CAB432 Cloud Computing
Mashup/Docker Project Specification

Release Date: August 12 2022
Submission Date: Monday, September 19 2022
[Monday of Week 9]
Weighting: 30% of Unit Assessment
Task: Individual Project

Note: *The Unit Outline lists this assessment as due in week 8. The choice of Monday of Week 9 is intended to give you that final weekend to pull it all together if you need it. The submission link will be made available early in Week 8. Marking will commence from the middle of Week 9, and we will be talking to you about the schedule well before the assignment is due.*

Aims:

The aims of this assignment are to:

- Introduce the use of cloud based VM resources for application hosting
- Practise configurable deployment of cloud applications using the Docker container
- Provide experience in the development of lightweight web applications which draw on cloud data and services

Introduction:

The first assessment for the Cloud Computing unit requires that you build a web application – a server-side mashup – and then use a Docker container to deploy this application to a public cloud instance. We will also require that your application interact with a cloud persistence service – details are provided below – to maintain a count of the number of visitors to your site. The task may be broken down as follows:

- ***Create a simple web application***
  - A mashup of services and data (see later)
  - Most of the work must be done on the server side
  - Technically an extension of the prac exercises from weeks 4 and 5
  - The server should be node.js-based

- ***Write a site visit counter and display this as part of your application***
  - This really is just a visit counter – a single number
  - Please keep this simple and clean too – no Geocities flashing please
  - But you must rely on a Cloud persistence service
  - More details of the services available below.

- ***'Dockerise' the app***
    - Create a Docker container that hosts your app
    - Generally using a Dockerfile and `docker build`.
    - Please talk to us about `docker compose.`

- ***Deploy to a public cloud***
    - *Must* deploy to a public cloud VM running Ubuntu 18.04 or later
    - *Azure, AWS or GCP all ok*
    - You ***must*** deploy via a Docker container to get a good mark

- ***Write a report***
    - Tell us about the mashup and the services used
    - Tell us how you built it
    - Tell us how to use it
    - Analyse your application based on some questions we ask

- ***Show us that it works in a demo***
    - Most of the marking will be face to face (subject to confirmation)
    - The rest will be based on your code and report

We will describe the task in more detail below.

Mashups:

The notion of a mashup is introduced briefly in the Assignment 1 overview, and it is essentially a combination of individual services to provide a more sophisticated outcome. Those seeking greater inspiration should take a close look at the wonderful Programmable Web site, the source of all things API on the web:

http://www.programmableweb.com/

In principle, there are few limits on the range of services or applications you may use, and you should certainly take a look at some of these examples. We will provide a list of example services that you might use, and some possible use cases. You are free to use those, but you can also come up with your own. Most people choose something that interests them.

You should check out your ideas by manually prototyping the workflow. And you should also apply for any API keys that you need ***immediately***. These can take (quite some) time. Please tell us about any delays.

We will talk about the mashup in more detail below, but these are the basic guidelines:

- *You cannot use existing one-click services. This clause is in place to rule out mashup frameworks, and we will leave it in just in case any still survive somewhere. Applications must be coded explicitly, and we will only support node-based servers.*
- Your application must be non-trivial and involve at least 2 services (details below)
- Your application must implement at least 2 distinct use cases (details below)
- Each use case ***must involve composition of the services.*** It is not sufficient to have a simple web page which displays output from two services with the same search query.

See the example in the overview presentation to clarify this. There must be some added value in the results which come back that would otherwise not have been possible.

- Your application must involve a significant server side component (details below). Some limited mashing on the client side is ok for presentation purposes, but the bulk of the work must happen on the server, and you will find this far more convenient anyway. Trust us on this – it is much easier. If in doubt, ask us.

Your work will be assessed according to its technical merit and usability, and on your ability to document and analyse the application. Naturally, these criteria are made more formal in the assessment rubric. There is no explicit assessment weighting given to the 'quality of the idea' as this would be very difficult to isolate, and we are providing you with some examples in any case.

The Page Counter:

The page counter is included here to allow you to work with at least one cloud persistence service. People come to this unit with a variety of software development backgrounds and some approaches are more familiar than others. The options below are included so as to make it possible for everyone to satisfy the requirements and get the marks. As you will learn soon, the big divide in cloud persistence is between entity or object stores – think `<key,value>` pairs like a Python dictionary – and relational storage based on SQL databases, whether standalone or part of a managed service.

