**Introduction**

This documents charts my thoughts/decisions etc. as I have completed this project. It is broken into two sections, one documenting my development journey, with the other documenting my efforts to deploy my project to AWS or IBM Bluemix.

**Choosing a Project**

Initially I was thinking of writing some generic service (e.g. a hotel reservation system or a fantasy football league) in C# using the ASP.NET Web API 2 or ASP.NET Core framework. I was planning on using Azure to get these hosted, and was planning on using React.js for the user interface. These are the technologies I’ve most worked with and am familiar with.

However, during Assignment 1 I enjoyed being exposed to the technologies involved there which were mostly new to me (node.js, express etc.), so I decided to iterate on the Assignment 1 codebase.

**My Initial Plan**

* Firstly, I plan getting my environment setup so that I can deploy to Bluemix or AWS. This is something I’m not familiar with at all, so I want to look at it first and ensure I am able to use it. I also plan on documenting the setup.
* In parallel, I want to complete the UI clean-up I started with in Assignment 1. I am using bootstrap[[1]](#footnote-1) for this.
* Add support for an Admin user – only show pages like Add Product to admins.
* Ensure the login/logout functionality works (currently it does not work correctly).
* Implement checkout feature.
  + Requires an orders service to be implemented.
* Stock Management Service.
* Code clean-up and commenting.

**Development Efforts Journal**

**User Interface Clean-up**

In Assignment 1, I had added bootstrap and done some basic styling of the user interface. To complete this within the front-end service, I completed the following:

* Moved the inline CSS in Index.html to its own style.css file.
* Used bootstrap to create a new header, navigation bar and login form integrated into the header.
* TODO

**Add Support for an Admin User**

The idea here is that we can have different user types for customers and employees who may need to update products, add products etc.

To achieve this, I am going to do the following:

* Rename the ‘customer’ table to ‘users’.
* Add a new column in this database indicating user type. This will be an integer value. 1 will be used to identify employees, and 2 will be used to identify customers.
* I’ll then need to modify any code that interacts with this table and ensure that it works successfully.
* I’ll then review all code that is used to access pages etc. so I can use the user type to determine if a user should be allowed to access that page.

I added some new cookies, one called user\_type and one called customer\_id that I use in the application.

For example, I use user\_type to determine wheatear or not to show certain items in the UI (e.g. the admin pages) to the user.

**Creating the Order Service**

As I reviewed the Assignment 1 codebase in terms of what was missing for me to create a functioning application, it was obvious that in order to implement features like checkout, I would need to implement an orders service which would be responsible for processing user orders.

I started by create a skeleton orders service with no functionality that would run on port 3005.

Next in the front-end service, I created express middleware functions in api/orders/index.js in order to be able to call the orders service.

Now I need to implement the actual functionality of the orders service. I started with an API (located at /order) where order details could be posted. This API would take care of updating the database:

* TODO – Check stock service???
* Update the orders table with the high level order details (customer ID, date etc.)
* Update the orderdetails table with the details of the items actually being ordered (tied back to the relevant row in orders).
* TODO – decrement stock levels for these products?

While implementing the orders service, I also discovered a defect in the user service.

The user service was not checking the password provided, only the username. So as long as a valid username was input, the login would always be successful, even if the password for that user was incorrect. I modified the user service to take account of the password also.

**Creating the Stock Service**

TODO

**Deployment Efforts Journal**

My high level plan on deployment is as follows:

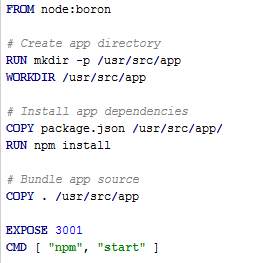
* Come up to speed on Docker in order to containerize my services.
* Investigate AWS as a means of hosting my services.
* Investigate IBM Bluemix as a means of hosting my services.

The rest of this section outlines my efforts to containerize my services and deploy them to a cloud service.

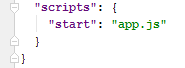
**Creating a Dockerfile & Docker Image for each Service**

As I was not familiar with Docker, I decided to take a ‘baby steps’ approach and begin by creating a Dockerfile for one of my web services, the user service. I used the online Docker documentation to come up to an initial level of understanding on Docker, as well as using the notes from Lecture 10.

