CAB432 Cloud Computing
CloudProject Specification

Release Date: September 23 2022
Submission Date: Friday, November 4 2022 11:59PM
Friday of Week 14 of Semester
Weighting: 60% of Unit Assessment
Task: Pair-based Cloud Project with Individual Components
[Individual assignments permitted by application only]

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 they are very separate problems. In particular, it is not compulsory to use Docker containers for this assignment, though you may do so if you choose. The task requires that you 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* and we will mark you down heavily if it is not.

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, as we explain later.

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 in a social media application, based on additional requests (see the guide to using Postman).
- 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 idea 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. For the best marks, we will expect your application to anticipate the load. While CPU usage is the most common metric, 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.

To avoid trivial applications and scaling behaviour, we will enforce a minimum standard for each architecture and scaling metric. As most people will use an application load balancer in front of a target group based on EC2 instances, we will make the following requirements the default for scaling in this assignment:

- Use of an AutoScaling Group based on Ubuntu 18.04 or later and t2.micro instances
- Target Tracking Policy based on Average CPU Utilisation with a target of 50%
- Scaling out from a single instance to a maximum of 5 instances
- Scaling in from the maximum to return to a single instance.

These standards are written with an internet facing load balancer in mind, but they also apply to internal load balancers and scaling groups. If you are using Azure or GCP, or you are using a different architecture – one based on a queuing system, for example - or some other

metric, then you must discuss this with us and we will negotiate an appropriate target with you. The point is not to make this especially hard, but to keep the task as realistic as possible.

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 a good mark for this section if automated scaling is not achieved 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. But first, a reminder that this is Assignment 2.

This Isn't Assignment 1:

Don't go counting the number of APIs or services. Also note that this doesn't need to be a mashup and you don't need to count use cases. 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 rest can take care of itself.

Some Scenarios:

We will give some 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 successfully since 2013. Occasionally we have concerns over the response time from Twitter in approving API keys. Please get in touch with us as soon as possible if this is an issue.

These descriptions will get pretty detailed, so I have pushed them all to the end of the document, where each of them gets their own section. A brief summary is provided below. At some point you will need to read the longer version, but for now you can just read these quick overviews and get a feel for what might be possible.

The scenarios are:

- **Scenario 1: Twitter:** use Twitter streams as a data source. Load comes from the volume of tweets, doing natural language processing and other analysis and presenting the results, and from catering for more users.
- **Scenario 2: Other social or news sources:** Here you will follow much the same plan as for Twitter, but we do not have strong suggestions here as the level of traffic needs to be consistently high. If you have some alternatives here, please discuss them in detail with your tutor to get an idea of the load.
- **Scenario 3: Image Transformation or Video Transcoding**: Here the idea is that users will upload images or videos at a particular resolution and the app produces the image or video in a range of formats and resolutions as specified by the user. Load comes from the task – which is very intensive – and catering for more users. This scenario could also be adapted to the case of image rendering using ray tracing. Here there is no original image, but much of the rest of the description is the same.
- **Scenario 4: Webcams or other video analysis:** Here the data comes from video streams emerging from webcams across the internet. The app uses computer vision to identify objects such as cars or human faces within these streams, returning counts and other analytics to the user. Load comes from the number of feeds and the complexity of the analysis.
- **Scenario 5: Open Proposals:** Here the student team comes up with an application proposal that is based on something they are interested in. The idea may be adapted from one of the scenarios above, but there is a lot of freedom in choosing the

domain. We will review any student proposal and help you refine it or find a suitable alternative. You cannot proceed with student proposal without our approval.

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 or SaaS 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 external Azure systems. If you have particular requests amongst the AWS 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 the AWS accounts. 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 – t2 or t3 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. If the instance type is running out of memory, talk to your tutor about the selection, and choose the next step up in specification from these families.
- Persistence is a ***required aspect of the assignment*** as discussed above – you must use at least two distinct services.
- 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, the expectations are the same as for assigmment 1: 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 team 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.

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 we 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 should 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, but they will not reply if the proposal is consistent with an earlier version they have approved verbally. Please note that this checkpoint is mandatory – no exceptions.

[Week 12 or Week 13] : Report Draft: Up to 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.

[Friday 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.