For this task, you may use a second Docker container (see below) or one of the following managed services in the QUT AWS Environment:

- S3 - https://aws.amazon.com/s3/
- DynamoDB - https://aws.amazon.com/dynamodb/
- RDS – https://aws.amazon.com/rds/

The first two of these are entity or object stores. S3 is the Simple Storage Service, one of the original AWS services, providing scalable and highly durable object storage. DynamoDB is a fast and flexible nonrelational database service for any scale. RDS, the Relational Database Service is a hosted SQL offering. We will teach you how to use S3 programmatically, and after that we will expect you to manage from the guides for any of these other services you don't understand yet. Interactions with DynamoDB are similar in principle but DynamoDB is set up to have more of a traditional database feel.

We expect the counter to track all the page visits since your application was first deployed. You will have a bucket unique to you, and a count object you can call `counter` or similar. The ideas seem trivial, but it is important to understand the process: you are not incrementing a variable but replacing an object with an updated version. The approach is as follows:

Initialise:

1. Check that the bucket exists. If not, create it.
2. Check that the counter file exists. If not, create a counter object with value zero and upload it to the bucket.

Visits: For any GET request to the site:

1. Access the counter object and parse it to get an integer count
2. Increment the integer count and create a new string object with the revised count
3. Display the current count as part of your application

The alternative to an object or entity store is to use SQL. Here we have two options:

- RDS with a low specification instance as discussed above
- A second Docker container hosting a local instance of MySQL or some other relational DB that you are familiar with. For example, the official MySQL Docker image may be found here: https://hub.docker.com/_/mysql

Here we will expect that if you choose to use a SQL option that you will already have some idea what you are doing with SQL. If you don't know SQL then use S3 or DynamoDB. Your task will again involve updating the record that exists in the DB and you will need to make sure that the table structure is set up appropriately.

In all of these cases the work - and the point of the exercise - is in the interaction with the services, using credentials of some kind and managing the transfer of information across the network.

Some comments:

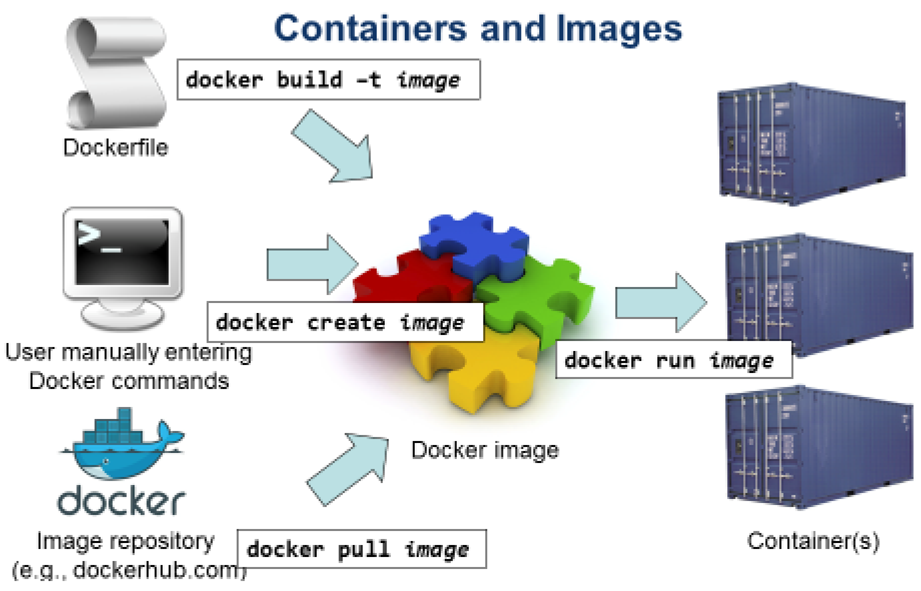
- The QUT environment already allows access to S3. The other two – DynamoDB and RDS – will be made available from week 4. There will be restrictions on the instance specification and the images available.
- You are permitted to use similar services offered by Azure and GCP.
- S3 is probably the easiest option, but please pick the alternative that works for you.
- In respect to the Docker option, yes, we completely understand that this approach is not stateless and that we are making a compromise here. That's ok for now.
- Finally, Elasticache (https://aws.amazon.com/elasticache/) might be a really good option here, but it is very expensive.