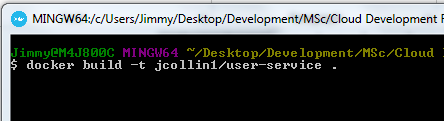
For the user’s service, my Docker file looks like this:



Note that I had to make a change to package.json in the user service also, due to the main file being called ‘app.js’ and not ‘server.js’ in this service:



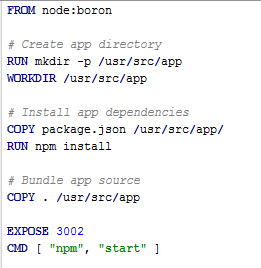
Once I had my Dockerfile created, I used Docker for Windows to create a Docker image using the following command:



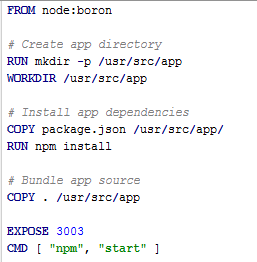
Now I had my user service Docker image, so I decided to create one for each of the required services, including the new services I was working on (the Stock Management service, and the Orders service).

Below are the Dockerfiles I created for each of these.

*Catalogue Service*



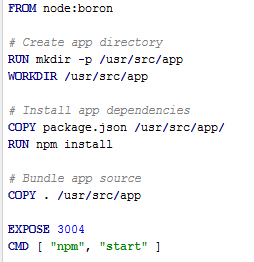
*Cart Service*



*Front-end Service*



*Stock Management Service*



*Orders Service*



Now that I had a Docker file for each service, and an image created, the next step was to test out actually running the images as Docker containers on my local machine.

To do this I used the below command (in this instance for the front-end service) to get each container running:



The full list of commands I used to get the services running is:

* docker run -p 3001:3001 -d jimmyc/user-service
* docker run -p 3003:3003 -d jimmyc/cart-service
* docker run -p 3004:3004 -d jimmyc/stock-service
* docker run -p 3005:3005 -d jimmyc/order-service
* docker run -p 8079:8079 -d jimmyc/front-end
* docker run -p 3002:3002 -d jimmyc/catalogue-service

***Note****: About this time I had a conversation with someone about Docker Compose, and decided to investigate it later. My priority for now was to make sure I could containerize each service and get them running successfully on my local machine.*

I had a couple of problems with dependencies during the process of getting my services up and running in Docker containers. This was due to the fact that the Assignment 1 codebase has a ‘*node\_modules*’ folder in the source tree, but the latest version of these dependencies are grabbed when the node app is being created in the Docker container.

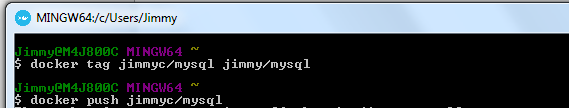
Whenever a service was acting up or not behaving as expected, I started that container last, and without the ‘-d’ parameter in the ‘docker run’ command. This allowed me to see the stack traces for any issues that happened.

Another problem I’m having is that most of the services that interact with the database have a hard coded localhost variable pointing at the database – how to handle this locally (by hardcoding?) or in an environment like AWS or IBM Bluemix?

TODO

After I had created each of my Docker images, I decided to create an account on DockerHub and upload my images here as a backup.

To do this, I used the following commands to create a tag and push each image:

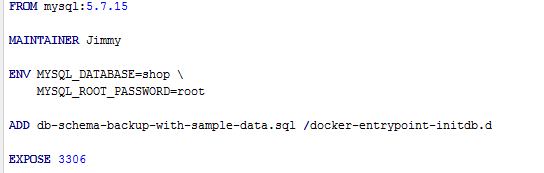


These can be viewed at <https://hub.docker.com/u/jimmyc/>.

**Creating Dockerfile and Image for MySQL Database**

At this point my services were running fine in Docker containers, but without the MySQL database they would not be much good.

To create this, I created the below Dockerfile.



After some trial and error, the above can be run using the below command and it creates the MySQL container and also runs the necessary SQL (stored in the db folder in the source tree) to setup the required tables in the database.

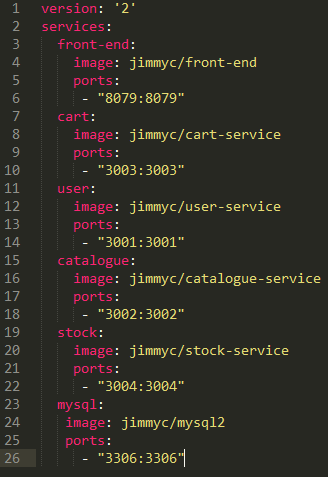


**Utilizing Docker Compose**

As mentioned earlier, I had a conversation with a colleague who had recommend that I look at Docker Compose as a means of defining and running my multi-container Docker application.