Scenarios in Detail

Scenario 1: Twitter

The Twitter-based task involves processing tweets, performing a range of analytics – usually natural language processing to extract terms or topics and perform sentiment analysis - and presenting the results using some graphing library. The original idea was based on the Twitter firehose, an API end point that produced masses of data. The API is now more restricted and some approaches that we have allowed in the past – generating load using a collection of search terms over multiple connections to the API - are unlikely to satisfy the target tracking requirements listed above. We thus recommend that students working on this scenario use the *volume streams* or *filtered streams* endpoints:

- https://developer.twitter.com/en/docs/twitter-api/tweets/volume-streams/introduction
- https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction

We will provide more details of these below. Many applications can be based on a stream of data from Twitter or other social media providers. Such platforms attract a large number of users who exchange and broadcast messages including text, links, photos, videos and other formats. 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. Others have looked at social media datasets to track conflict or political movements. While I would urge some caution in following some of these things too far down the rabbit hole, it is perfectly acceptable to try to write an application that uses social media analytics and follows the same strategy as others as long as your cloud application is written by you and satisfies the technical requirements of the assignment.

Some more detail of the scenario and the data source:

The main idea is that the Twitter platform provides a stream of tweets that is analysed by the application and potentially filtered by the user. Earlier students have used the standard API search endpoints that came with version 1.1 of the Twitter API. These have now been deprecated and the new API (V2) is available here:

https://developer.twitter.com/en/docs/twitter-api

All programmatic interaction with the Twitter API requires that you sign up for a developer account. You will need to look at this here:

https://developer.twitter.com/en/docs/twitter-api/getting-started/getting-access-to-the-twitter-api

A range of endpoints are available, but our view is that you should focus on the two alternatives we mentioned above, volume streams and filtered streams.

Volume Streams:
https://developer.twitter.com/en/docs/twitter-api/tweets/volume-streams/introduction

This endpoint provides a small random sample of publicly available Tweets in real time – approximately one percent of those available. In testing, we saw something 2500 tweets in a minute, and the great advantage of this endpoint is that there is no limit to the number of tweets that can be retrieved.

Some issues to consider – and some partial solutions:

- The endpoint does not provide a filter parameter and so queries for hashtags or words would need to be handled in your application.
- Tweets are most commonly in English, but this is a sample of all of the whole platform.
- Only one connection is allowed at a time, so this will affect your architecture significantly. Think about this first, before doing anything else.

Filtered Streams:
https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction

As the name suggests, it is possible to filter the stream using this endpoint to produce a more restricted set of tweets. In particular, you can set it to provide tweets in English only. But there is a hard limit of 500K tweets a month and some of the interactions with the endpoint are much more complex – see the link for details.

Cloud Architecture:
The key to all of this is ensuring that your architecture is chosen to work well with the data source and the application. This is true for all of the scenarios, whether they are ones we have suggested or applications you have designed yourselves. In the Twitter case you need to think carefully about the way the data is passed into the application and how it is handled when it gets there. Do you store all of the tweets and then process them? Or do you try to process the data in real time? Can you filter the data as it appears or do you just have to discard some of the feed?

The lesson for all applications is that some of this means that you have to explore the data source – get an account, make some requests to the API and see what it all looks like. You can't just write an app two days before the assignment is due and hook it up to the feed and expect it all to work.

Sketch out your architecture. Throw it away and try again until you can answer some key questions:

- Where is the load coming from?
- How am I scaling – is there an internet facing load balancer or is it internal?
- Do my persistence choices make sense?

Remember, this applies regardless of whether you use Twitter or not – you have to understand how the application passes data around and how that affects your approach.

Some Suggestions:
We have emphasised the need to analyse the tweets and present the results. There is here an emphasis on natural language processing and visualisation of the data. The application itself may be a simple one 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.

Term extraction (named entity recognition and so on) will normally use an NLP library like natural (https://github.com/NaturalNode/natural) though there are other choices like Compromise, a simpler NLP library. Sentiment analysis is also commonly used, but you may need to build this from other components. The blog here http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/ gives a good introduction, but using a Python library. to this). Computational load can be added by hitting your server with a greater number of requests (using Postman or similar tools) or by increasing the comparisons undertaken or adding more complex semantic searches.