Please note that we will give no credit for page visit counters which do not rely on cloud services or a Docker container. If you have a cloud alternative that you are unsure of, please ask a tutor or email cab432@qut.edu.au.

The Target Environment: Docker and the VM.

You are required to deploy the app via a Docker container on top of a publicly available Linux VM. Ubuntu has been chosen because of its widespread adoption, and we will allow Ubuntu 18.04 (as used in the lectures and pracs) or later (like 20.04 or 22.04). You may develop the image on a local Linux machine, WSL or a local VM under Virtual Box, or on a cloud instance. But the final deployment must be on a major public cloud: AWS, Azure or GCP. Please talk to your tutor if you are using a Mac – we need to make sure early that your image will run on a standard VM as this has been a problem.

You have had a full lecture on Docker and Pracs in week 2 and 3 based on the use of Docker and AWS. You should all now have an AWS account and some experience of deployment, and you will find the other public clouds equally painless in provisioning VMs.



More specifically, we require that you set up (i.e., install and configure) your software stack in a Docker container. You should follow a similar strategy to the Docker prac, creating a Dockerfile which will allow you to create an image and to then deploy the container. Please pay *very* close attention to the port mapping. ***Hint: sort this out early with a very simple server and then use the same ports and port mappings forever afterwards.***

Once the image is created, you should push your Docker image to a ***private*** DockerHub, GitHub or BitBucket repository. Deployment will then be a simple matter of pulling the image from the repo and running it on the target VM. You will need to demonstrate this deployment to us live as part of the marking process, whether face to face or via Zoom or Skype.

The public cloud requirements for this assignment are limited to the use of a Linux VM and some simple persistence as discussed above. There is no need to use multiple instances, a load balancer or any autoscaling strategy.

Assignment 2, needless to say, is a very different story.

The Application:

In this section, we will give some guidance on your application. The basic guidelines were summarised above for ease of reference. Here we will go into a bit more detail.

Service or Data APIs:

- Your mashup must involve at least two (2) separate, non-trivial, service and/or data APIs. The combination must make some coherent sense and the APIs must be distinct. In the examples we give, we have organised the services into two columns, and it makes much more sense to choose one from each column.

- If you are choosing your own services, talk to the tutors or to me as a reality check - especially to check that your service choices are 'non-trivial' enough (see below). If you are struggling for inspiration, take your interests into account, and go searching at Programmable Web.

The Use Cases:

- The mashup service you provide must support at least two (2) non-trivial use cases that make sense for the domain. To make this more concrete, if my mashup focus was to provide restaurant information for tourists (using a search API hitting a site that provides tourist information, coupled with a translation service and a mapping service) then a sensible use case (expressed here as a user story) might be: *As a tourist in a foreign country I wish to find a restaurant near me and be able to read their menu*.
- Broadly speaking, we will see the use cases as separate if there is a substantial shift in the underlying service accesses. So a use case in which I am seeking *Thai* food is to me NOT different from a use case in which I am seeking Italian food. But seeking Italian food from a restaurant with some minimum government hygiene rating is a different use case from just searching for Thai food. [No, I have no idea if there is a suitable hygiene API – my point is to emphasise that it would require a separate API and more parsing and processing.]

Some Constraints:

- There must be no cost to me or to QUT for use of your service – so if you wish to use an API that attracts a fee, you may do so, but it is totally at your expense. It should not be necessary in any case.
- Request your service accounts and API keys ***as early as possible***. These may be instantaneous or they may take quite a while. Talk to us early if delays become a problem. Don't just wait until the day before submission hoping that the key will appear.

The Server:

- The actual mashing of the services – the parsing and combination – ***must*** take place largely on the server side. The architecture should be explicit in your proposal and in your final report. As noted, the software stack will be organised through a Docker container, and you will be required to deploy in our presence and to make the app accessible from a public VM. More details on this closer to the submission time.
- You should use node.js and base the application on work done in the pracs. I will consider requests to use servers other than node on a case by case basis but there will have to be a good technical reason. Node is mandatory for assignment 2 and it is time to get used to it. The other advantage is that if you complete the server exercise in Prac 4 then you are already most of the way there. This of course is deliberate.