Through some research and reading of the Compose documentation[[2]](#footnote-2), I was able to get a simple Docker compose file up-and-running that creates the necessary containers for my application in one Docker Compose command (docker-compose up).

My Docker Compose YML file is shown below. This is committed to the root of the source tree also.



**Attempts at Deploying to Amazon Web Services**

**NOTE**: I had initially decided to try to deploy to AWS, but midway through I decided to back-out of this due to two things:

* I decided that the complexity and learning-curve of AWS were too much in the time I had to do this assignment (especially as I had a lot of code to write also).
* I started to get charged for my AWS usage (only $10.01 albeit) – I had signed up for the free tier usage, so I’m not sure what happened, but when I saw this and my next month projected bill of $133, I immediately cancelled my account.

I have included the steps I had taken on AWS here anyway.

I learnt that you can deploy Docker containers using the Amazon EC2 Container Service[[3]](#footnote-3).

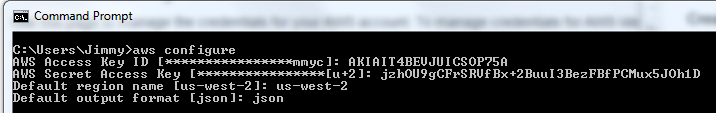
The first thing I done was to sign up for the AWS Free Tier (or so I thought).

Next, I got setup in ECR as per the documentation[[4]](#footnote-4).

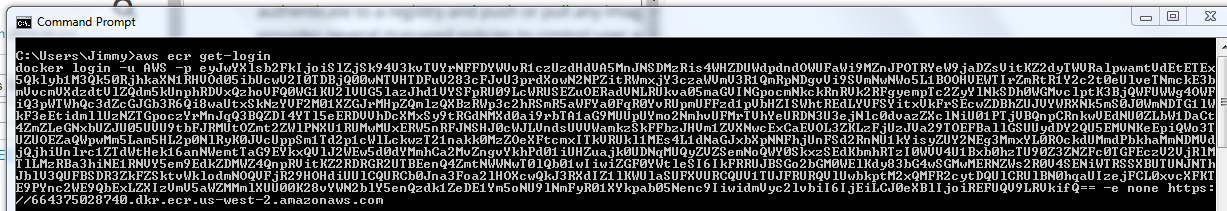
In order to be able to push Docker images to AWS, I would need to install and configure the AWS Command Line Interface (CLI).

I installed this on my Windows machine and created a new access key via my Security Credentials dashboard in AWS.

After installation, I next ran aws configure from my command line to configure the CLI:



This gives us the command we need later to authenticate our Docker client with the Amazon ECR registry:



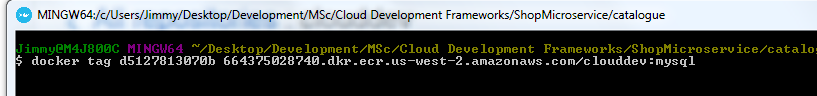
(Authentication tokens are valid for 12 hours)

Now my AWS CLI was operational, so following the instructions here[[5]](#footnote-5) I begun by creating a new repository to store my Docker images.

Next, using Docker I authenticated with the repository I had created in AWS:



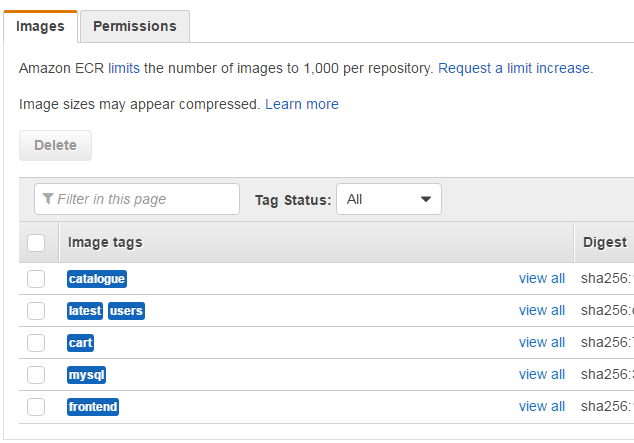
I then tagged each of my images for deployment to EC2, e.g.



Finally, I pushed each of my images to the repository in AWS:



Now I could see my images in the repository in AWS:

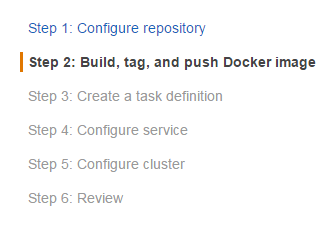


Now that my images were deployed, the next thing I needed to do as per the AWS documentation was to create a task definition[[6]](#footnote-6).

