Arbnor Aljilji

Abstract

Documented evidence of Arbnor Aljilji implementing and demonstrating the KSBs.

Dev ops engineering apprenticeship

T. Rowe Price Portfolio

Contents

[Introduction to myself, the team, and projects 2](#_Toc97820233)

[Health and Safety 4](#_Toc97820234)

[Data Protection 4](#_Toc97820235)

[Minimum KSBs met in Assessment 2: 5](#_Toc97820236)

[K3, K6, K9, K18, K19, K20, K21, K22, K23, K24, K25, S1, S2, S4, S8, S13, S16, S21, B1, B2, B4 5](#_Toc97820237)

[Project Unity ChatBot 6](#_Toc97820238)

[Project Developer Desktop Self-Service Portal 10](#_Toc97820239)

[Project Resource Enhancements 15](#_Toc97820240)

[Resource 1 (Unity-Elasticache) 15](#_Toc97820241)

[Resource 2 (OpenSearch-Domain) 18](#_Toc97820242)

[Resource 3 (Snaplogic/Unity-ASG policies) 20](#_Toc97820243)

# Introduction to myself, the team, and projects

My name is Arbnor Aljilji, I’m 28 and I’m part of the DevOps Engineering Apprenticeship at T. Rowe Price. I came from a HR background interested in tech and studied BTEC IT in college. However, I graduated in BA Hons HRM. I’ve always had an interest in tech itself but wasn’t sure where exactly. I narrowed down different tech pathways and found that DevOps engineering was something that interested me, especially with the cloud ever-growing.

I’m part of the Developer Services Group (DSG) in the London team. We are a team of 9 Senior Software Engineers, 1 Product owner, and between us there are specialist skills as SDETs, SREs, and developers. Initially, the team looks after developers who create apps, and our mission is to enable the developers to move as quickly as possible and enable them to push cleaner code at great frequency, and higher volume. Through automation around development tool chain, release management, software quality, builds, and developments, we aim to achieve these goals. Unity is an in-house product designed and looked after by the DSG developers to allow other developers to deploy their application to the cloud, at a much quicker pace.

There are a few projects and tasks that I have worked on which I will mention in the portfolio, but my main KSB were met in my Unity ChatBot project as it was covered from the beginning to production environment. Other projects came from hackathon ideas such as a user-friendly front-end app for users who can look after their own VDIs. Updating deploy config files to update resources to meet developer needs. The main project is the Unity ChatBot, which currently has met its MVP state. Further features will involve being able to deploy to the cloud with a push of a message without needing to access our internal Unity Deploy app that does that for us.

T. Rowe Price week schedule differs from the usual technological diary. However, I have 2 fixed meetings are and different tasks throughout the day:

- Every Monday morning at 9.30am I have an hour meeting with my line manager to discuss my goals for this week, and what I have achieved last week. We either note it down paper, confluence or write a Jira ticket

- Every Friday morning at 10am I have a team meeting with the DSG team to discuss what support issues we have had in the Linux Developer Desktops (extra responsibility in looking after the Linux VDIs), or any other discussion such as projects etc.

- Studying AWS SAA, NodeJS, Linux, Dev Tools, and attend other meetings required or optional

The work is assigned via tickets in Service-Now or Jira and the team and I work through them. For example, if a client has an idea for a product we own, they would create a question/user request on AskTrusty (similar Stack Overflow concept) and then a ‘feature’ ticket is raised for the team of the app to implement a new feature to the application. Another way of receiving tickets is when our clients have issues with their LDD, and the DSG team supports them from 9am-2pm, in which support is then handed over to the US team. Unity Folio, which is an internal app designed to make the SDLC a smoother operation for many developers. This app allows developers to upload their app into their cloud by themselves and can be done via pipeline or the UI. The focus is to improve the automation of the procedure, to minimize the work time to upload your work and automate any procedures into deploying their apps and maintaining them. Lastly, my project sits under one main ticket ‘Unity Boy’ with sub-tickets underneath to define each task I will be working on monthly.

**Notice**: TRP doesn’t really follow the standard technological structure many people follow; we have our own way of working which works best for us. However, different cultures work differently as I will highlight throughout the portfolio. When working with the US team, I noticed we done sprints, however, in the UK I rarely needed to do sprints.

---- Give examples above -----

# Health and Safety

Signed off the health and safety training at T. Rowe Price and Makers.

# Data Protection

There are confidentiality regulations and policies I will follow when uploading work, however, due to the nature of my examples provided, there will be no confidential information uploaded.

Minimum KSBs met in Assessment 2:

## K3, K6, K9, K18, K19, K20, K21, K22, K23, K24, K25, S1, S2, S4, S8, S13, S16, S21, B1, B2, B4

Additional KSB met in Assessment 2:

# Project Unity ChatBot

**KSBs acknowledged**:

**Intro to project**  
Unity ChatBot is a chatbot for the Developer Services Group chatroom. It was designed to do API calls to specific internal apps from the Symphony chatroom. As MVP product, for example, seeing who is on support on xmatters and displaying it to the user.

**How I landed the project:**

The app got first introduced to me as a potential project idea from my Line Manager and colleague. It caught my interest as I saw the value it could bring to the team, and learn to code in a new language, as I wasn’t comfortable in JavaScript, NodeJS.

**Current usage and future cases:**

Currently:

* Xmatters call to display rota shifts

Development:

* Notifications for 9am support ticket summary from servicedesk

Future:

* Deployment of AWS resource from the bot

This project has potential future uses from a larger target audience as one of the future features is to be able to deploy a resource using the chatbot without needing to use our deployment app. It will cut down a lot of manual steps, which saves the team time and effort to do it manually. The current feature I’m working on in deployment is to set notifications for 9am summary for support tickets that the clients get.

**Business value**

The current business value the bot has that it saves time for the client to log into another app and see who is on support.

* Removes toil
* Quicker process: usual would be 1 or 2 minutes, bot takes 3 seconds
* Easier for clients to see who is on primary, secondary, and execution support level
* By seeing who is on call quicker without the need of using another app, the team members can discuss any current issues quicker with the member on support.
* Current development feature: summary of support tickets gives more of a structure in the
* With the current feature I’m trying to code, I want the clients to be notified in the morning of any support issue tickets that have been raised. As the team and I support Linux Developer Desktops, it would be helpful for the team to know how many tickets came in, so they could tackle the issues quicker.
* Lastly, just building an open source chatbot for the community with small features makes it easier for the community to understand how to collaborate and implement their own features, by following the documented README.md file.

**Planning Design Phase**

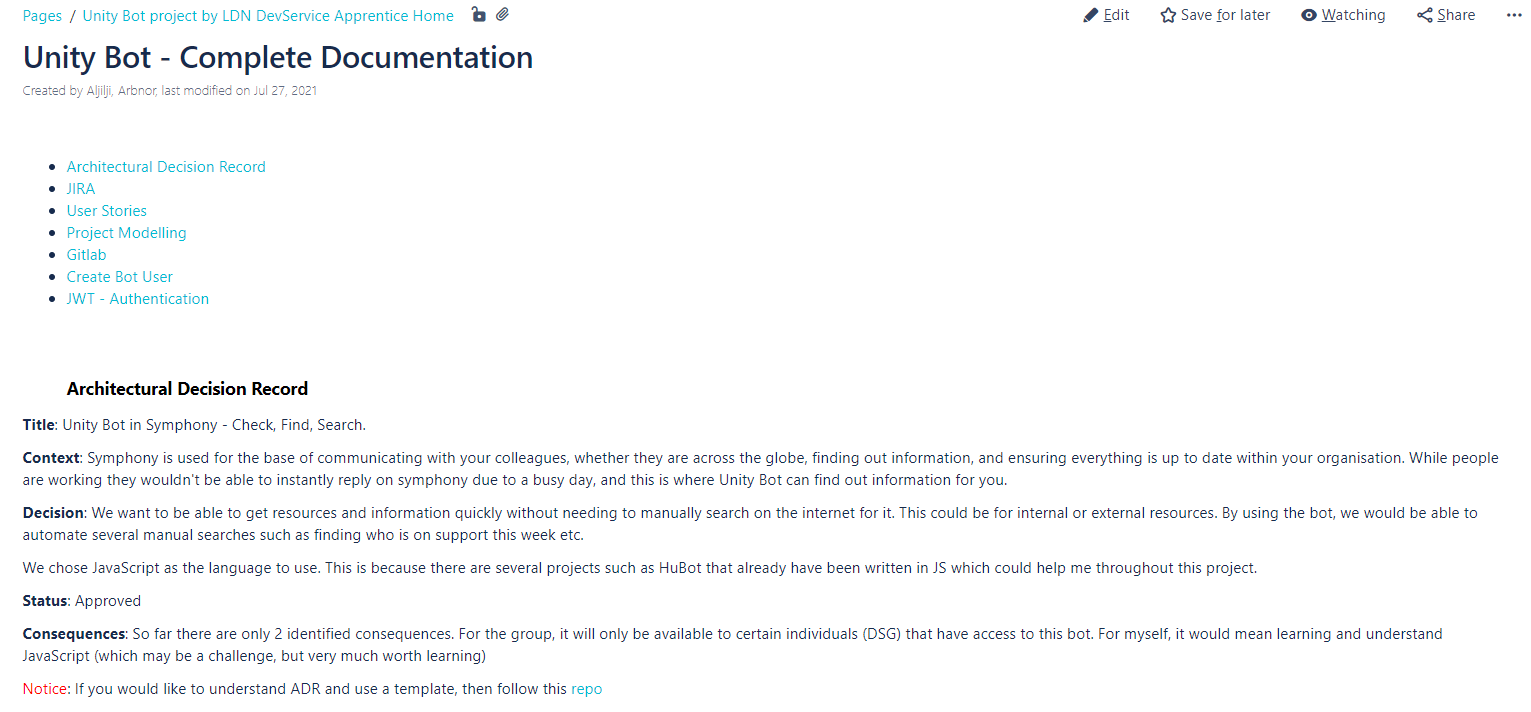
**Introducing Project lead meeting**

The initial step of the project was first to be introduced to my project lead. I had a meeting with both the then to be project lead, and my line manager. Discussions were made on what the project is about and what value it will bring to the team. I decided to have weekly meetings with my project lead from then on forward every Tuesday at 10am for an hour, and if possible, every Wednesday at 10am too.

