric to use as the CPU usage increased when the server was flooded with requests from postman then the CPU usage decreased after the requests stopped. An alternative metric which could have been used to create scaling would have been the network in metric. The number of bytes into the VM server by the network in would accurately measure the number of people connecting the server. From there the number of instances would be scaled based on the number of incoming requests and as a result be more appropriately scaled to match the number of users accessing the site at once

The Global Application



The S3 bucket system used in the assignment is appropriate for large scale because it is a scalable database option. Using the S3 bucket the way we used it also improves the site efficiency the more people who connect. To the server the more the S3 will expand and create more storage allowing for as much scaling as Amazon has space. The load balancing and AMI system is not stateless in it's current implementation and could not safely be scaled up to take millions of concurrent users. In order for this to be achieved, the application would need to achieve statelessness to a better degree.

Eventual consistency within my application is achieved by the S3 bucket storing a cached set of results which are accessible to all users. This results in the application being sped up overall because it means a lot of computation is saved because the results are already ready, and the computation has already happened. It does however mean that the application is no longer strictly stateless. However this is not an issue as it does not reduce the quality of the website or its ability to scale.

My application does not have an edge cache, and this is an oversight. An edge cache can be created using Redis, they have a specific set up which can be used specifically for edge caching which would be ideal for this setup. This is because Redis would work well with the current implemented aspects of the web application because it uses key value pairs, the same as the JSON on the front end. The edge cache on the web application would hold the results for a query the individual user had looked up. This is because the current caching using on S3 which would become the application cache and the edge cache needs to be closer to the user. The edge cache should be accessed first if applicable and be fast. Redis would be running as an edge cache and thus has a fast response time however S3 would require another call to data on another server. This takes extra time and data transfers and the edge cache. The edge cache is meant to be fast responding and not require extra transfers which makes using Redis for a fast responding cache an ideal solution here.

Currently the application uses a 20% CPU cutoff. A computer can function at a much higher CPU usage than 20% and still function. If needed, the threshold for the scaling could be increased theoretically all the way to 100% depending on the cost limits. The less we have to increase the better, as a CPU which has a high amount of its capacity being utilized will run slower and as a result slow down the application for all users. So because of this, for a best outcome for the user, the application should use as little CPU as possible while staying within budget.