MAKERS ACADEMY, PORTFOLIO SUBMISSION

No. 3

Placement: Compare the Market
Apprentice: Krzysztof Kasprzak

Table of Contents

Dockerising the tests task 2
The task 2
Project, tools, and languages I worked with 2
The ticket, it's requirements and how I got involved 2
What is Docker and why is it in use 3
What is Docker 3
Why is it in use 3
Our CI/CD environment 3
Steps to complete the task 3
Creating docker files 3
Writing test script files 4
Updating README.md 5
In conclusion 6
Things I learned and improved upon during the work 6
Things I would do differently next time 6
Value I brought and impact I made 7

Dockerising the tests task

The task

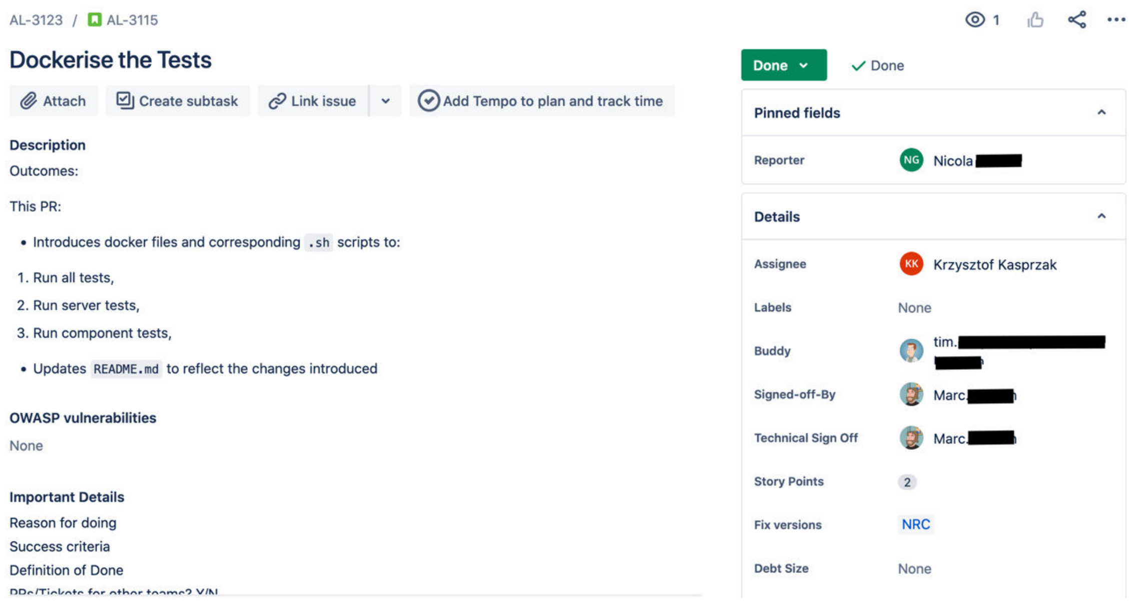
Project, tools, and languages I worked with

In this portfolio piece, I will be describing how I have used Docker (a tool for containerising applications), to allow for the Allspark UI rewrite project tests to be run on the CI/CD (continuous integration/continuous development) pipelines. For that, I have worked with and used:

- Docker Engine, software that allows for containerisation of applications,
- Docker dashboard, a UI app that allows for easier management of dockerised (apps containerised by docker),
- .md files to create documentation about the changes I introduced,
- GO Language (dockerfiles are written in it),
- Bourne-Again shell, language used to write scripts and script files on Unix-style operating systems (OS's that I use at work, MacOS which I use for local development and Linux, OS ran on project's pipelines both are Unix-style).

The ticket, it's requirements and how I got involved

The ticket's number was AL-3115 and it was one of the first tickets created when efforts to create the Allspark UI rewrite begun. The ticket itself was not detailed; it only included its title (as seen on figure 1.1 below). It was because the title itself was descriptive enough to indicate what had to be done. Only upon the completion, I have added a bare-bone description of what this ticket achieved. I've done so to let the ticket better document what was created for this task.



AL-3123 / AL-3115

Dockerise the Tests

Attach | Create subtask | Link issue | Add Tempo to plan and track time

Description

Outcomes:

This PR:

- Introduces docker files and corresponding `.sh` scripts to:

1. Run all tests,
2. Run server tests,
3. Run component tests,

- Updates `README.md` to reflect the changes introduced

OWASP vulnerabilities

None

Important Details

Reason for doing

Success criteria

Definition of Done

Done ✓ Done

Pinned fields

Reporter	Nicola [redacted]

Details

Assignee	Krzysztof Kasprzak
Labels	None
Buddy	tim.[redacted]
Signed-off-By	Marc[redacted]
Technical Sign Off	Marc[redacted]
Story Points	2
Fix versions	NRC
Debt Size	None

Fig. 1.1, a screenshot of the Jira ticket I completed in order to dockerised the project's tests.