Before setting up the project, I decided to use confluence as a place of storage for the documentation of the project. It contains various headings of the Architectural Decision Record, JIRA ticket, User Stories, Project Modelling, Gitlab, Creating User Bots, and the JWT authentication steps. This documentation helps me, and others understand how to use the bot, the future design of the bot, and how to make their own. **Reason**: confluence is easy to use and easy for everyone to access the documentation of the project as its public.

As this was the planning phase, using zoom as the communication method felt more suitable for me to discuss the project with my project lead. It made it easier for me to ask questions and note down any comments made. This led to the design of the Architectural Decision Record while discussions were being made. See figure 2.0. I had meetings with my project lead for 1 or 2 hours a week, so I can discuss what progress I’ve made on the project so far. Due to covid, everything had to be done virtually which made it harder to be able to always discuss the bot as being in person meant I would be able to see the lead more. However, I adapted to this and kept ensuring to have regular weekly catchups. The discussions throughout the weeks covered pair programming (code review), cloud infrastructure architecture, current blockers, show and tell, meetings with external parties, and learning development/AWS SAA.

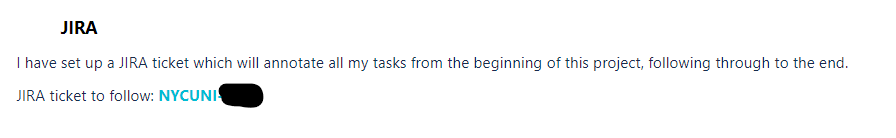
The planning phase needed a documentation page, which is below.



*Figure 2.0 Confluence page documentation*

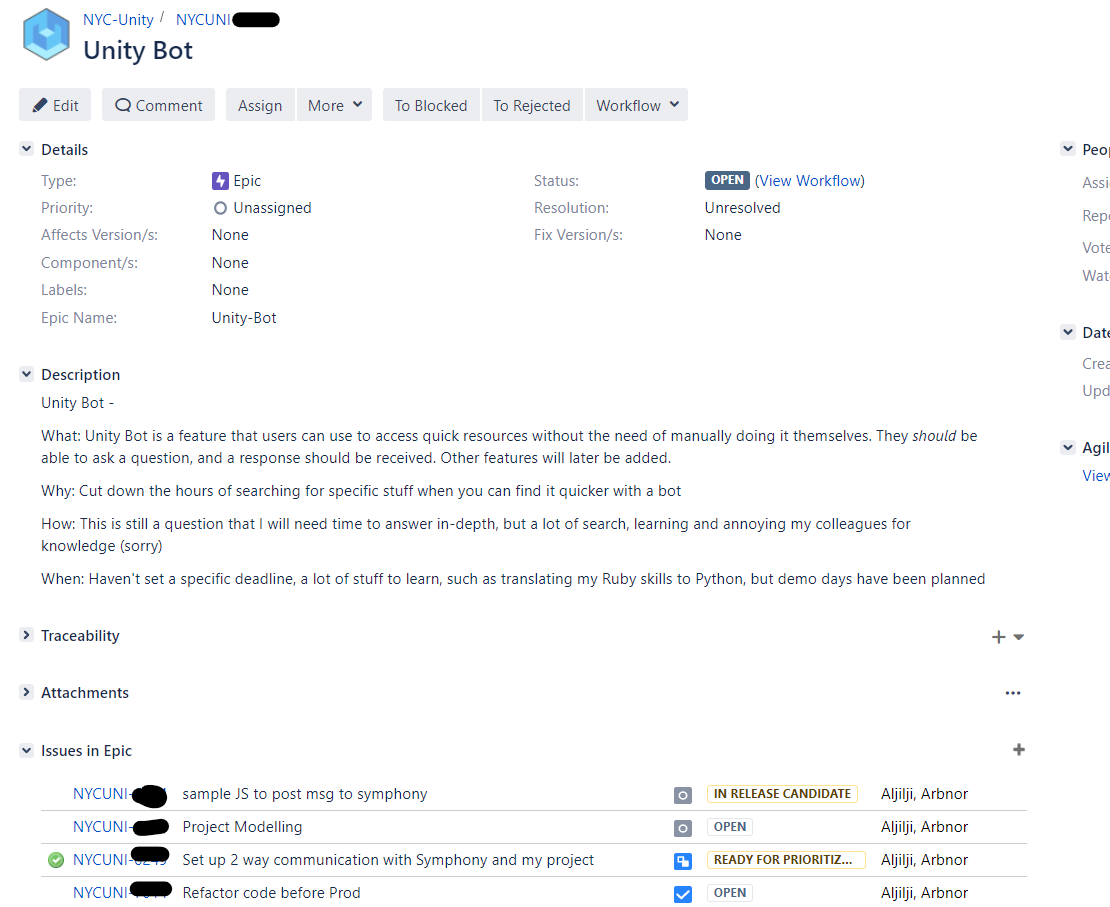
A lot of the architecture design was suggested by the project lead, involving the programming language and the AWS resources used to get the app live and running as they were more experienced than me.

Step 2 – JIRA Ticket



*Figure 2.1 JIRA ticket*

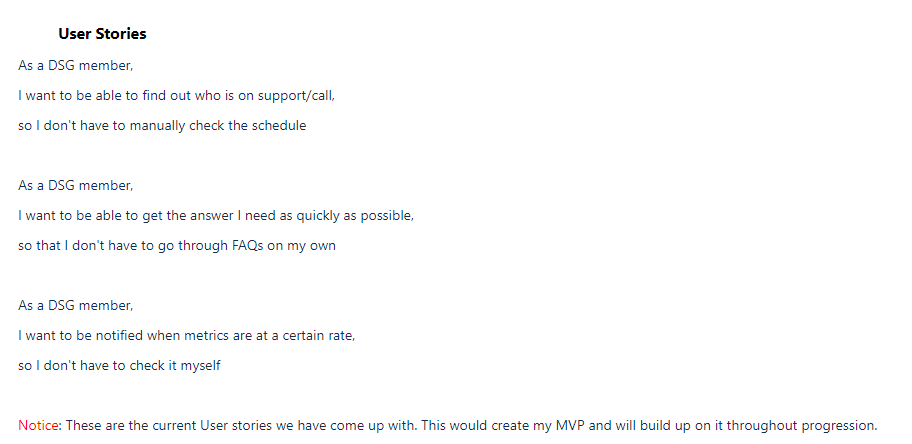
The above screenshot is in the documentation, which mentions the main JIRA ticket which will be posted below.



*Figure 2.2 Unity Bot JIRA ticket*

This is the main JIRA ticket that I used throughout the project to create any sub tasks within the project, as seen in the ‘Issues in Epic’. The reason being is that having all the tasks in a centralized place, or one main ticket, it is easier to trace on what I’ve done so far, what I will need to do, and what blockers I’m currently on, and lastly, my project lead and manager have an idea on what task I’m currently working on.

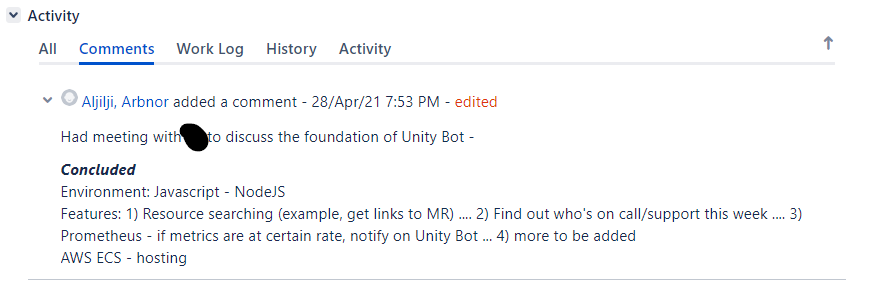
As the DSG team were my initial clients for the use of this app, I had to see what the MVP of the product should be. This is where I designed the user stories which helps me translate what the product service should be offering. User stories help understand the foundation of what the client’s needs and wants are.



*Figure 2.3 User Stories – notice: User Stories have changed since the documentation stage*

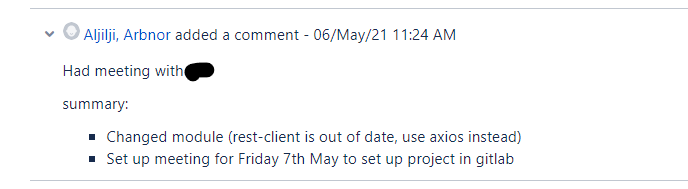
The next few paragraphs will highlight what steps I did to get my project up and running, before implementing any code.

As the documentation above had been completed, it was time to set up my VDI so I can have the right software stalled such as the IDE and NPM etc.



