CAB432 Cloud Computing
Mashup/Docker Project Specification

Release Date: August 8 2016
Submission Date: Friday, September 16 2016
Weighting: 50% of Unit Assessment
Task: Individual Project

Introduction:

The first assessment for the Cloud Computing unit requires that you design a sophisticated mashup that draws upon a set of publicly available services, construct a solution with a significant degree of server side processing, specify the necessary environment as a Dockerfile, and demonstrate deployment to an Azure hosted Linux VM. This first paragraph thus contains a number of different components, which we will consider in turn.

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 applications which draw on cloud data and services
- Reinforce your understanding of web development using modern servers such as node and apache, and client side scripting in Javascript.

The notion of a mashup will be introduced through the lectures and developed in the tutorials. 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. You may also decide that it is worthwhile to explore some service combinations using a kind of manual prototyping. Sadly, the mashup frameworks are now all dead…

However, there are some basics:

- You cannot use existing one-click services. This clause is in place to rule out mashup frameworks, and I will leave it in just in case any still survive somewhere. Applications must be coded explicitly.
- Your application must be non-trivial and involve at least 3 services (details below)
- Your application ***must involve composition of the services.*** It is not sufficient to have simply a web page which displays the output for three services with a common search.

There must be some added value in the results which come back that would otherwise not have been possible – we will explain this through examples.

- Your application must involve a significant server side component (details below). Some mashing on the client side is ok, but really you will find it far more convenient to do this on the server side.

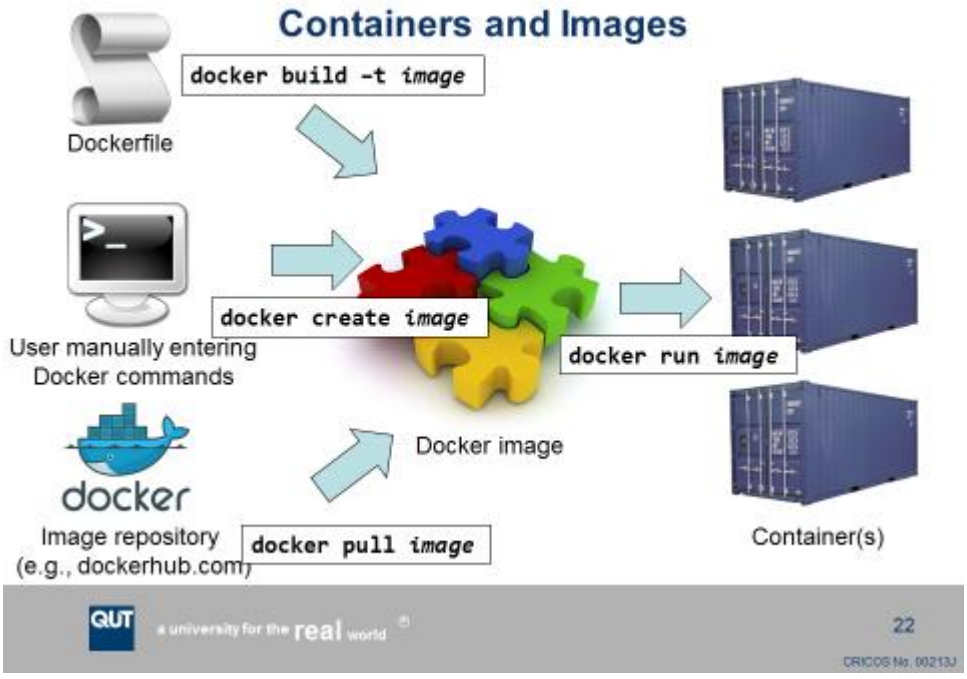
Your work will be assessed according to its technical merit and usability, and on your ability to document the application and its deployment and usage (yes, we require a report). Naturally, these criteria are made more formal in the assessment rubric. There is no explicit assessment weighting given to the 'quality of the idea' as this would not be fair, and it would be very difficult to isolate, and to assess its originality.

We will begin with some discussion of the environment in which you must deploy the application, along with our recommendations for how to develop it.

The Target Environment:

You are required ultimately to deploy the app via a Docker container, sitting ultimately on top of an Azure Linux VM. Ubuntu has been chosen because of its widespread adoption. The specific version (16.04 amd64) was chosen for consistency with the lecture example, but others will work ok. Just remember that this is ultimately the target. For those who do not already have a VM on their machines, please consider Oracle Virtual Box https://www.virtualbox.org/ and the range of Ubuntu images available here: http://virtualboxes.org/images/ubuntu-server/. You may also download the ISO directly from Ubuntu.

You have had a full lecture on Docker and pracs in week 2 and 3 based on the use of Docker and Azure. You should all now have an Azure account and some experience of deployment.