Some useful resources include the following:

- The NLTK may be found here: http://www.nltk.org/
- Natural may be found here: https://github.com/NaturalNode/natural
- Compromise is here: https://github.com/spencermountain/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

Some Cautions:
Don't rely a lot on location tagging. This has improved, but there are still limits because 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 geotagged identities may remain problematic.

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. You may start with Compromise as a very basic choice, but note that natural is much more mature.

And 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: Other Social or News Sources

Mostly this is just repeating the advice in respect of the Twitter scenario, but generalising to other sources. We suggest that you pay close attention to the Twitter discussion above, and follow much the same plan as for Twitter, paying very close attention to the data source and to the architecture you are using to handle it. The level of traffic needs to be consistently high, so please discuss your ideas with your tutor in more detail to confirm that the source and the resulting load is suitable.

Scenario 3: Image Transformation or Video Transcoding

As discussed above, the idea is that users upload images or videos at a particular resolution to some cloud storage and the app processes them in turn into a range of formats and resolutions as specified by the user during the upload.

There are a couple of good examples of vendors who provide these services if you want to get a clearer idea of what is involved:

- https://cloudinary.com/
- https://clipchamp.com/en/

However, you don't need to reproduce the range of services offered, and I would caution that the architecture used by the commercial vendors might be wildly different from what is needed here. Clipchamp, a company founded by our former colleague Soeren Balko, has an edge computing based technology, in which video editing and transcoding are done in the browser. You can't use edge computing here, but you can look at Clipchamp for inspiration for the task, and for future possibilities for your own start-ups: https://clipchamp.com/en/blog/microsoft-acquires-clipchamp-to-empower-creators/ .

The basic idea might be something like a just in time transformation for an image, where a user loads a source image at 1920x1080 resolution at 300 DPI with a request for 300x300 at 75 DPI for use as a thumbnail. The task can be simplified somewhat by restricting the range of alternatives and by using the Node Sharp library to deal with processing:

- https://www.npmjs.com/package/sharp

Note the potential for use of multiple types of persistence and alternative methods of managing the application. An additional complication may be to use a library to perform object recognition.

A final alternative here is to use an application like Blender to create graphic images at various scales and resolutions using ray tracing. The approach is very similar to the ideas described above, excepting that the application does not rely on an uploaded image, but rather creates it from scratch.

Scenario 4: Webcams and Other Video Analysis

There are a large number of public webcams available on the internet. Here you access a set of such feeds and use a computer vision library such as http://opencv.org/ to count the number of vehicles or faces in a frame. We have had well over twenty groups implement this application successfully, and many of the ideas are explained in this forum answer: http://answers.opencv.org/question/2702/simply-count-number-of-faces/. None of this is perfect and you are not marked on the overall success of the computer vision analysis. But people commonly process the video and show a map indicating the feed and the count in each location. Scaling is then controlled through the number of data sources.

Some Ideas and Some cautions:

Our requirements largely as stated above. But please be aware that libraries such as openCV have their own overheads and you do not want to spend too much time 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.

There are now at least two node projects (see https://www.npmjs.com/package/opencv for example) with bindings to recent versions of opencv, but please:

- Have a good look at the maturity of the node projects
- 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.

You are of course free to use other video data sources but be aware of the need to find a reliable source of video streaming data. Please also see the transformation and transcoding topic.

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:

- More sophisticated object recognition may be obtained through the use of tensorflow and other libraries. Please see the API github here for details: https://github.com/tensorflow/models/tree/master/research/object_detection. I have not searched for language bindings – you will have to do this yourselves.
- For neural network based style transfer – transforming images to look as though they have been created by a particular artist - please see this blog and the links it

draws on. This work can be used here or perhaps be linked to the Transformation and Transcoding scenario: https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398

Scenario 5: Open Proposals

Scenario 5 is entirely open ended - you propose something, and either we agree or we do not. We will help you refine the idea to make it more suitable, but sometimes you will just have to change to a different topic. Please note that you cannot undertake an open proposal without our approval.

Some suggestions:

We have a range of public research data that we are happy for people to consider, and we can also connect you with colleagues elsewhere in the university. These extra alternatives may include:

- Bioinformatics sequence data and related images or metadata for comparison.
- Images and training of machine learning models
- Large scale social media data
- Public and research oriented sensor data
- 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 for discussion as soon as possible, and certainly by sometime early in Week 11.