*Figure 2.4 Meeting April 28th – VDI setup*

This was the very first ticket in my project as shown in the JIRA ticket above. Setting up the environment was proving more difficult for myself so I asked my project lead to jump on a zoom call to get NPM installed etc. Being able to message my team directly on a live one-on-one communication such as Symphony makes it easier to communicate rather than waiting for an email response, which is why every time I was stuck within a task, I would shoot a message to the team. As it was my first-time using Linux, I wasn’t aware of how we get software’s installed into our VDIs. Due to security reasons, we have an internal component (TRPbrew) which is like brew itself to install languages, softwares etc. Knowing that there is this security vulnerability meant that ill be more aware of not installing directly from the internet any harmful libraries that could impact the firm. By being vigilant, it makes myself and the company more safe from any malware attacks, and so. My project lead set up my environment by doing the right configurations in my PATH settings. This took me a day to complete as our internal app was having some issues with redirecting node/npm from the internet to our internal component, meaning, I had to wait for that issue to be fixed first. During this time of waiting, I decided to do some studying on Udemy and Codecademy to prep for using JavaScript. Once that issue was resolved, my environment was completed.

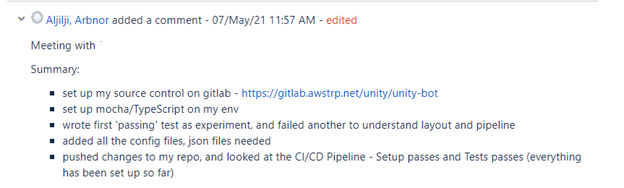


*Figure 2.5 Module change*

As this was part of the planning, the following week I had a zoom meeting with my project lead to which I learnt how in depth REST api works as that’s what essentially my app will be doing, and what certain modules already exist in the NPM package to use. Rest-client was the first module my lead had thought of, but due to security reasons that the module had not been updated for a long time, I researched with the tech lead on what other libraries we could use, to which we found axios.

Next step, get my project set up in gitlab.

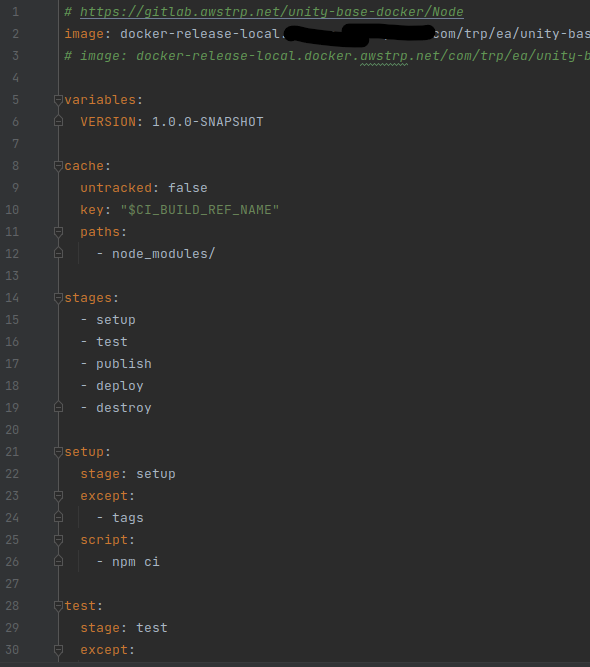
I had to find out where I will be storing my coding in a centralized control source. Gitlab is the choice where I can upload my code changes and push to, thus using continuous integration (**k15**). The reason of having this centralized source control with different versions of the app every time I push the change, is that if I mess up in my project, I can always remove that push, and retrack back to an older version. Other reasons being able to spot bug fixes easier if I’m committing small changes more frequently. Nonetheless, the community can also see the project as it’s a public project so they can clone it, or fork the bot.



*Figure 2.6 TEA integration and Gitlab setup*

With testing in mind, I had to set up my testing reports to TEA, which is an inhouse Testing component of Unity that is designed to put all the quality evidence under one place so that an informed risk/reward release decision can be made by Release Managers or Change Approvers.

Due to the nature of the task, I knew that I had to plan a meeting with a skilled SDET in setting up the testing environment for my project. The agenda of the meeting was to explain what the architecture of the project and get advice on what testing frameworks can be used. As this was my first project, I needed guidance in setting up the project within GitLab and ensuring every setting has been set up correctly. As I knew testing was not my strongest skill, having this meeting with the SDET team member made it easy for me to set up the project, and get advice on what testing framework I would need for the project to be successful. Thus, making my first step of the project setup complete. Below is the gitlab-ci.yml file that I managed to make with my team.



*Figure 2.7 Gitlab-ci.yml*

I knew I had a testing environment, and that I could start coding the project. Following the TDD principle meant that I should test code before implementing code. However, due to working with different team members, I was advised and shown various options. For example, when I worked with the SDETs, you’re taught to initially test before you code. Further down the line however as will be shown, working with another colleague, I done some coding before I tested the code. Reason being, everyone works differently for various reasons. The tech lead and I wanted to see if the function that the API call does would get a call back. Once I saw that the API call works with our network settings (due to proxy settings), I was able to refactor code and do proper testing.

As the testing framework had been completed, it meant time to structure the skeleton of my project. Thus, moving the files and folders around. For example, the testing environment sat in the testing folder, and the main functions for the project will sit in the src/functions folder. This clear structure of the folder layout made it easier for me to link resources together when importing functions into different files, but also, it would be easier for other developers to navigate throughout the folders if they were to contribute to the app itself.

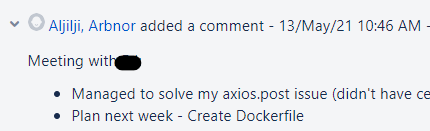
As the project was living in gitlab, it was time to create the Symphony ChatBot account in the development environment. I spoke to my Manager on our Monday 9.30am standups for guidance on how to create a symphony bot, to which I was shown a documentation on how to fill in the form to create the Dev ChatBot. Once I had completed the form, I had to wait a day approval and I was sent the pemfile of the chatbot so I can store the creds somewhere in vault for prod, and locally for now.

After creating the chatbot service account, I needed to get access to Symphony test environment where I would be able to test my development code. This is when I joined the Smyphony Developer chatroom and asked for access in the Symphony Test environment. By getting this access, I saw that my chatbot has been added to the symphony environment, and I had everything setup to start coding.

**Implementation Phase**

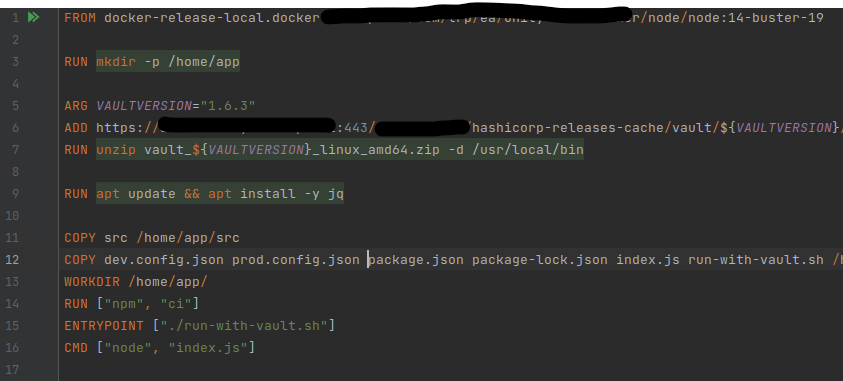
Now that the planning has been placed, environment has been set up, account has been created, it was time to start coding.

As I had the gitlab-ci.yml and README set up, I had to do my first production task. The next task to my project was to create the Dockerfile.



*Figure 2.8 Dockerfile template*

I remember the simple context of what Docker is, and Dockerfile. However, creating the Dockerfile is more advanced within a firm so I set up a meeting with my project lead to understand the steps of the Dockerfile again. I created a basic Dockerfile using the Docker docs, but then pair programmed with my lead to update the Dockerfile in accordance with TRP standard.

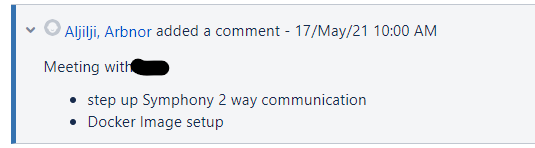


*Figure 2.9 Dockerfile*

Completing the Dockerfile creation meant that I was ready to move onto my next task.

Next on the agenda:

Symphony Authentication documentation



*Figure 3.0 Setting up Authentication*

This was a difficult task for me to wrap my head around the concept, so I asked my project lead to jump on a zoom call. I was able to discuss with my project lead the issue I’m facing and managed to go to the Symphony documentation to understand how authentication works and how the bot will listen to commands applied to it. Going through the Symhpony docs, I saw that Smyphony provides an SDK.

Looking at the main file, I saw that it accomplishes four things:

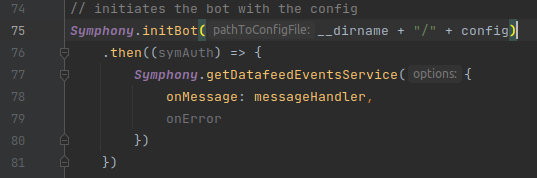
* Configurations for my bot
* Authenticates my bot
* Starts up the bot’s datafeed event service
* Adds custom event listeners to the bot’s datafeed event service



*Figure 3.1 Initializer for bot*

The documentation above led to the next step of implementing the config.json file which will contain the authentication using the pemfile received from creating the bot user prior.