“*A task definition is like a blueprint for your application. In this step, you will specify a task definition so Amazon ECS knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.*”

**NOTE**: Soon after this I began to start getting charged for AWS usage, so I closed my account.

It looks like I was on Step 3/6 in terms of getting my services up and running on AWS:



**Attempts at Deploying to IBM Bluemix**

After my failed attempts to get my application up on AWS, I decided to look at IBM Bluemix, the recommended solution for getting our final projects deployed to a cloud service.

First off, I signed up for IBM Bluemix using my CIT credentials in order to get a free trial (hoping that it wouldn’t end up like my ‘free tier’ on AWS).

I then followed the instructions[[7]](#footnote-7) to install the Cloud Foundry CLI and Bluemix CLI tools on my Windows machine.

After completing these steps, I ended up with a private container repository on IBM Bluemix, located at registry.ng.bluemix.net/jimmyc.

First I needed to authenticate correctly with Bluemix to push images. From the Docker CMD, I ran the following commands:

bluemix login (login with Bluemix credentials).

bluemix ic init

Next, to upload my images to this repository, I tagged each one for upload:



Then I uploaded each one individually, specifying the tag:



Now it looks like my images are on Bluemix:



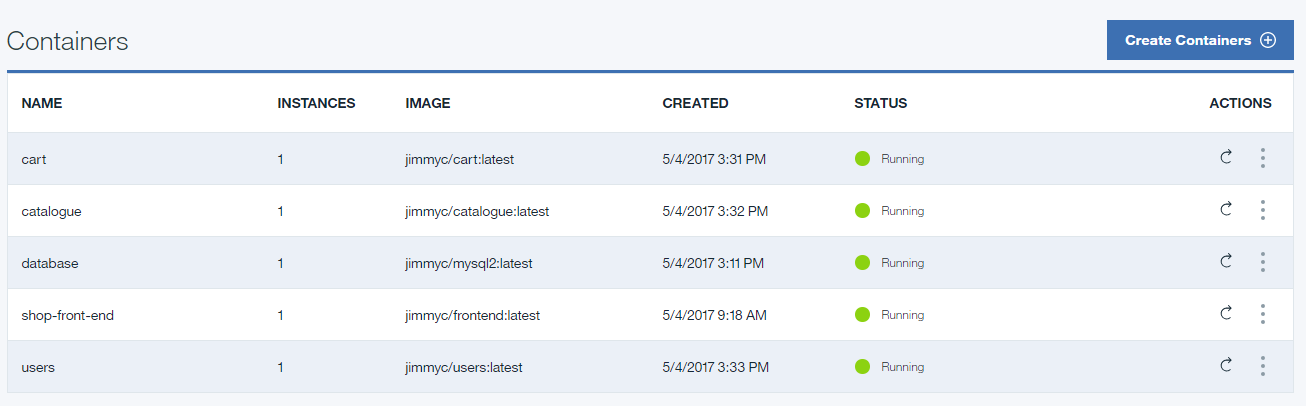
Questions I have at this point:

* I’m still concerned about how my services talk to the database (as mentioned earlier each one has a hard coded localhost variable used for the database).
* How/can I utilize Docker Compose in Bluemix?

At this point, I tested spinning up my front-end service on Bluemix to see if it would at least show the user interface (the database etc. would not work yet).

I spent a good few hours wondering why my initial attempts didn’t work, until I realised that our corporate network firewall was blocking all the traffic due to the UI being served on port 8079 – serves me right for working through lunch!

At this point, all my services are running successfully in containers in Bluemix:



The problem I now need to look at (and am not sure about) is how these services communicate with each other in Bluemix.

1. <http://getbootstrap.com/> [↑](#footnote-ref-1)
2. <https://docs.docker.com/compose/> [↑](#footnote-ref-2)
3. <https://us-west-2.console.aws.amazon.com/ecs/home?region=us-west-2#/getStarted> [↑](#footnote-ref-3)
4. <http://docs.aws.amazon.com/AmazonECR/latest/userguide/get-set-up-for-amazon-ecr.html> [↑](#footnote-ref-4)
5. <http://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html> [↑](#footnote-ref-5)
6. <http://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html> [↑](#footnote-ref-6)
7. <https://console.ng.bluemix.net/docs/containers/container_single_ui.html#container_gettingstarted_tutorial> [↑](#footnote-ref-7)