More specifically, we require that you set up (I.e., install and configure) your software stack in a Docker container. To do so, install Docker on your PC (usually under linux) or virtual machine (such as Oracle VirtualBox or VMWare Player), then create a Dockerfile to build a Docker image that contains your software stack. You may attempt this under Windows directly if you wish, but be aware of the issues discussed in lecture 2. We require that this container be deployed to an Azure VM of the specified type and that you leave it running for a period after submission. This may be a separate VM from one used for development. We recommend that push your Docker image to a repository, such as Dockerhub, though obviously this should not be made public.

Other than the requirement to deploy an Azure Linux server, there is no requirement for direct use of the Azure Cloud or other cloud services for this assignment. Assignment 2, needless to say, is a very different story.

Some Basics of the Application:

The nature of mashable services chosen is up to you, but the following guidelines apply:

- Your mashup must involve at least three (3) separate, non-trivial, service and/or data APIs. The combination must make some coherent sense. So use me and the tutors as a reality check, and 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 programmableweb.com.
- The mashup service you provide must support three (3) non-trivial use cases that make sense in your environment. To make this more concrete, if my mashup focus was to provide restaurant information for tourists (say using a search API hitting google or tripadvisor or some other source, 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 that supplies food from my own ethnicity so that I can waste the whole opportunity of travel. (Yes, that one is a little facetious, but it will serve). Broadly speaking, I will see these as separate use cases 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 a use case in which I am seeking Thai food is different from one in which I am seeking Italian food from a restaurant with some minimum hygiene rating. (I am making this up as I go, and I really don't know whether one can find a service that lists hygiene ratings, but you get the point that it would necessarily involve another new method parsing web pages, or perhaps another API.
- 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 be instantaneous or they may take quite a while.
- The actual mashing of the services – the parsing and combination – must involve a substantial server side component, though the mix will depend a lot on the application. This architecture should be explicit in your proposal and we will give clear feedback on whether it is acceptable. As before, the software stack will be organised

through a Docker container, and you will be required to make it accessible – at least to us – from an Azure VM. More details on this closer to the submission time.

- The choice of server is up to you, but there appears to be no good reason to use anything other than node.js, Apache or some other lightweight system like Jetty. You may even use Python WSGI if
- There is little alternative on the client side to javascript. If you wish, you may write in another language and compile to JS (See ScriptSharp et al) but JS owns client side computing and you better get used to that fact.
- On the presentation side, 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 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 mashup project is an individual assignment and there is enormous flexibility in the specification. However it is still worth 50% of your assessment, so we need to be careful in ensuring that we are able to produce something that really is worth 50% of a 12 credit point unit. So there will be two 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.

[Week 5] : Proposal: During week 5 – and certainly by Monday, August 29th, the beginning of week 6 - I expect a one pager with the following information to be submitted via email to your tutor and cc'ed to me (see details below):

- Overall mashup purpose and description (1-2 paragraphs)
- At least three 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 (up to 1 paragraph), and a list of the services to be used for each user story (see above).
- 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 Proposal] to your tutor – either m4.chandler@qut.edu.au or nathaniel.jones@connect.qut.edu.au – and cc'ed to me at j.hogan@qut.edu.au.

[Friday, September 9] : Report Draft: A week before submission, I expect an updated version of the proposal which fleshes out explicitly the details of the API support of the user stories, a discussion of the technical issues encountered and how they have been – or are being ☺ -

overcome, test plan and results and a basic user guide. I will provide more details of the precise format in the submission guide. As well as providing an update in preparation for submission, this will provide an opportunity to flag any projects which are not up to scratch. Please send this to the same set of emails as used in the proposal.

[Friday, September 16] : 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. I will provide you with a submission link and detailed instructions – including details pertaining to the Azure VM – closer to the due date.

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. Note that it is ***NOT*** permissible to leave your javascript and CSS in the header of the html page(s). NO!

The report should be liberally enhanced with screenshots and include the following sections:

1. Introduction – mashup purpose and description (updated as needed from the proposal). This should include an introductory section, outlining the overall purpose and some idea of the sorts of usages supported, but described informally (this is probably 3 or 4 paragraphs). You should then have a description of the services used, with URL and one paragraph description of each. Don't get to the deep API level here.
2. Mashup use cases and services – this section should outline explicitly the use cases supported by the mashup (these *must* be illustrated using screen shots) and the service API calls used to support them. This is a semi-technical description, probably occupying about a page per use case.
3. Technical description of the application. This is a deeper discussion of the architecture, the technology used on the client and server side, any issues encountered, and overall, how you implemented the project.
4. Discussion of the use of Docker, configuration choices. You may include excerpts from the Dockerfile and list this file as an appendix to the report
5. Testing and limitations – test plan, results, compromises.
6. Possible extensions – short section.
7. References
8. Appendix: brief user guide

Please get in touch if you have any questions.