Below is the symphony.init of my own code



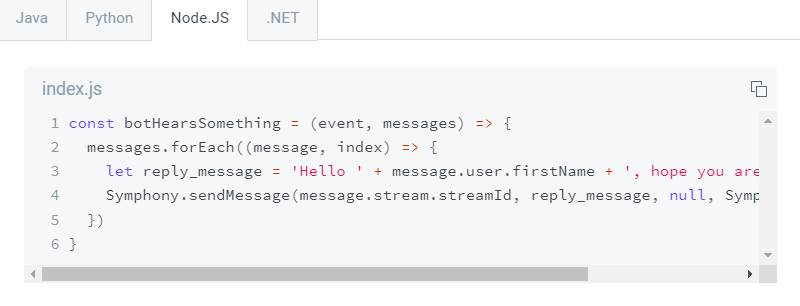
*Figure 3.2*

Below is a screenshot of the config.json file in local env.



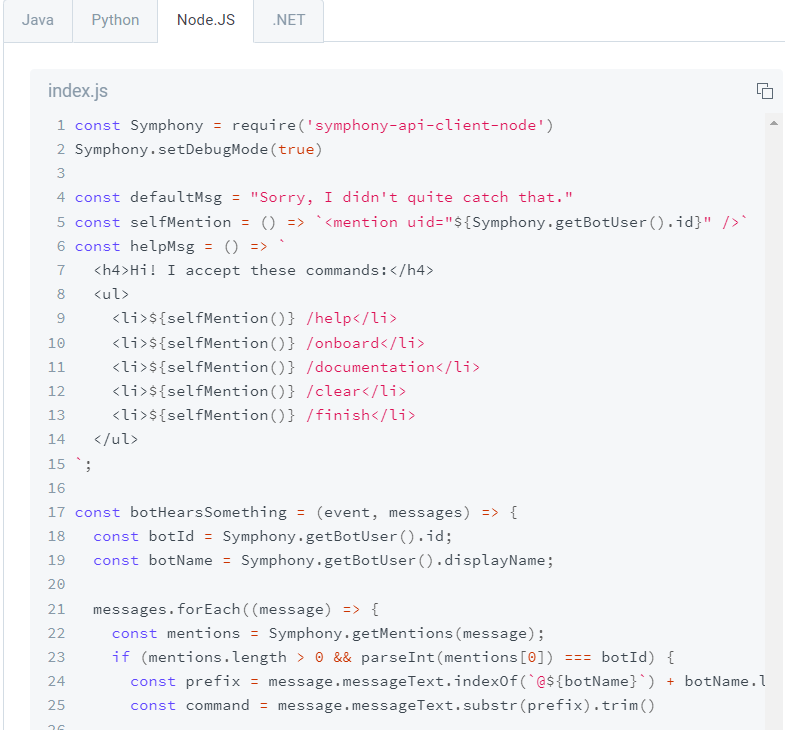
*Figure 3.3*

Now that the authentication has been set up with Smyphony, it was time to see whether it can listen to any commands. This is where the Symphony doc helped me write the event listener. However, I had to change my coding a bit differently to what was in the document. Below is what the docs showed, but the design of my bot was that it should ONLY listen to messages it has been mentioned with the @ signal, for best practice that if it is in a room with multiple users, then it doesn’t listen to everyone message.



*Figure 3.4*

To test that the authentication works, I had to create a simple function that returns a value to the user. This is where I followed the guide on how to create a list of commands from the Symphony docs and implemented my own.



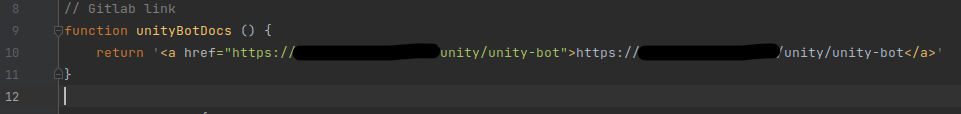
*Figure 3.5*

As seen above, that’s the Symphony doc example on how to make the list of commands and below is the code I implemented using a switch statement for what command the bot will listen to. Before creating any advanced functionality to the bot, I wanted to create a simple function that returns a value when a user mentions the bot with the @ signal.

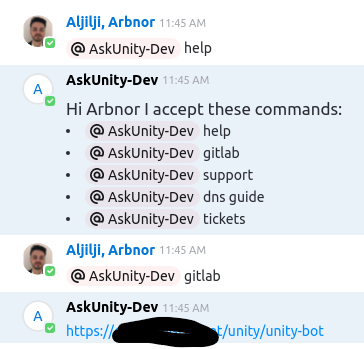


*Figure 3.6*

Below is the simple function that returns a web URL for the user from the bot. To test this, I went to the symphony test environment and initialized the bot for the first time.



*Figure 3.7*



*Figure 3.8*

The authentication was a success, and my bot is being heard from the client side. As I had no experience in testing much, and being new to mocking, I had to set up a meeting with my team member to get advice on it. However, I was able to create the tests for the simple functions early on.



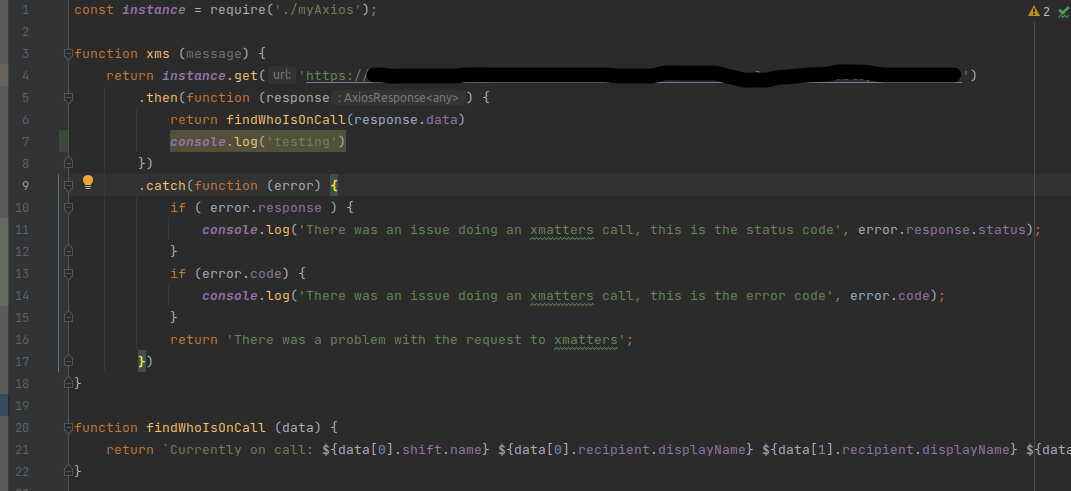
*Figure 3.9*

Tests like these provided only a simple value, whereas the more advanced functions I needed guidance with.

Now that the authentication is working, it was time to write a user story required from the client. The first user story that needed to be worked on was the xmatters rota shift call. The first step to completing this task, I had to get an xmatters access key and password from the website. This is so the chatbot can do an authentication login and do the HTTP request. This was a simple step of logging into xmatters and going on settings to get these credentials.

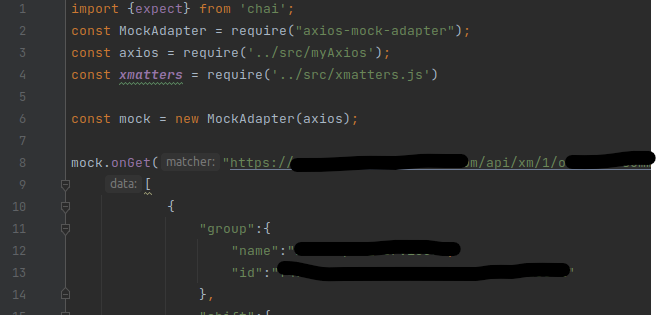
Once I got the credentials, I stored them locally for now so in my IDE so I can test the new function I’m about to create.

Below is the function I created with my project lead as I wasn’t sure how to tackle the axios library, and I ended up pair programming for the next 2 days with my lead.



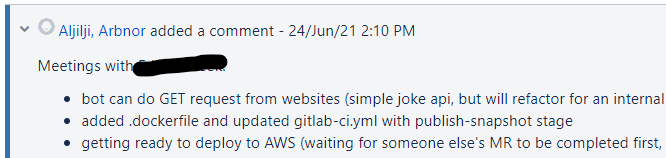
*Figure 4.0*

After trial and error, we tested this functionality in the symphony test environment, and it worked. The chatbot was able to do an API call and return the results of who is on call.



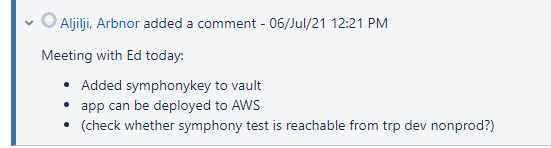
*Figure 4.1*

As its not practiced doing API calls in a testing framework, it is better to mock and test against that. Above is the screenshot of the mocking I did with my project lead.



