CAB432 Cloud Computing
CloudProject Specification

Release Date: September 22 2021
Submission Date: Monday, November 1 2021 11:59PM
Monday of Week 14 of Semester
Weighting: 60% of Unit Assessment
Task: Pair-based Cloud Project with Individual Components
[Individual assignments permitted by application]

Note:

The due date for this assignment is early in week 14 of semester. Later that week or during the exam period, we require that you meet with us by arrangement to demonstrate your work. In particular, you must explain the scaling capabilities of your application, and answer questions as appropriate. Note that this time *there is a formal mark* attached to your demo as part of the report and demo section in the CRA. There will be clear instructions as to the format to be used – essentially a 'speed paper' with a max of 4 presentation slides + demo.

Introduction:

The focus of this assessment task is to explore a scalable, load-balanced cloud application with a stateless architecture. The application may rely directly on work you did for assignment 1, but you need to view them as separate problems. In particular, it is not compulsory to use Docker containers for this assignment, though you may work with them if you choose. The task requires that we generate some load and make the application scale automatically. The pracs – especially the assessable pracs - are essential background.

In this assignment we expect you to select an architecture that is suitable for your problem and that has an appropriate use of scaling, load balancing and persistence. Unless you are given specific permission because of the nature of your proposal, we expect your architecture to be *stateless*.

At the application level you are encouraged to consider your own proposal, one targeting something that you are genuinely interested in pursuing. I am particularly keen on students considering unusual data sources such as those arising from IOT devices or major science projects. But if you do not have your own pet application, there are a few basic scenarios on offer. These differ in the way the computational demand is generated.

The requirements are outlined in more detail below. We will expect you to use APIs to get the data, to process it, and to report on your work – for example the visualisation library d3.js (https://d3js.org/) will allow you to do a great deal in reporting the progress of your tasks and their results. This specification is not intended to be the final word on the data, analysis and display APIs. I have done a basic search to ensure some level of plausibility, and a number of these alternatives have been executed incredibly well in previous years. I expect that the class will share source materials without any particular fear, as there is plenty of work to go around if you wish to differentiate the assignments. It is also appropriate to point people to useful blogs or examples. It is not appropriate to share actual code – let them adapt the code from the blog themselves.

But first, let us consider some of the key aspects of the assignment.

Scaling:

The crucial aspect of the assignment is that you demonstrate the elastic scalability of your application. The application must scale robustly in response to the variations in demand inherent in your problem. Computational demand might vary based on:

- The number of concurrent "queries" being processed across a Twitter feed, based on additional requests (see the guide to using Postman) and multiple hashtags.
- The number of cameras being considered in a webcam application
- The resolution in a rendering application.
- The number of requests to a server as generated by a massive number of interested customers (or as simulated by a load testing application such as postman).
- Some other data feed or computational task as suggested by the student pair.

As you have now understood from the lectures and prac exercises, the game here is to generate instances sufficient to cover the expected load and to fire up others or terminate those which exist whenever it is necessary to maintain responsiveness.

There must be performance-related KPIs for your application, and once the demand is such that the current number of application server instances fails to meet these requirements, you must add further instances to overcome the violations and restore compliance. While CPU usage is the most common, students have also used network traffic, unprocessed items in an SQS queue, or the number of incoming messages on a messaging bus as appropriate markers for load and subsequently a need to create more resources.

As you write your code, we expect that you will approach scaling in stages, regardless of your application. Some of these may blur, but don't be in any great hurry:

1. Single instance - no scaling
2. Multiple instance – no scaling
3. Multiple instance – manual scaling
4. Multiple instance – automated scaling

Don't consider the scaling pool before you have properly understood the overall architecture of your application. Is your scaling pool going to be behind an internet facing load balancer? Or is scaling going to be internal to the application, behind an internal load load balancer which receives requests from some controller instance or service?

Work out your architecture and draw a proper Cloud application architecture diagram. Sketch with pen and paper and then discuss it. Throw away bad choices. Once you have something reasonable you may use a drawing tool like cloudcraft (https://cloudcraft.co/). At this point you will have some idea of how the scaling aspects of the approach will play out.

