**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.
* Clean-up login functionality?
* Implement checkout feature?
* Support logout functionality?
* Investigate Discovery Service for REST?
* Stock Management Service?

**Development Journal**

**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 wheater or not to show certain items in the UI (e.g. the admin pages) to the user.

TODO – Document coding briefly as I go…

**Creating the Order Service**

The orders service is critical to a number of functions, so it needs to be implemented.

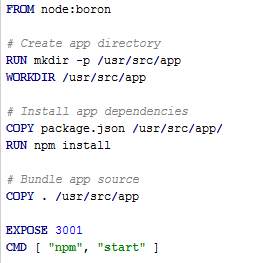
I created a skeleton orders service that will run on port 3005.

**Deployment**

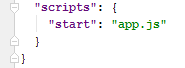
**Creating a Dockerfile & Image for each Service**

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 11 (TODO: CHECK THIS).

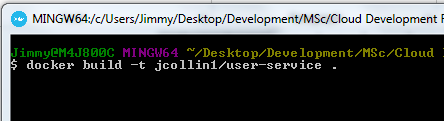
For the users service, my Docker file looks like this:



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:



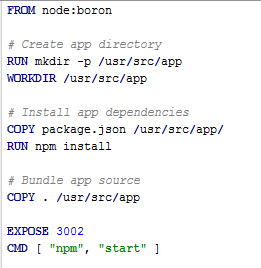
Once I had my Dockerfile created, I used Docker for Windows to create a Docker image using the following command:



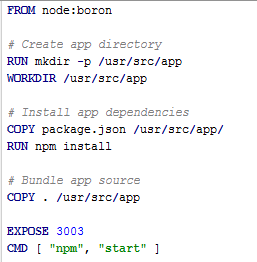
Now I had my user service Docker image, so I decided to create one for each of the required services, including the new service I was working on (the Stock Management service).

Below are the Dockerfiles I created for each of these:

*Catalogue Service*



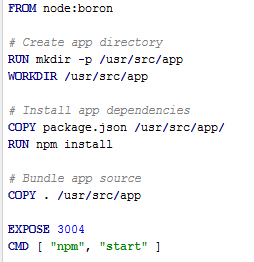
*Cart Service*



*Front-end Service*



*Stock Management 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 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.

TODO – Talk about deploying to DockerHub.

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

**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 pulled down the MySQL Docker image from <https://hub.docker.com/_/mysql/> and got it running on my local machine.

I started this container using the following command:



Next, I connected to port 3306 on 192.168.99.100 using MySQL Workbench and ran the SQL to create the shop DB.

A question occurred to me at this point – how do I do this initial setup of the MySQL database in an automated fashion?

Deploying to AWS

TODO – include nodes but indicate that I backed out due to complexity.

Deploying to Bluemix

TODO

**Bug Fixes Notes**

**User Service**

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

1. <http://getbootstrap.com/> [↑](#footnote-ref-1)