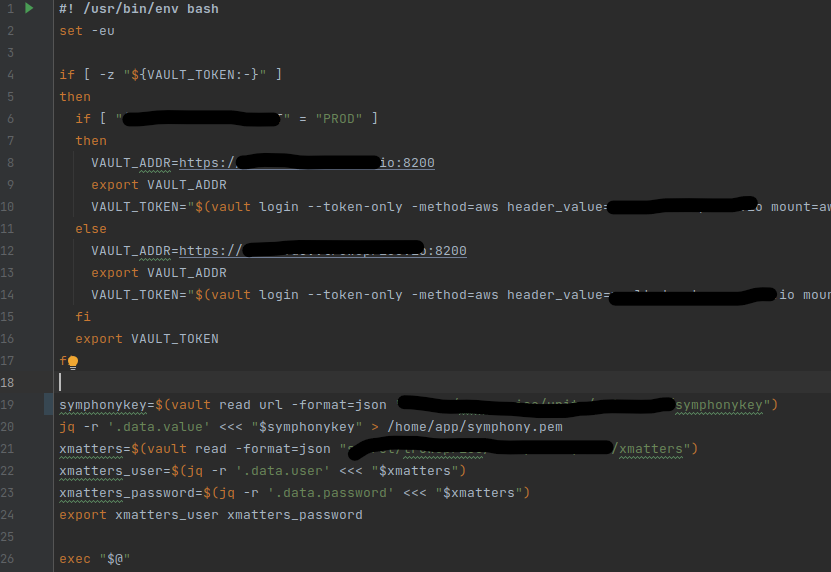
*Figure 4.2*

The following few weeks are the stages where I had to try and deploy the bot to AWS dev environment and add the secrets to vault.



*Figure 4.3*

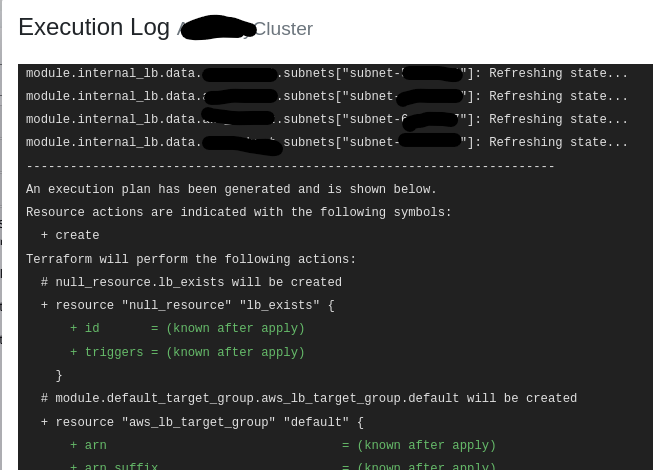
The next following few weeks I had to solve how to add my pemfile to vault, so now it can be used either in prod or dev environment. Reason being for safety reasons I don’t want my credentials stored anywhere in the app, only to be called from vault.



*Figure 4.4*

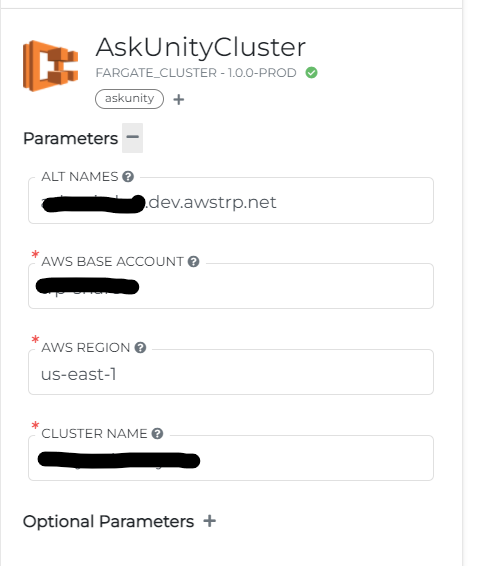
This is the entry point for the aws resource to run to initiate the app. Depending on what environment the app is in, prod, dev or local..it will use different configurations. Below will be posted the different files for each configuration environment. Due to confidentiality, I cannot show the config files for dev and prod. However, it is like figure 3.3 above.

The project lead and I set up meetings to use Unity our internal app to be able to deploy the bot to AWS. Due to the tech lead being experienced, they knew what resource I would need to be used for the app. During the early planning stages, there were discussions on what sort of resource I want for the bot to use. For example, whether the bot should be running 24/7 or whether it should work as a Lamba function where it listens only when it has been spoken too. As Terraform is wrapped around Unity, there is no need for me to do any terraform.

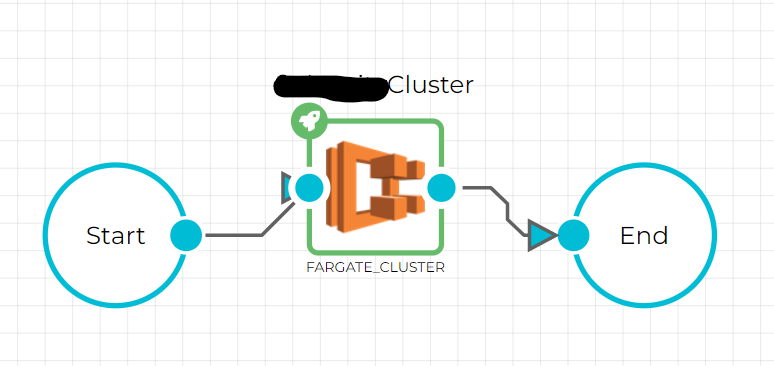


*Figure 4.5 Terraform wrapped within Unity*

Screenshot that shows Terraform being executed behind Unity Deploy when executing my strategy.

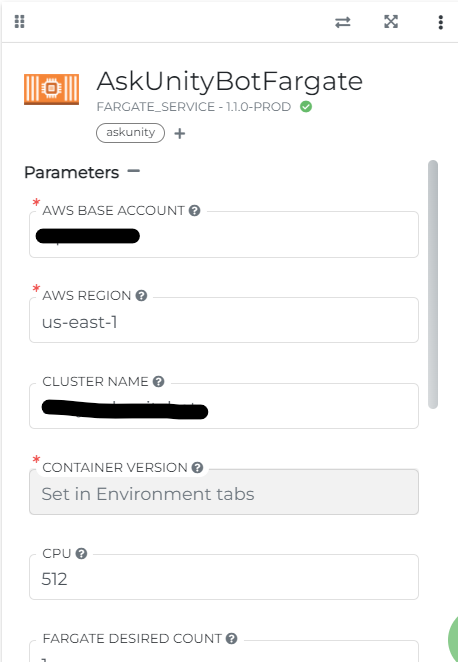


*Figure 4.6*

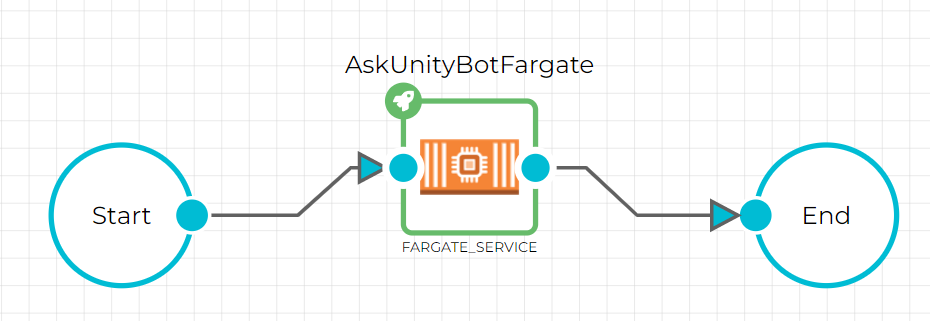


*Figure 4.7*

Above are 2 screenshots which I created in Unity which invokes the infrastructure for the bot in AWS. Once the infrastructure has been set in AWS, I would then need to invoke the resource strategy. Below will show the screenshots of my resource strategy.

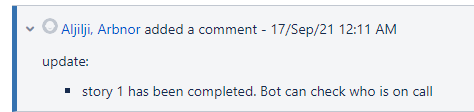


*Figure 4.8*



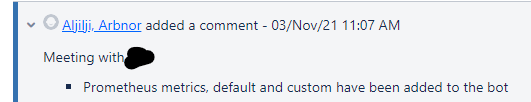
*Figure 4.9*

Above are the 2 screenshots of what I created to deploy the resources for the bot into AWS. Once the resources have successfully been uploaded to AWS, I would then go to the symphony test environment again and talk to the chatbot. To see that the service is working, I would be able to look at the logs on AWS. The next few weeks were to let the app run in the dev environment and test it daily that it’s working.



*Figure 5.0*

However, as the project was developing more in the dev environment and soon to be ready for prod, I needed to add Prometheus metrics.

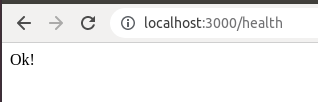


*Figure 5.1*

Due to my first-time implementing Prometheus to an app, I had to set up a meeting with my project lead. Firstly, I decided to use express module to create a simple health check as express is an easy solution for single page applications, websites, hybrids, or public HTTP APIs.

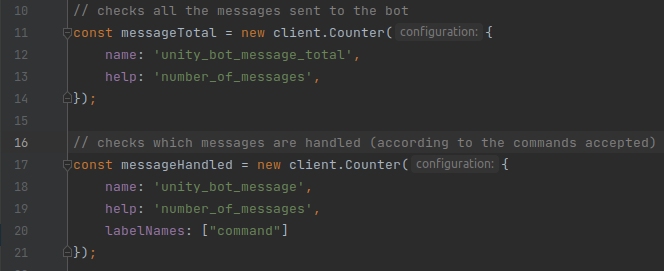


*Figure 5.2*



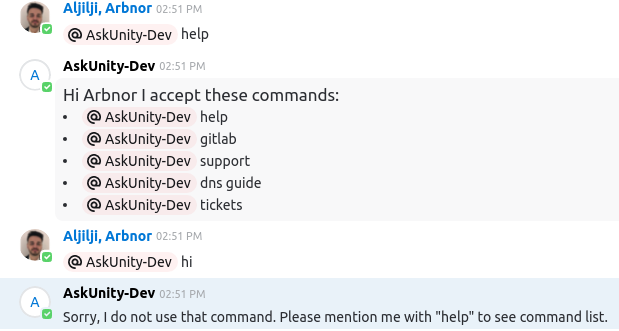
*Figure 5.3*

Next, I added the custom metrics on what should be captured. For the case of my bot, I implemented what messages (commands) are being sent to the bot, and which messages are being handled.

