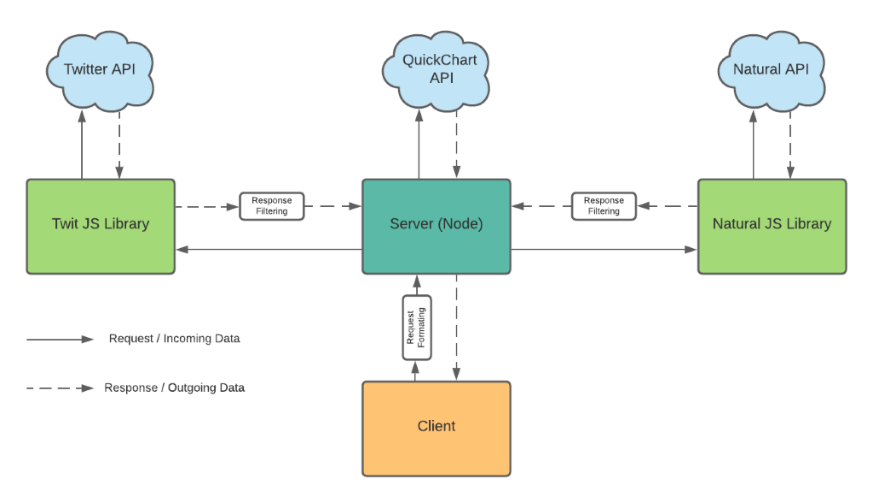
CAB432 Assignment 2 Individual Report

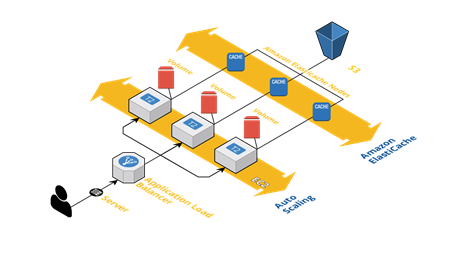
Name: Callum Longman

SN: n10218858

Partner: Chu Weng Law

## Statelessness, Persistence and Scaling (8 marks)





If an instance were to terminate without warning, AWS autoscaling group should relaunch the instance to reach the desired capacity of instances. If multiple instances were running, the use of ‘pm2 start’ means while the instance is active so to is the application, without the need of running ‘npm start’. The use of persistence (that will be described in more detail below) will store queried responses and therefore when an instance terminates and restarts both the cache and long term storage will be intact for the instance to continue to use. This applies to the cache as well because Elasticache was used instead of a local Redis cache. Elasticache provides a cache that can be accessed by all instances and the data stored is shared with other instances.

A stateless application does not store queried variables/responses on the server, this is the case for our application through the use of persistence. The application is robust and does not require transient states as data is not passed between instances, merely stored in a cache or long-term storage where it can be accessed by other instances. If an instance were to terminate after starting but prior to completing a query the response would not be store, which is desirable as the data could possibly be incomplete or corrupted.

Both choices for the levels of persistence are adequate choices. S3 is a very easy service to use with results being expected when in use. For the current state of the application and expected usage, S3 will be a good enough choice as it is extremely unlikely that our application will exceed data size limits stated by AWS. As for the cache, Elasticache was used. Elasticache was chosen over a local Redis cache to improve latency when the application is scaled and to help mitigate data loss if an instance were to die. Elasticache offers the above uses with minimal changes to application code. The duration that data is stored in the cache was set to 1 hour, this should provide ample time that if a user were to query again, the response will be extremely quick but also the duration is not too long that the cache becomes overfilled with old data. Old data refers to the fact that current twitter trends yesterday may still not be relevant.

Some alternatives that could be chosen are related to how the application scales. Currently the application scales when Network In exceeds 1MB. It was observed that when 50 requests were executed by postman that the Network In quickly reached peak and the application increased the number of instances from 1->3. Although this handled the issue of increased requests to the server, I believe under normal usages of the application is should scale one instance at a time and should only jump to greater instances when a more aggressive approach to scaling is necessary such as when a DDOS is occurring.