I have worked on the ticket in a pair together with Tim, with whom I already paired with frequently. Despite not having a great deal of knowledge about dockerising apps or programs (prior to the task, I've only ever used written scripts to run apps in containers), I approached the work, as Tim and I are two main developers involved in the project. To get myself ready for the task, I have followed a docker tutorial on the Docker Inc. website and studied how Allspark's other projects were dockerised.

What is Docker and why is it in use

What is Docker

Docker is a set of PaaS (platform as a service) products that allows developers to containerise applications and programs. In essence, containerising is a process of running an app or a program by an isolated, virtual machine: a sub-part of a physical computer that docker dedicates to behave like a separate machine.

Why is it in use

Use of containerised applications is Compare the Market's technology incentive; this way the company is ensuring that its resources (both internally and externally facing) are kept as micro services. The idea of micro services is to keep any software that the company uses compartmented in separate, easy to use, maintain and develop projects. This way company can be more Agile in its ways of software development.
Docker is the company's technology of choice for containerising its services.

Dockerisation allows for all of the company's applications to be developed and shipped in the same conditions across environments. That is, application that is being developed and tested in its dockerised form, is then easy to push down the pipeline to a usable server, since being containerised allows it to run on a server with minimal to no prior set up.

Our CI/CD environment

The continuous integration and continuous development set up for the project is rather simple. There are three environments to consider:

- Local: it is used to develop the application and run it on developers' own machines,
- Shadow: it is a server hosted environment which uses a mocked data (sourced from some previous versions of production environment) to test that the app behaves a sit should in the live environment,
- Production: it is a server hosted environment which will be used by the user base, and it connects to the production-data facing back-end.

The last two environments are being continually integrated with new, developed features via GO pipelines which source code version of from GitHub.

Steps to complete the task

Creating docker files

Firstly, to dockerise the app's tests, I've created two docker files (I did it while pairing with Tim, I was driving, and he was navigating): Dockerfile-test and Dockerfile-test-server.
Both of which were simple, extremely similar text files that used Go Language: statically typed, compiled language (see the fig 1.2 below for the code I've written for Dockerfile-test).

```
FROM node:erbium-alpine3.12

RUN install -d -m 0755 -o nobody -g nobody /.npm

USER nobody

COPY --chown=nobody:nobody . /app

WORKDIR /app


CMD ["npm", "test", "--", "--watchAll=false"]
```

Fig. 1.2, code I've written to create Dockerfile-test.

As seen on the previous page, this file outlines what must be done by docker to containerise tests for the project. On the first line, I defined which docker image has to be used; in this case it was alpine, a minimal yet complete Linux image to be ran on the virtual machine in the container. Docker images are standalone packages of software which can run applications. Alpine, for its lightweight is a company advised docker image. Next, using `RUN` and `COPY` commands, I made sure to install the alpine package as a `nobody` user (with the GO command `USER`). This means that in the container, the package can be accessed by any application. It also is a best practice, since `nobody` does not specify users and cannot allow for infiltration of containers by exploiting other users' privileges. Furthermore, the use of `COPY` GO command is also regarded as best practice. This command, unlike `ADD` is more transparent and cannot be spoiled to `ADD` malicious images, `COPY` as the name suggests, only copies the installed image.
Then, I defined from which working directory should the tests be run: `/app` (by using `WORKDIR` command). Lastly, to run the tests, I've used `CMD` (command) command to execute `npm test – watchAll=false` command within the virtual machine environment. The appended ` watchAll=false` was there to run the npm tests (which uses Jest testing library) outside the watch mode. Locally, jest watch mode allows developers to run the unit tests in real time as those are being updated, i.e. it doesn't terminate on its own.

Writing test script files

Next step to dockerise the tests, was to create bash (acronym of Bourne-Again Shell) script files to invoke dockerfiles' logic and run operations on the created containers.
There were four `.sh` script files created:

1. test-build.sh (Tim was driving and I was navigating): as the name suggests, the file directly invokes dockerfiles to build up images.
2. test-run.sh (Tim was navigating and I was driving): runs a script to conduct app's unit tests.
3. test-server-run.sh (Tim was navigating and I was driving): runs a script to conduct app's tests on its server connection readiness and correctness.
4. test-all.sh (Tim was navigating and I was driving): runs a script to conduct both, unit and server tests.

```
#!/bin/bash
err_report() {
    echo "Could not build tests : Error on line $1"
}
trap 'err_report $LINENO' ERR
docker build -f Dockerfile-test -t rtl-tests .
docker build -f Dockerfile-test-server -t rtl-test-server .
```

Fig. 1.3, code included in test-build.sh.