The Client:

- There is little alternative on the client side to Javascript, but it is *not* necessary to use a modern UI framework like Vue or React or Angular.
- You may 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 site should look clean and be logically organised. You will be marked down if the site is poorly designed and clumsy, but we do not expect professional graphic design. Simple is fine. Cluttered pages with blinking text reminiscent of the 1990s are not.

- If you are unsure that your front end is acceptable, do a mock up and show a tutor or me.
- If you don't know much HTML, talk to us and we will help you get started with resources from W3Schools and CAB230. This won't take more than a couple of hours at most.

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

The Report:

A significant fraction of the marks for this assignment is based on the report that you submit. This falls into two distinct parts – one a standard software development report, and the other analysing your application in response to the questions and prompts that we provide. The sections we require are described in detail in the report template and the standards are laid out in the report CRA. Some aspects of the report appear in the one page mandatory proposal discussed below, and you should feel free to use the report template for this simple one pager to save time later.

The Process:

The mashup project is an individual assignment and there is enormous flexibility in the specification. However it is still worth 30% of your assessment, so we need to be careful in ensuring that it is of the appropriate standard. So there will be a checkpoint to make sure that you are on track, at which time you will be given clear and unambiguous feedback on whether your proposal is acceptable. This checkpoint will involve discussion with your tutor, and won't attract any marks, but please take the opportunity to ensure your submission will be a good one.

The checkpoint requirements below should be seen as drafts of the final report to be submitted as part of the assessment.

[Week 5]: Mandatory Proposal: During *week 5* – and certainly by sometime in week 6 - I expect you to have produced a one pager as discussed below. Most of this is required for the final report and you may use the report template, leaving the other sections blank. The document should be submitted as a pdf via email to your tutor and cc'ed to the class enquiries email cab432@qut.edu.au. The information required is as follows:

- Overall mashup purpose and description (1-2 paragraphs)
- At least two non-trivial use cases to be implemented. Please use the user story style: ***As a <USER-ROLE> I want the system to <DO-SOMETHING> so that <GOOD-THING-CAN-HAPPEN>***. For each one of these, you should make it clear (below) how the APIs can support the user story.
- List of service and data APIs to be utilised. This must include a short description of the API (1-2 sentences is fine), and a list of the services to be used for each user story (see above). You can add an entry here to describe the persistence service used for the page visit counter, but feel free to change your mind later.

- A clear statement of the division between server side and client side processing, and the technologies to be deployed.
- [optional] a mock-up of your application page.
- The email should be sent with subject line [CAB432 Assignment 1 Proposal] to your tutor and cc'ed to cab432@qut.edu.au. Your tutor will look at your proposal with you in the pracs and give you quick feedback. If they are unsure of your proposal, they will take a look at the document and follow up via email or at the next prac. In general, however, the tutor will not reply to this email.
- The copy to cab432@qut.edu.au is so that we have a record of every proposal and we can look at the overall standards.

[Week of Monday, September 12]: Report Draft: Around a week before submission, you should update your proposal and flesh out explicitly the details of your application and the analysis we require. The report should follow the guidelines and questions in the report template. You do not need to send this to us, but you should use the last prac before submission as a chance to make sure that things are on track. It will also ensure that you ***actually have a report to submit…***

[Monday, September 19] : Final Submission: This is the due date for the project. I will expect a completed report, code and tests, along with deployment and execution instructions. Precise requirements are listed below.

Submission:

The project report, source code and a copy of your Dockerfile are to be submitted via Blackboard on or before the due date. The report should be submitted as a pdf derived from the report template. I will provide you with a submission link and detailed instructions closer to that time, including a guide to creating an assignment submission zip.

I don't have strong views about this, but I would like to see some clear organisation of the resources, and I would also like to see some clear separation of the client and server side code. **DON'T** leave your Javascript and CSS in the header of the html page(s).

And as always, don't ever submit your `node_modules` - just ***DON'T do it…***

Appendix: Example APIs

These APIs may be seen as a starting point to get you thinking, or you may actually use them in your mashup. As discussed above, it will in most cases make more sense to use a service from each list, and even then, not all the combinations will make sense.