You have seen also how to create an instance and configure it for your needs, and then to save it as an image that can be used to create other instances. This is the approach you must take. Note that it will ***not be possible*** to obtain full marks if item 4 is not completed successfully. Each of the clouds available have policy configurations and APIs which support automated monitoring, creation and destruction of instances.

Persistence:

Persistence is an essential aspect of this assignment and the architecture marks depend heavily on your choice of storage services and whether these align sensibly with what you are trying to do. You will use at least two distinct storage services, and values must be recoverable should a VM instance die. We will not mandate that you use the managed service version of these storage types, but you will certainly need to have at least one of your persistence choices as a service away from the instance. Just running a MySQL installation on each of your VMs will not cut it given the way the CRA is written.

Usable examples might include:

- Redis cache (managed service) + Long Term Store (like S3)
- Redis cache on each instance + Long Term Store (S3 or similar)
- Cache + managed NoSQL DB service or SQL service
- Cache + designated VM hosting a SQL DB

And many others – the key is that the approach be stateless. In the case of a Redis cache on each instance, as long as we maintain regular syncing with the larger, slower longer term store we can be sure that the state is recoverable.

In the next section we will consider some potential scenarios and applications.

This Isn’t Assignment 1:

Don’t go counting the number of APIs or services. Your application will probably be complex and involve a range of services, but you don’t have to count them. You need a sensible architecture, persistence and sufficient load to generate scaling. The number of services can take care of itself.

Some Scenarios:

We will give some basic ideas on the scenarios that we are supporting. Note that the last of these is open-ended in any case, and you should feel free to propose anything that fits the guidelines, ***as long as you have it approved by us***. Note that the first task has been the default for some years now, and has now been used with considerable success – and occasionally with more limited success – since 2013. Occasionally we have concerns over the response time from Twitter in approving API keys. At the moment this is not an issue, but please get in touch with us as soon as you can if you experience difficulties.

Scenario 1: Twitter: The task usually involves tracking a selection of search terms and performing a range of analytics on top of the feed. This often includes additional term extraction (using an NLP library such as NLTK or natural (https://github.com/NaturalNode/natural) and sentiment analysis (see http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/ for a basic intro to this). In this case, the scaling comes from increasing message volume as you broaden the range of terms in the search. Note also that computational load can be added readily by hitting your server with a greater number of requests (using Postman or similar tools), increasing the comparisons undertaken, using semantic searches such as the interface to wordnet, and of course correlating events as described below.

Social platforms like Twitter attract a large numbers of users who use these platforms (among other things) to exchange and broadcast messages (text, links, photos, videos, etc.). As a result, these platforms produce a huge number of events, originating from their collective user base. Making use of this data beyond the core social networking purpose has been the subject of a number of studies and commercial projects. For instance, disaster management applications may use social network feeds to contextualise the situation. That is, citizens "report" relevant incidents (like flooded roads, fallen-over trees, power outages, fires, etc.) on social networks, which are then filtered and matched against other entities such as places (i.e., geographical location), incident categories (e.g., fire, flooding, etc.) and incident severity.

One cautionary tale here lies in dependence on location tagging. This has improved a bit, but remains relatively weak as not enough people geotag their tweets. You can, however, do location identification based on the text itself, with mentions of New York or London or wherever, but you should not depend on any sort of geotagged identities – at least not crucially.

The other caution lies in the selection of libraries. Natural remains probably the obvious choice for NLP in node (https://www.npmjs.com/package/natural) but it is important that you understand the limitations of each library and that you make sure that it does what you need it to do.

A slightly more formal description of the tasks follow below.

Scenario 1 is a simple Cloud-based query processor based on Twitter messages. Its core principle is to let its users enter "queries", specifying multiple "hash tags" (i.e., keywords that are manually emphasised by the authors of Twitter users). Once submitted to the application, a query may become a "live filter" applied against the inflow of Twitter messages. Any message which passes through the filter defined by the query (i.e., contains the stated hash tags and comes from the specified region) shall then be displayed on the screen. A query shall remain "active" (i.e., continue to filter incoming messages) until it is manually revoked by the end user of the application.