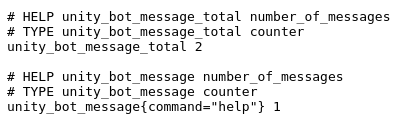


*Figure 5.4*

This code snippet highlights the custom metrics. To see that this worked, I accessed the localhost to see the metrics.



*Figure 5.5*



*Figure 5.6*

The final step to the project was to get it running in the prod environment. To change the project from dev to prod, I had to submit a change request using Unity Release Manager. However, when deploying to AWS prod, there is an issue with Symphony and AWS Prod which doesn’t listen to the bot. This ongoing blocker has been here for a while, and I’ve escalated it to higher management so it can be discussed with the CloudOps team who look after AWS and Symphony developers.

# Project Developer Desktop Self-Service Portal

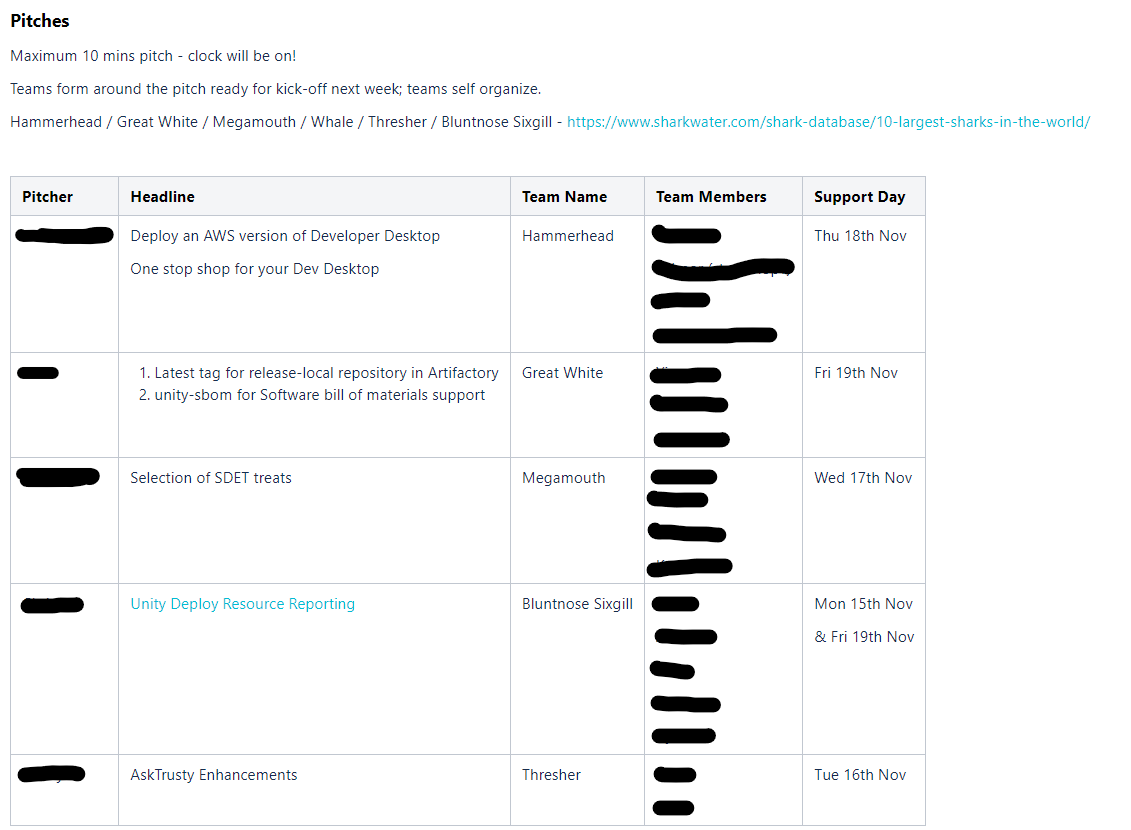
**KSBs acknowledged**: K2, K10, K20, S2, S4, S13, S16, S17, S21, B1, B3

Rescheduled my diary to fit their need, as I was the only UK based member. Shift pattern

**S2** - **Situation**: Working with US team, different time zone and working culture compared to UK team. **Task**: Work together to make the MVP of the project. Plan it and execute. **Action:** Rescheduled my diary to fit their need, as I was the only UK based member. Shift pattern changed and worked the way they did, doing sprints etc. **Result:** I was able to work with my team more often due to working hours meeting their needs. I managed to do majority of the pages requested and complete majority of my tasks within the time frame. It was presented and teams were happy with my progress.

K2:

Within TRP, I have opportunities to work with different teams on different projects. I had the choice of picking between several projects that caught my interest. The listed projects were put on confluence and the leaders of the projects were able to present their ideas briefly on confluence. – Describe the project here –

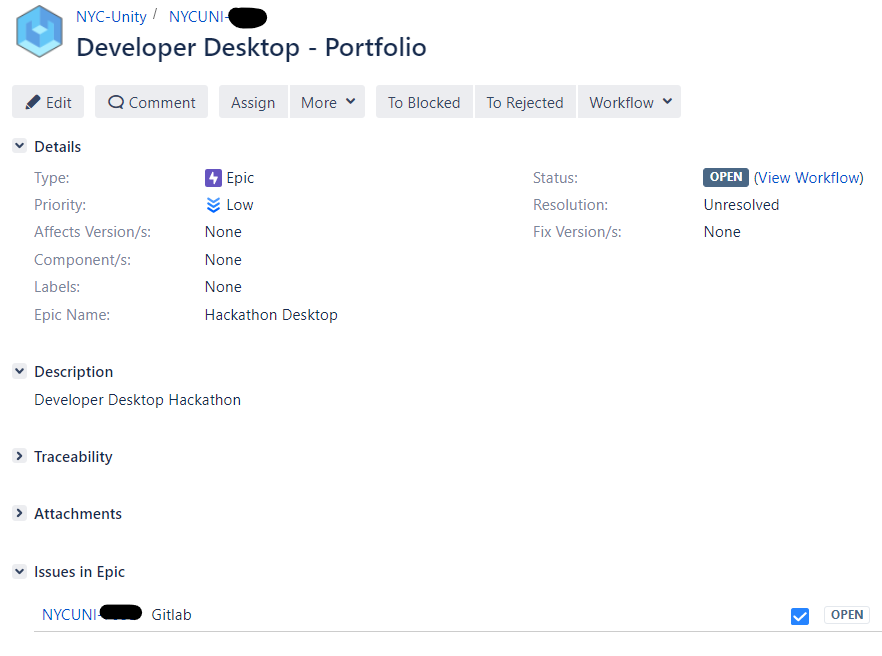


*Figure 1 – Project Overviews*

I attended the zoom meeting which the leaders were presenting the ideas and I added my name to the Team that I wanted to work in. I chose to work in the Hammerhead team as it was an interesting topic which would allow me to learn various new skills such as HTML coding.

**Time adaption (S2)**

(***S2***) The first thing I did was adapt to the changes of the team’s timing as they were in the US, so I decided to start working from 11am-8pm which meant 6am-3pm for the US team, which was enough time to work with the team, and they could then carry on with the remaining task throughout the day.



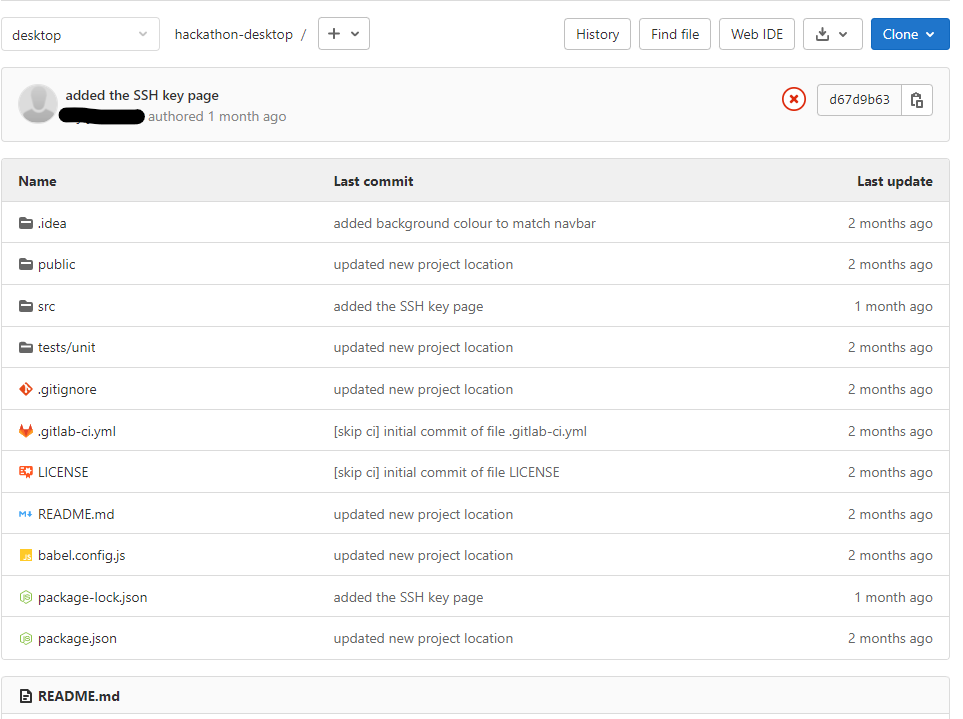
*Figure 1.1 - Jira ticket for Developer Desktop Project*

**Solo first day**

The following week of starting the project, I unfortunately had to work by myself on the first day due to the US team being on holiday on Monday. I took the initiative of setting up the project on gitlab so the team can clone the project the next day.