Since we last updated this list, there have been two significant changes:

- The Zomato API is no longer readily available. In one of our examples below, we used Zomato for restaurant info, so in practice this would need to be replaced with an alternative. I have left this example in place, but included a warning that is no longer available.
- The NewsAPI has restrictions on the free level – 100 requests per day and some CORS limitations. I have not always warned about this below, but you will need to check this out if you wish to use it.

We have not checked on the current status and timing for API keys or the reliability of the remaining services – these examples are intended to illustrate the ideas. However, most of the services listed have been used successfully in the last year or two by CAB432 students. Note that some seemingly obvious choices – like TripAdvisor – are not suitable (https://developer-tripadvisor.com/content-api/request-api-access/). Some very powerful APIs such as those under Microsoft Cognitive Services may be available under an Azure subscription, but these may be costly if accessed directly (https://docs.microsoft.com/en-us/azure/cognitive-services/cognitive-services-apis-create-account). Have a good look and ask our advice if need be.

You may also find it helpful to consider some limited use of charting libraries such as d3.js (https://d3js.org/) and chart.js (https://www.chartjs.org/) to help display some of the data, but this is not mandatory.

List 1:

1. **Flickr - images and image collections and metadata**
   https://www.flickr.com/services/developer/

2. **Songkick – API for live music information**
   https://www.songkick.com/developer

3. **Zamato – restaurant information – NO LONGER AVAILABLE**

4. **NewsAPI – news articles – RESTRICTED ON THE FREE TIER (See above)**
   https://newsapi.org/

5. **Twitter – Tweet streams and search**
   https://developer.twitter.com/en/docs.html
   Note that API key access can take time

List 2:

1. **Leaflet – interactive maps (JS)**
   https://leafletjs.com/
   Note that you need to choose tiles
2. **Google Maps – full mapping services**
   https://developers.google.com/maps/documentation/
   Note that Google maps requires a billing account to get an API key.
   Not usually an issue, but there are access limits and potential charges

3. **Edamam – Recipes and Nutrition**
   https://developer.edamam.com/

4. **Spotify – music metadata and streaming**
   https://developer.spotify.com/documentation/web-api/
   https://developer.spotify.com/documentation/web-api/quick-start/

5. **HostIP – geolocation from IP addresses and other services**
   http://www.hostip.info/use.html

In many cases, the ideas are immediately obvious – though I am making these up as I go along and some of them will not work all that well when you step through them manually. If the results don't seem all that interesting, they probably aren't. But let's give it a try:

- Suppose I were able to search for a restaurant using Zamato ***[API no longer available]*** and then I find out whether it is healthy by using their menu information to search the Edaman nutrition API.
- I take news tweets from twitter and find out more about them using the news API (yes, those are both in List 1).
- I search Spotify for artists similar to my favourite band (caveat - you will need to check that this is available in the restricted API) and then use Songkick to find information about their gigs.
- I search Flickr for location tagged images on a particular topic and display them on a map using leaflet or Google Maps (or Bing Maps https://www.microsoft.com/en-us/maps/create-a-bing-maps-key and others).
- Search NewsAPI for articles on a particular topic, and strip the URL of the published article back to the host – like www.abc.net.au. Use the HostIP API to find the IP address and then to geolocate the publisher. Then show the articles on a map. [Note, it doesn't matter for our purposes if there are errors in the location. Note again that there are now some restrictions on the free level of the NewsAPI.]

These are some ideas which I have thrown together which you are free to use. Past student examples have included Zamato + Flickr + Maps (get nice pictures of the food on the menu, and optionally show them on a map), NewsAPI + Flickr (often for nature and celebrity news) and a wide range of other alternatives. Both of these examples may require alternative APIs – some replacement for Zamato to source the restaurant information, possibly a less restrictive media API.

When you deal with news sources or tweets it is often necessary to parse the text. Some of this is just about stripping away JSON fields you don’t need. But sometimes you want to tokenise the strings and identify the names to use as a search term. Other people have done this before ☺. You might find these libraries useful:

- CoreNLP – Stanford Core NLP for Node: https://www.npmjs.com/package/corenlp
- Natural – NLP for Node: https://www.npmjs.com/package/natural

Anyway, that should be enough to get you started. Please explore by hand and reality check your ideas, and talk to us if you aren’t sure that the approach is viable.