CAB432 Assignment 2 Individual Report

Name: Eric Zhang
SN: n10319191
Partner: Matthias Eder

This report template uses similar conventions to those we used for Assignment 1 and the group task for Assignment 2. However, our requirements are more focused and there will be a strictly enforced limit of 3 pages - including diagrams and the heading and student information above. You should use only two diagrams – your overall assignment architecture and an alternative version suited to a global scale cloud application (we will provide more detail on this below). These diagrams should occupy no more than half a page in total, leaving one and a half pages for actual text. This is more than enough. The marking is based on the quality of the analysis and not on its length.

This individual task has two parts.

*In the first, we want you to analyse your assignment architecture with respect to statelessness, persistence and scaling. Note that this is not about **justifying what you did in the assignment.** Here we want you to demonstrate your understanding of these key concepts. We expect you to look at your work with fresh, analytical eyes and to make some judgments about it. The marking is based on the quality of your arguments and not on how well you have designed and built the system. To answer this question you should draw on our coverage of statelessness, persistence and scaling in the unit materials since week 5. We expect that there will be some overlap between the analyses within a pair, but we will be marking them one after the other and the same marker will deal with both, so please remember that this **must** be individual work.*

In the second part we want you to consider your application and your architecture as it stands, and tell us how you would change the architecture to manage a high traffic, global version of the application in a similar domain, serving a similar purpose. This is a slightly artificial task, but here we want you to assume that the application does much the same thing, but that it does it with world-scale levels of traffic and data exchange. Here we expect you to supply a revised architectural diagram and to highlight the changes that become necessary at the revised scales. This question will draw further on your understanding of statelessness, persistence and scaling, but here we are especially interested in the principles discussed in Iain's lecture on Architecting for Cost.

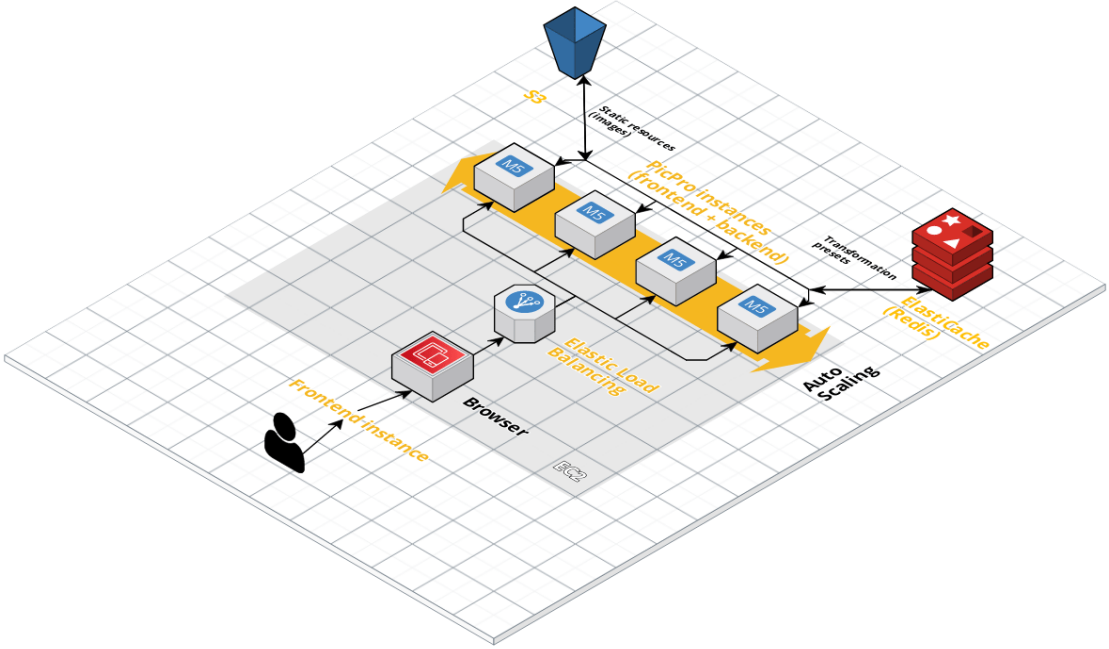
The different sections are discussed below. The task will carry a maximum of 20 marks and this will be scaled down to 10% of your final grade for the unit.

Statelessness, Persistence and Scaling (8 marks)

The application does not rely on any form of state and can be considered stateless. Any input uploaded by the users, which would be images and presets for this application, are stored in persistence services that are shared across all instances, as shown in the diagram below, so any data is easily retrievable if instances do fail. The only information that needs to appear during the start up of the application would the names for stored images and presets from S3/Elasticache, so there aren't any local, unrecoverable resources

S3 was chosen to store the images uploaded from the application due to how large and flexible it is, as users may upload several original and transformed images in quick succession. S3 can scale alongside the data stored in it, making it very suitable for large amounts of images. Elasticache was chosen to store the presets that contain the transformation values due to its speed. As these presets are stored as JSON objects, they do not require the large flexibility in storage scaling that S3 boasts, and need to be accessed quickly, which Elasticache provides very fast access time with very little latency. An alternative for S3 could be GCP Cloud Storage, as it follows many of the same conventions of providing flexibility with storing data, while an alternative to Elasticache could be Memcached, which is also better with scaling in case large amounts of presets or other types of potential data need to be stored in future versions.

The scaling followed a similar pattern to the example shown in the scaling practical, increasing to the necessary instances. Traffic is fed to an Elastic load balancer, which will distribute requests and responses across multiple instances depending on how much scaling is needed. The scaling policy in the Auto Scaling Group is set to use the average CPU utilization as the scaling metric and will scale out at a threshold value of 50%. A Simple Queue Service (SQS) could be considered as an alternative to distributing traffic, as it can still scale elastically and eliminate some overhead for reliable and secure data transfer



The Global Application (12 marks)

Scaling could be increased to include more instances so that auto scaling can keep up with increase in traffic, the below diagram shows an example of what it could look like when the instances are doubled, another Elastic Load Balancer to distribute traffic across the additional instances to keep up with the millions of new customers. Requests will be first run through a Cloudfront CDN so that content can be delivered securely with low latency and high transfer speed. Stateless should hypothetically still be maintained, as the core features should not change when considering the application at a global level. Considering scaling metrics, increasing the CPU utilization from it's current 50% would better suit the increase in traffic, but some problems that were encountered during testing would need to be investigated, like when the Auto Scaling Group was scaled to its maximum and further requests were sent, the resources would get maxed out and cause Gateway problems

S3 will most likely still be used for image storage, as it keeps everything running within AWS. GCP Cloud Storage might be considered, as S3 has region issues where the resources available may be fewer in certain regions, which may cause issues when the application is used globally. GCP Cloud Storage has many regions available to store data to help mitigate this issue, but depending on usage, it might end up being far more expensive due to the support fee and expensive downloads. Elasticache could be swapped with Memcached, as it does boast more cores and better scaling. Elasticache is better suited to small datasets, so as the transformation options grow with a global application, along with the possible millions of users accessing the application, Memcached may be better suited to handle the load that would be created with such an upscale.

There is not much in place to prevent DDOS attacks when it comes to this application. There is AWS Shield, which is provided to all AWS users at no additional cost, which also has an advanced version that can be used to specify what AWS services to protect during such attacks. There is also Amazon Route 53, that makes sure any time a customer tried to access the application (e.g. typing picpro.com instead of www.picpro.com), they are routed to cloud distribution instead of accidental or malicious discovery of the application origin