Above is the ticket for the project. As it was the first day of the project, I had to plan by myself how we will tackle the design of the architecture of the project. As I was not experienced in Front End development, I set up a meeting with an UX expert on what framework to use. The design of the project was that I use ReactJS as it will be a single page that renders (***S4***).

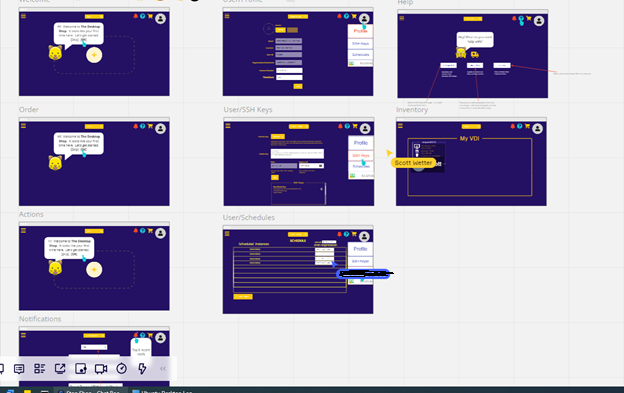
I planned the project of what framework we will use, what the project will look like (mocking), and what language to use. After the initial planning, I took the rest of the evening to learn HTML and CSS as it was my first time trying to do front-end developing. I took a quick course on codecademy and Udemy (***S16***) as it taught me the basics of front-end coding. This gave me an idea of how the elements in HTML work such as <div> etc and what framework to use (**React**). This initiative of taking lead in the project helped me set up the initial git lab repo so the team can clone it the next following day after the holiday.



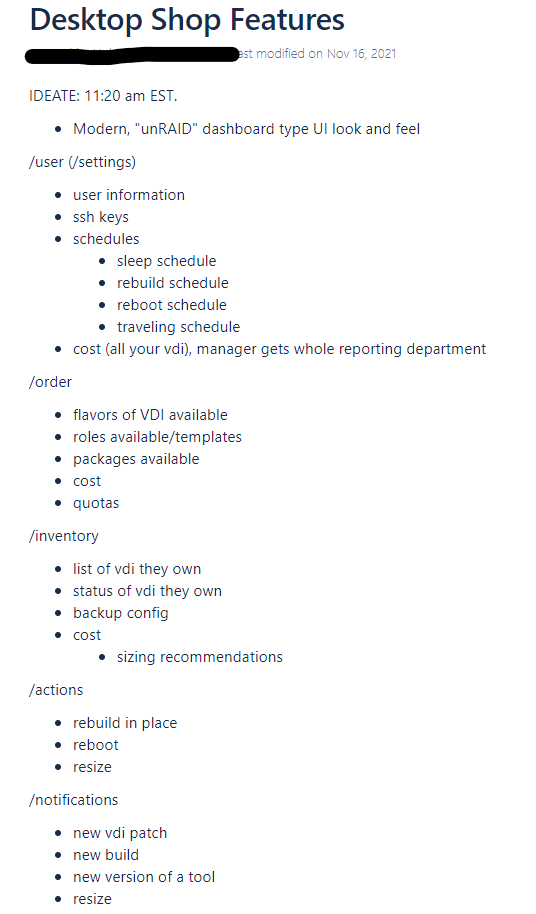
*Figure 1.2 Gitlab Repo (****K2****)*

Source control helps maintaining the project in one centralized place, where I can view the history of the project, and I could add a branch and work on my feature without disrupting my colleagues working on theirs. With small code changes, I can push for it to be reviewed and merged.

**Whiteboarding using Miro/confluence**



*Figure 1.3 Mocking up the single-page application using Miro*

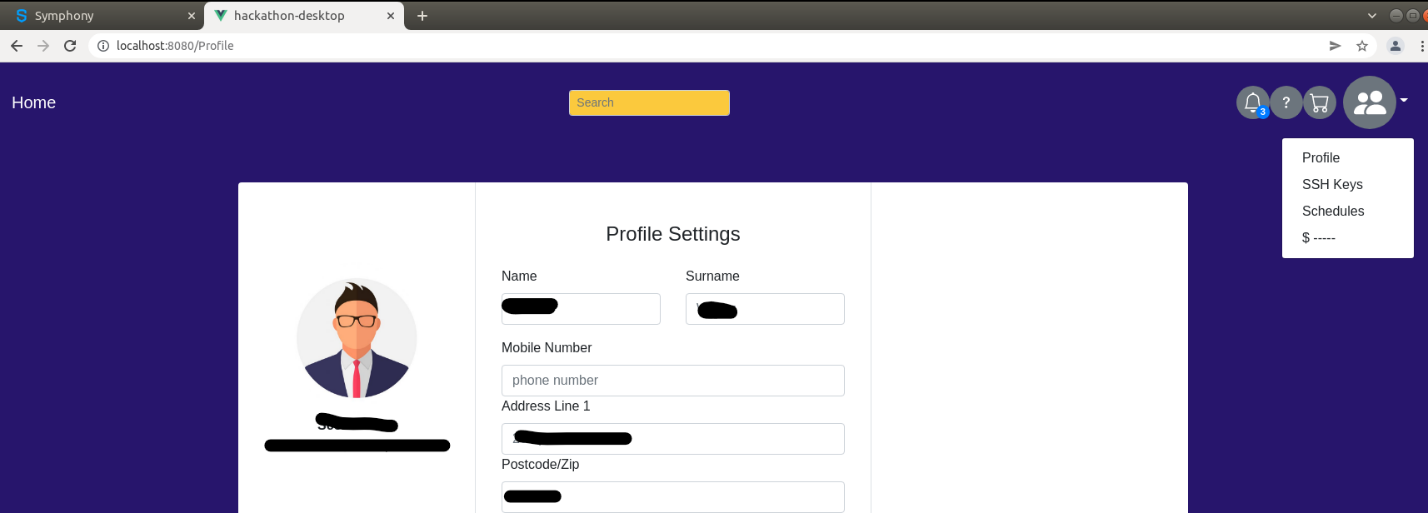


*Figure 1.4 - Project planning of the different routes and features of the project*

* Throughout the project planning on what the MVP of the product should look like and contain, the team and myself used Miro to mock-up the pages and whiteboard, and confluence for note taking (**S21**).
* Used client feedback to design the MVP and future features for the project (**K10**)

After the mock-up, it was time to code in JavaScript, HTML, and CSS (**S17**). I would sometimes pair up with a colleague and code together for the HTML/CSS (**K20**) as we were both learning, but we could also discuss the design of the page together while it’s changing. I really enjoy pair programming as we can take turn to drive while other researches, or figures out the issue (**B1**)

Below are the pages I designed using the mock-up, with slight adjustment:



*Figure 1.5 – Profile Page template*

Above is the profile page mock-up I coded using JavaScript, HTML, and CSS. The JS does the routing of the pages, while the HTML presents it and CSS styles it. As I was acting as project lead on this project, and specifically for this page, I was making sure I fixed whatever issue my code had raised. This led to continual improvement and seeing pages develop quicker, and team bonding more (**B3**). The success of this project gave us a small template to use, for the upcoming future I have with the project.

# Project Resource Enhancements

## Resource 1 (Unity-Elasticache)

**Intro to the project**

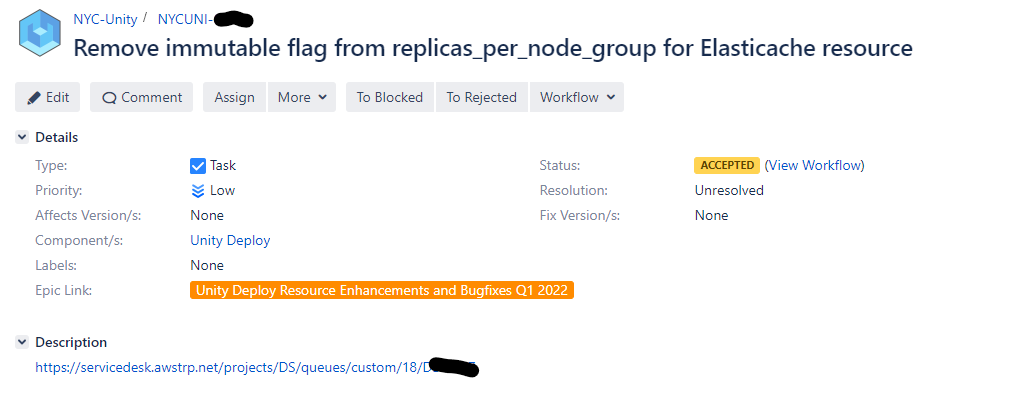
T Rowe Price use Unity Deploy as an internal app for managing the infrastructure in AWS accounts. Developer Services Group look after many AWS resources in the cloud so if there are any enhancements to the resources, the team and I will update it.

**How I got the ticket:**

Clients who use our services requests for an enhancement to one of the AWS resources we manage. I was messaged by a colleague within my team to look at potential resource changes to be made due to AWS updates, or client requests.