We here require that you use two or more APIs to 'do some work' on the results. Ideally we would like you to use NLTK or natural to extract fresh entities from the related tweets, and also to perform sentiment analysis on the tweets provided. Or something of equivalent complexity that will be interesting and justify a scalable cloud deployment. The links above and below some idea of how to do this sort of work. The application itself may be a simple page web site. You may base this on whatever technology you wish, but Twitter Bootstrap is a useful default. You will most likely use a JS visualisation library such as d3js to show the changes in sentiment, topic and tag clouds.

Some sources:

- The primary data source is the Twitter public REST API ***V 1.1*** which may be found here: https://dev.twitter.com/rest/public
- Note that Twitter are currently providing an early access release of their new V2 API. The link is here: https://developer.twitter.com/en/docs/twitter-api/early-access. I have not used it at all and have no advice to give at this stage, but you are welcome to explore it. If you do, please feel free to share your impressions on the #assignment2 slack channel.
- The NLTK may be found here: http://www.nltk.org/
- Natural may be found here: https://github.com/NaturalNode/natural
- NLP-Compromise is here: https://github.com/nlp-compromise/nlp_compromise
- Twitter bootstrap may be found here: http://getbootstrap.com/
- The d3js libraries may be found here: http://d3js.org/ and you may use the tutorials there or at https://www.dashingd3js.com

Note that if you see the wrapper for the Stanford core it is probably not you want here – that is for more sophisticated NLP research and processing, but of course if you are interested and can master it in time, please feel free to work with it.

Scenario 2: Webcams: There are a large number of public webcams available on the net. In this scenario, you are to capture the feed of a set of them, and use a computer vision library such as http://opencv.org/ to count the number of vehicles or faces in a frame. Now this will take some tweaking to be plausible, but it is evidently not too tough in principle and over twenty groups have done this one over the past few years. There is a reasonable discussion here: http://answers.opencv.org/question/2702/simply-count-number-of-faces/, and we would expect you to have a map showing the density at places monitored by the webcams. This will provide some decent processing, and we may scale up based on increasing the number of data sources.

Our requirements are really little more specific than stated above. This is all about the scaling and your own preferences. But please be aware that libraries such as openCV do have their own overheads and you do not want to get hung up on those at the expense of the mark-laden cloud-related work of the assignment. There is also some variation in the quality of the language bindings for some of these libraries and services. That community is ***strongly*** C++ dominated, and other libraries and adaptations seldom have the same level of maturity and active maintenance associated with the core work. That said, I have had capstone students working with this in Python very successfully, and there are at least two node projects (see https://www.npmjs.com/package/opencv for example) with bindings to recent versions of opencv. Either way, have a decent look at things first before committing, and make sure that your programming expertise matches the demands of the libraries.

There are quite a number of open (or at least fairly open) web cam APIs, though you may need to use a lot of them. Note that many of these do not really provide video, but rather stills updated every so often. This is getting better, but spend some time looking at the data sources before you commit. If you aren't happy with the data, pick another alternative.

As a related topic, you could instead look at transcoding or video analysis using similar libraries. Here we would scale on the number of entries – i.e. videos to be analysed – in the queue. I have also had people do some very successful projects using machine learning libraries that transform images in style or colour. So there are many variations on this theme that you can consider. Pick some that interest you.

Resources (web cams):

- QLD Traffic APIs: https://data.qld.gov.au/dataset/131940-traffic-and-travel-information-geojson-api
- Also search on "public webcam feeds" but please use only those likely to comply with the QUT ITS usage rules… [Yes, I mean it.]
- The OpenCV project: http://opencv.org/ - this offers bindings for various languages.
- As before, use d3js or similar to display the results – there is a map library available.

Others:

- There are a lot of transcoding libraries available. For the applications that people create, have a good look at Soeren Balko's company Clipchamp: https://clipchamp.com/en/
- For neural style transfer, please see this blog and the links it draws on: https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398

Scenario 3: Rendering.
This is based on rendering a graphical image at a variety of different scales. Most, we expect, would rely on an open engine such as Blender. However, we have usually relied on people with experience in this domain to make their own proposal.