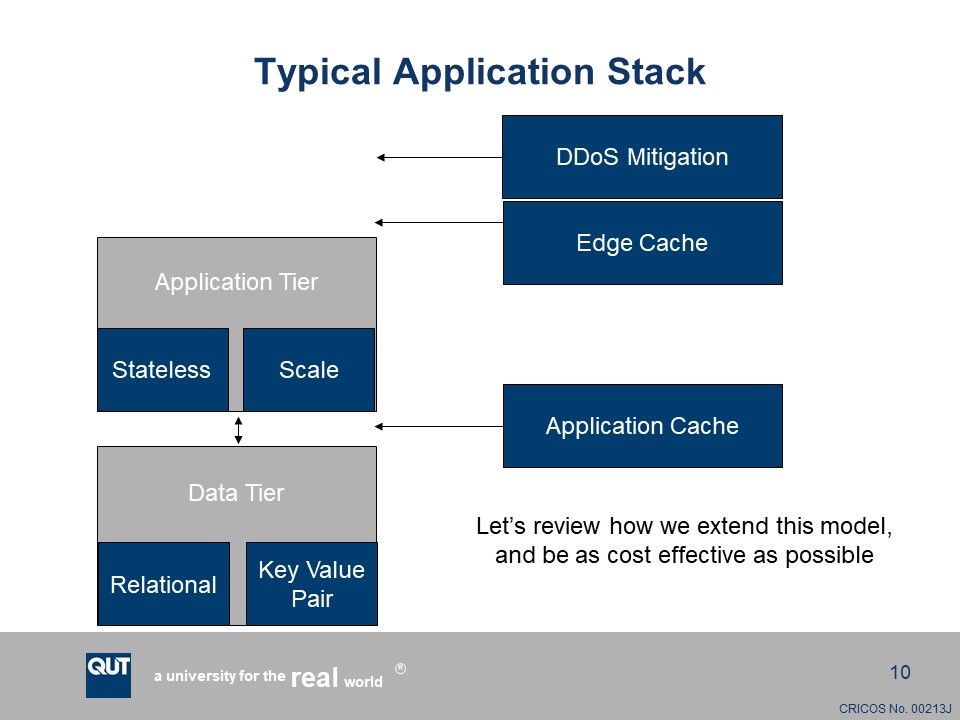
The planned approach of application scaling was to begin with the application running under 1 instance, then to manually scale to multiple instances with increased load, and eventually have auto scaling enabled. The metric used was Network In, this was based off the /search page of the application and not the ‘homepage’ where the responses are shown to the user. A threshold of 1MB was chosen but could have been lowered so the auto scaling had a faster response time when more than usual queries occurred. The maximum number of instances was set at 3 but this could have been increased further to accommodate more than 150 queries, as tested with postman. The planned approach was not closely followed, as the stage of manual scaling was skipped. Instead the use of one instance with 50 postman queries was used to get a ball park figure of the effect of that many queries on the server.

I believe the Network In metric was the correct choice for our application. CPU usage as a metric covers a very broad spectrum of possible issues, for this reason is a great choice for many applications. Upon testing the application under stress with multiple queries in quick succession, it was observed that CPU usage did not change significantly, but Network In had a large, noticeable peak once the queries began to come in and be executed. With the observed peak, it was easy to choose a threshold value that would guarantee proof of scaling. The issue being, proof of scaling does not equate to a good choice of thresholds and metrics, merely showing that under specific circumstances the application will scale to respond to increased queries.