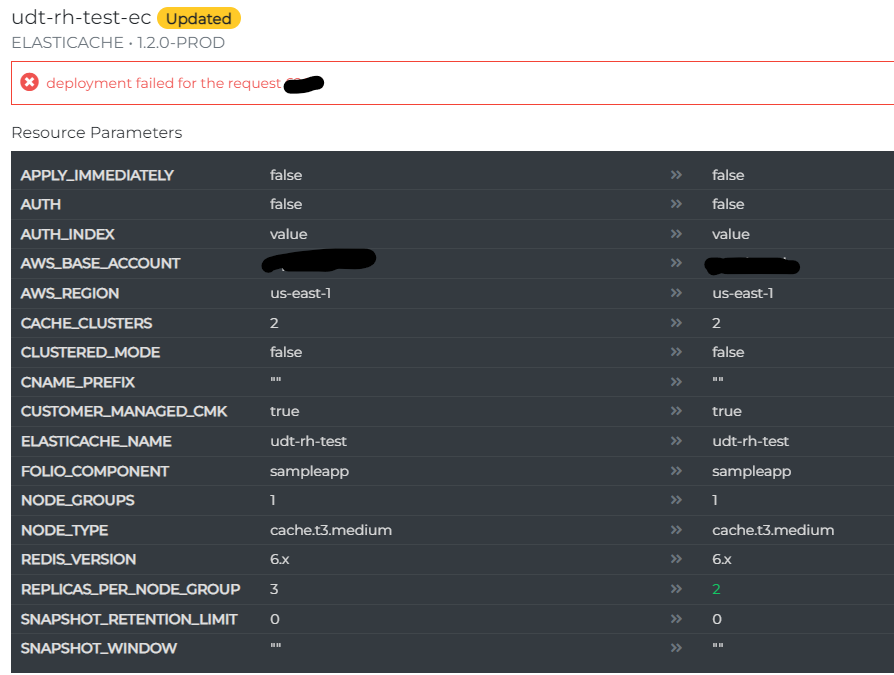
**Planning Phase**

To understand the request made, I set up a Zoom meeting with my lead colleague who assigned the ticket to me. I went through the discussion of the ticket, to ensure I understood what is being suggested and where I can find the resource for changes to be made.



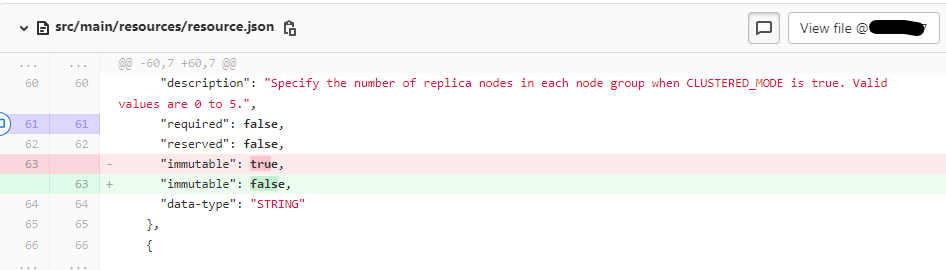
**Implementation Phase**

To get started with the code change, I had to test that the immutable flag that was already set to ‘true’ meant that you couldn’t redeploy the resource without needing to destroy it first. The client wants to redeploy without needing to destroy it.

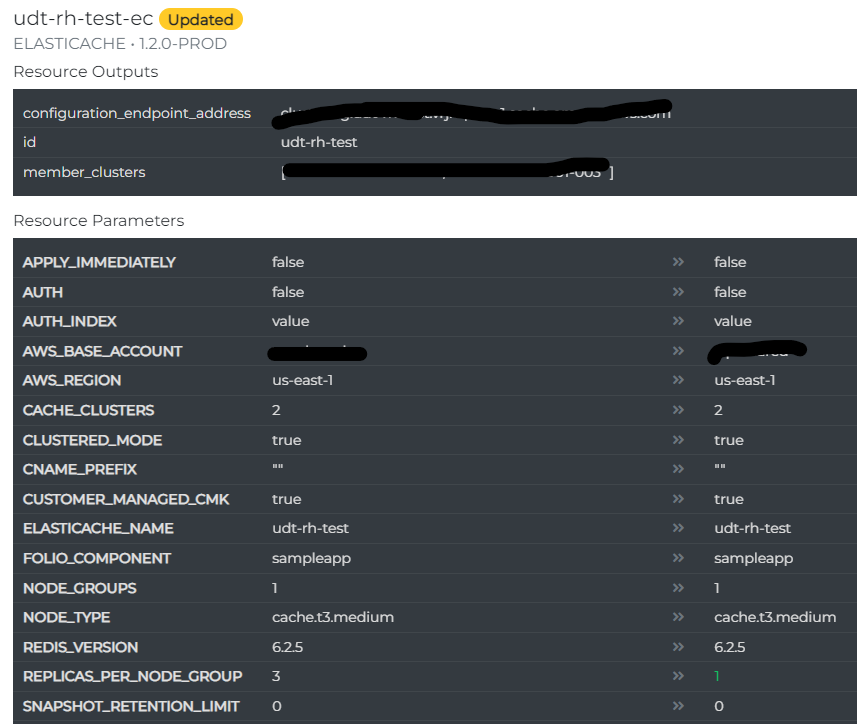


This threw an error above with deployment. I made then the code change as seen below to allow the number of replica nodes to be changed.

The snippet below shows the before and after of the code change:



After pushing the code change, there’s a build stage where it syncs the latest code to the dev testing environment. I was then able to test in the Unity Deploy test environment to see whether the changes had worked, and once I saw that it works, I would be able to create the merge request and create a release.



**Value**: The benefit this gives the client was that by allowing them to change the number of replica nodes meant that they didn’t have to spend time destroying their resource when they can just redeploy it, as some tasks can take minutes to run.

## Resource 2 (OpenSearch-Domain)

**Intro to the project**

Similar introduction to Resource 1. This resource was meant to get migrated

**How I got the ticket:**

Like Resource 1, I was given this ticket from my colleague who gave me more resource changes to make as AWS now allows you to change field values without needing to destroy the TRP cluster.

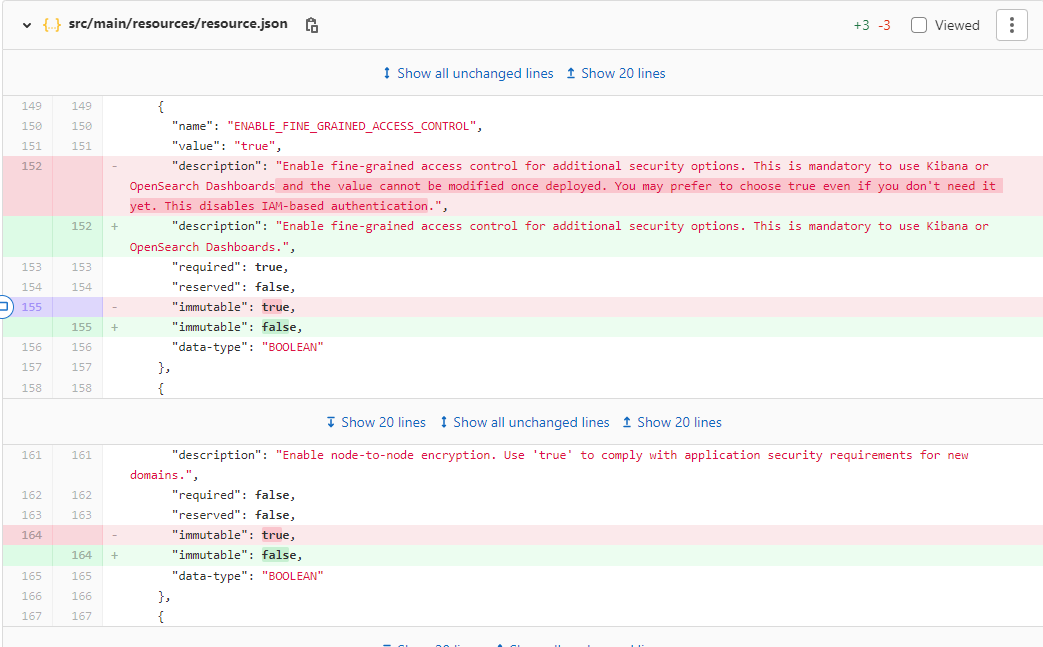
**Planning Phase:**

After receiving this ticket from my colleague, I planned out what my next task will be. I didn’t really have to use any whiteboarding techniques or mapping as the task itself was straight forward. I knew I would have to test that I can change the field values without destroying the cluster as that’s what the client had reported. To confirm this, I didn’t step up a zoom meeting to clarify this as I was confident in what the requirement was, but I did message the lead to confirm I understood the requirement. For this case, this method of communication was best suited. The next step was to find the resource that will get the enhancement made to it and make the change.

**Implementation Phase:**

The next step was to find the resource that will get the enhancement made to it and make the change. From understanding the ticket, I noticed it was these fields that needed to be changed, and to clarify it, I messaged my lead on Symphony to confirm to which it was. Firstly, I tested the old resource ES\_DOMAIN 1.1.0 with all the security settings turned off and tried to redeploy the OPENSEARCH 2.0.0 by using the migration wizard. However, without having done any code changes the deployment didn’t work. I then made the changes.

Below is a snippet of the code change.



After pushing these changes, the sync updated the testing environment which I was able to then go and test the old resource ES\_DOMAIN 1.1.0 with all the security settings turned off, then upgrade using a built-in migration wizard to OPENSEARCH 2.0.0, turn on the security settings and redeploy.

This led to a successful redeployment of the resource OPENSEARCH 2.0.0.

**Value:** With this change, it allowed the resource to be redeployed without needing to destroy and recreate the cluster, losing your data or creating downtime for the app while you copy the data from backup.