Scenario 4: Open Proposals:
Scenario 4 is entirely open ended - you propose something, and we agree or we do not. We also have some public research data that we are happy for people to consider. These extra alternatives may include:

- Bioinformatics sequence data and related images or metadata for comparison.
- Images and training of machine learning models
- Open data sets available as part of Gov Hack and other initiatives (see https://hackerspace.govhack.org/data_sets)

However, be aware that some of these datasets are often static, and your task may be complicated by the need to work out how to scale things. More queries? More analysis? You will need to think this through carefully, propose it and have it reality checked. You also need to think very carefully about the architecture: where is the computation pool? Where is the load balancer? What triggers scaling? Do I use a queuing system?

Whatever your ideas, please give us a rough outline as soon as possible, and certainly by sometime early in Week 11. We can refine this as needed.

We now consider the overall structure of the assignment.

Some Basics:

The Key Learning Objectives of this assignment lie in this elastic scalability of the process and the tasks are set up to support this objective. While we are interested in the creativity of the applications, and encourage this, don't get carried away. Look at where the marks lie.

The following guidelines apply:

- The application itself will usually appear as a simple single page web site. You may base this on whatever technology you wish, but Twitter Bootstrap (http://getbootstrap.com/) is a useful default. The site must be flexible enough and architected sensibly enough to support variations in scale and processed data.
- You may use alternative client side frameworks but this is not the time to learn a new one. [Same comments as for assignment 1.]
- The application will be implemented on top of a Cloud IaaS service (e.g., AWS EC2, Azure, GCP) where you manually install the application server (node.js). You are free to use other services provided by the infrastructure (e.g., the load balancer, storage services, management services, etc.), but we don't allow PaaS offerings as these do too much of the work for you.
- Your application *must* be deployed to a scalable public cloud infrastructure, but you may choose which one to use. Most people will just use the QUT managed AWS or Azure systems. If you have particular requests amongst the AWS or Azure services that are not currently accessible, talk to us early. These will be assessed on the basis of their suitability and cost, and we will try to allow them if we can. But there are budgetary limits.
- There must be no cost to me or to QUT for use of any external service other than A– so if you wish to use an API that attracts a fee, you may do so, but it is totally at your expense.
- Given the focus on scalability, you must choose the smallest available instances – micro instances in Amazon terminology, with similar configurations for Azure – and launch as many of these as are needed to make the application cope with the demand. DO NOT under any circumstances provision a massive scale individual server instance. This is expensive and completely defeats the purpose of the exercise.
- Persistence is a ***required aspect of the assignment*** as discussed above – you must use at least two distinct services. But it is not necessary to store the entire history of a dynamic application such as that based on Twitter. I expect that you will have sufficient storage to maintain a window of recent and/or relevant events. The

storage chosen depends on the latency acceptable and this is something you must justify.

- ***YOU*** are responsible for switching off all instances at the end of each session so as not to attract usage charges.
- On the client side, we expect that you will again use javascript, but you may choose others as you wish. You should base your work on a standard web page layout. You may find Twitter Bootstrap (http://getbootstrap.com/) a good starting point, or you may roll your own based on earlier sites that you have done, or through straightforward borrowing from free css sites available on the web. Your work will be marked down if it doesn't look professional, but won't attract fantastic extra marks for beauty. Simple and clean is fine. Cluttered pages with blinking text reminiscent of the 1990s are not.

Please get in touch if you need to clarify any of this.

The Process:

The cloud project is designed as a paired assignment to keep the workload more manageable: there are no learning outcomes related to teamwork and you don't need to document your process closely.

I expect that you will be forming pairs early in the first week after the holidays if you have not done so already. I will assist with this process as needed through the dating agency channel on Slack. Some people will inevitably wish to undertake the assignment individually. This will be considered on a case by case basis, but:

- No special consideration will be given to people who do the assignment individually.
- The specification is intended for two people, and this general scope will be the benchmark.
- You must ask me about this ASAP and certainly early in week 11 at the latest.