## The Global Application (12 marks)

*This question requires that you reimagine your application at a much more professional scale. You have now managed to establish a global user base. You have thousands and even millions of customers and you now successful cater to their needs. You are going to tell us about this new system and the changes that you have made.*

*You will begin by reconsidering your architecture through the lens provided by Iain’s Lecture 9 on Architecting for Cost (cose to designer $, cost to user time). This overall cloud application stack is a particularly good starting point:*



*You should review your application and consider how a globally successful version of this app might work, and use the framework provided in Iain’s lecture to produce a revised architecture diagram. Some of this will be really obvious, but some of it won’t, and that is where the discussion comes in. You might want to consider the following questions:*

* *Are the persistence choices made for the assignment appropriate for this new scale? Do they maintain statelessness? Do they allow you to manage the needs of your users? Do they only work well when you don’t have to consider more significant latency?*
* *Is there any effect on your ability to scale to very large pools?*
* *What is the effect of eventual consistency on your application?*
* *What is the split between relational and entity-based storage?*
* *Do you need additional cache levels? What is the role of the edge cache in the global version of your application?*
* *What sorts of changes might be needed to your scaling metrics and policies? You may assume that you have a greatly enhanced budget, but it is not unlimited.*

*DDOS mitigation*

*Edge Cache*

* *Consistent high speed/low latency no matter your location*

*Cloud Front*

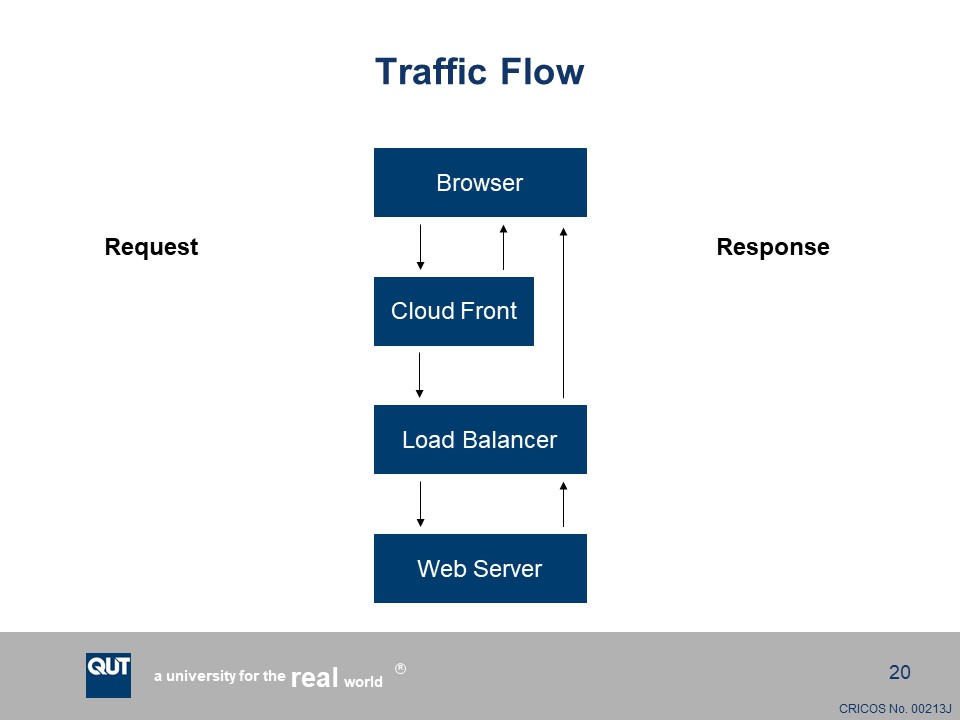
* *Data that doesn’t change much (images) is saved so it doesn’t need to keep querying it or scaling if multiple people are searching for it*

*Collect logs and user data (queries per second, origin of queris)*

* *Don’t need to retain lots of data of it*

*If DDOS scale out aggressively*

*The other image that you might find useful as a prompt is this one:*



*Think about the traffic flows in your application when you are considering the role of caching and the application responses. I strongly recommend that you review Iain’s lecture in preparation for this task.*

*One final series of questions: you have also had a lecture on cloud security. Briefly – a single paragraph will suffice – give us your thoughts on the risk presented by your application. Here we are interested in what the application does – we understand that the one you have produced for this assignment is likely full of security vulnerabilities. Tell us now about what you are doing, and the sorts of security and privacy challenges that this might present at a grand scale, and how you might mitigate them.*

*Your responses to the second task can use up the rest of the three pages you have available, but they don’t need to be that long. If you have something to say and can say it quickly, do so.*