But as this assignment is now worth a full 60% of the unit, there are some additional requirements at a individual level – we can't have pure group work above 50% of the unit weighting.

So, we will split things as follows:

- Most of the assignment will remain as a group task. The development and the presentation are all quite reasonably viewed a joint effort. We will also require that you present a joint report, and in this report you will document the basics – what your app does, how it is put together and how you tested it and made it scale.
- But this time we will require an individual report in which we are going to get you to analyse your project as a Cloud Application. Does it do scaling well? Does it do persistence well? Is it stateless? And then we are going to get to reimagine your application on a global scale, and tell us what would need to change. This will be intentionally challenging. The task is so that you can show us how well you really understand the principles of a modern cloud architecture. To make this less threatening, it is not necessary that your application show all of the principles we talk about. What we want is for you to show us some insight into how these things

relate to the work you have done. The details for this are in a separate report template, and you should respond to the prompts I have given you.

There is a lot of flexibility in the specification, and we will again ensure that you are producing something that really is worth 50% of an advanced 12 credit point unit. So there will be checkpoints to make sure that you are on track, at which time you will be given clear and unambiguous feedback on whether your proposal is up to scratch. The pre-submission requirements below should be seen as drafts of the final report to be submitted as part of the assessment. Please bring a hardcopy to the pracs for quick feedback. This will help take the load off the emailed versions which become very hard to keep track of.

In respect of the application itself, the process must be broken down into defined stages. While many are possible, we would see the main steps as follows:

1. Accessing and learning to use the relevant APIs
2. Simple deployment of a basic application to a single instance, supporting basic operations and reporting.
3. Developing and reality checking your cloud architecture, including your choices relating to persistence and ensuring statelessness.
4. Adding other API work to the application – especially visualisation.
5. Exploring scaling behaviour manually – including deployment of additional instances.
6. Automating the scaling behaviour and including a scaling pool.
7. Verifying your application performance.

There is some variation possible in this, but do ***NOT*** attempt to undertake these at once. There are clearly defined partitions between single instance deployment and then manual and ultimately automated scaling.

There are also some checkpoints that need to be considered:

[Early in Week 11] : Proposal: A one pager with the following information should be sent to your tutor and copied to cab432@qut.edu.au :

- Overall application purpose and description (1-2 paragraphs)
- Architecture diagram and proposed phases of implementation – including specific Cloud service APIs and facilities to be exploited.
- Discussion of persistence choices and scaling metrics and application thresholds
- Set of example use cases
- List of service and data APIs to be utilised. This must include a short description of the API (up to 1 paragraph), and a list of the services to be used for each user story (see above).
- [optional] a mock-up of your application page.

As with Assignment 1, please discuss this with your tutor – the email is more for our records. The tutor will read it and follow up with you if there are any issues. Please note that this checkpoint is mandatory – no exceptions.

[Early in Week 13] : Report Draft: A week before submission, I expect you to produce an updated version of the proposal which fleshes out explicitly your application development in accordance with the report template we have provided. This is not compulsory, but provides you with an opportunity for feedback.

[Monday of Week 14] : Final Submission: This is the due date for the project. I will expect a completed report, code and tests for the group task, and individual reports from each team member. As discussed at the start of this document, we will expect each team or individual to demonstrate the app over the week or two after submission. Precise requirements will be made available later. As with assignment one, it will ***not*** be necessary for you to maintain the app in the cloud. You may deploy again just prior to the demo.

Submission:

The project code is to be submitted via Blackboard on or before the due date. I will provide you with a submission link and submission guide closer to time, but the general structure will be very similar to that suggested for Assignment 1.

The submission must also include a group report which meets the requirements discussed in the report template, and *independently written* individual reports meeting the requirements discussed in the individual report template.

Marking:

Marking of this assignment will require that you present for a live demonstration of your system by appointment after the due date. In contrast to the mashup assignment, the assessment rubric includes a small component covering the professionalism of the demo, but the major focus will be functional. Given the difficulty in getting the whole class together at this time, these appointments will be individual and not public, though we will normally schedule several groups at once, and we expect you to act as a professional audience for your colleagues. We will provide you with a template for the slide deck.

More details in due course.