As seen on the fig. 1.3 above, the test-build.sh invokes the dockerfiles to build images; it names them `rtl-tests` and `rtl-test-server` respectively (two bottom code lines). While driving on this code, Tim has shown me this pattern of `error report()` and `trap`. Bash commands that used together, check for any errors in executed operations and act on them. In this case, it was desired that upon any errors on building images, it would be printed to the terminal that tests could not be built (third line of the code snippet).

I've used this newly met pattern to write test-run.sh and test-server-run.sh files. Both of which were very similar (see fig 1.4 below for the code I've written to create test-server-run.sh).

```
#!/bin/bash
err_report() {
    echo "Could not run test server : Error on line $1"
}
trap 'err_report $LINENO' ERR
if [  "$(docker ps -aqf "name=test-server-container")" ]; then
    docker container rm $(docker ps -aqf "name=test-server-container")
fi
   docker run --name test-server-container rtl-test-server
```

Fig. 1.4, code snippet of test-server-run.sh file contents.

As seen on the figure 1.4 on the previous page, I've not only used `error report()` and `trap` to act and `echo` the message to the terminal on unsuccessful script termination. I have also used a simple `if fi` bash statement. `if fi` statement in bash is just a way of writing simple if statements.
This statement encloses its condition between `if` and `; then`, and contains any logic-dependant actions between `; then` and `fi`.
In this case, it makes sure to remove (`rm`) a given docker container if this container was found in the list of existing ones (`docker ps -aqf "name={defined container name}")`). After the if statement, the script creates a new, up to date container.
Carrying out the pre-run deletion of any existing image in a container was required as otherwise, the script would never run up-to-date image, rather its first ever created version.

Lastly, I've created a simple test-all.sh script which simply runs both types of tests with one command. This was added mainly for the convenience of local development, to allow developers to run both types of tests in the locally dockerised container with one, simple command (see the fig. 1.5 below for its code).

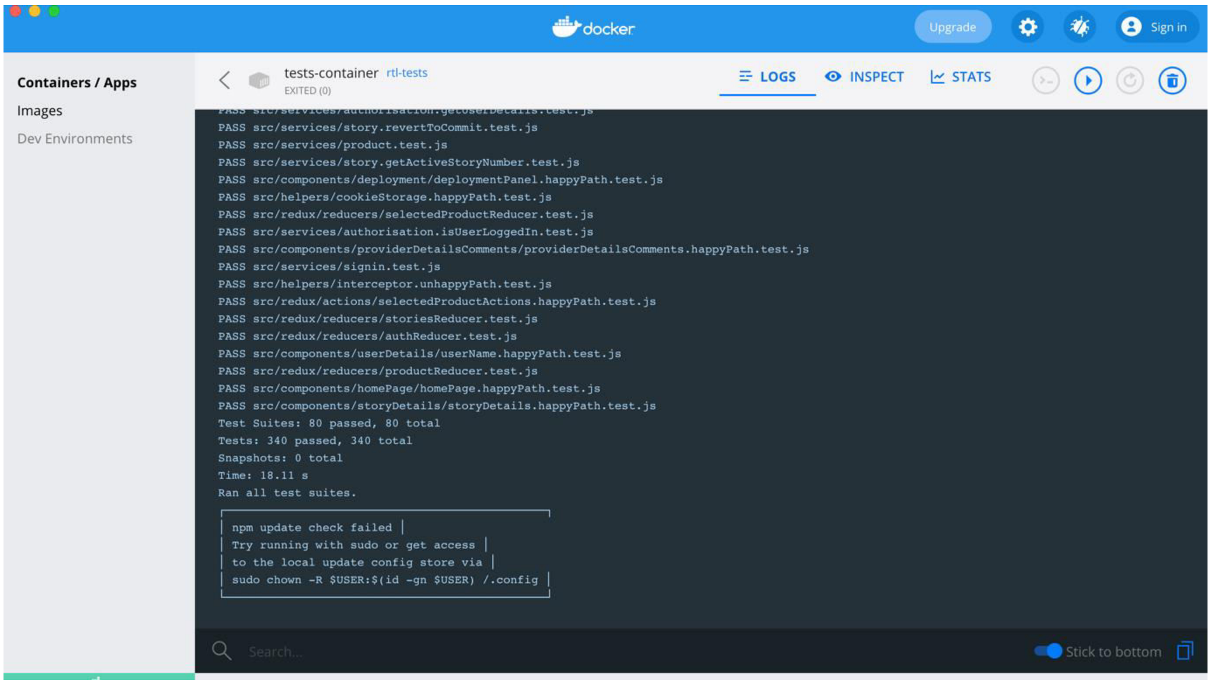
```
#!/bin/bash

./test-server-run.sh
./tests-run.sh
```

Fig. 1.5, simple code written to run both tests via a single script.

The only thing to address with test-all.sh is how it uses `#!/bin/bash` as its first line of the code, same as any other `.sh` file described here. It serves the purpose of finding the correct script shell to execute the commands in the file, i.e. Bourne-Again shell.

Updating README.md

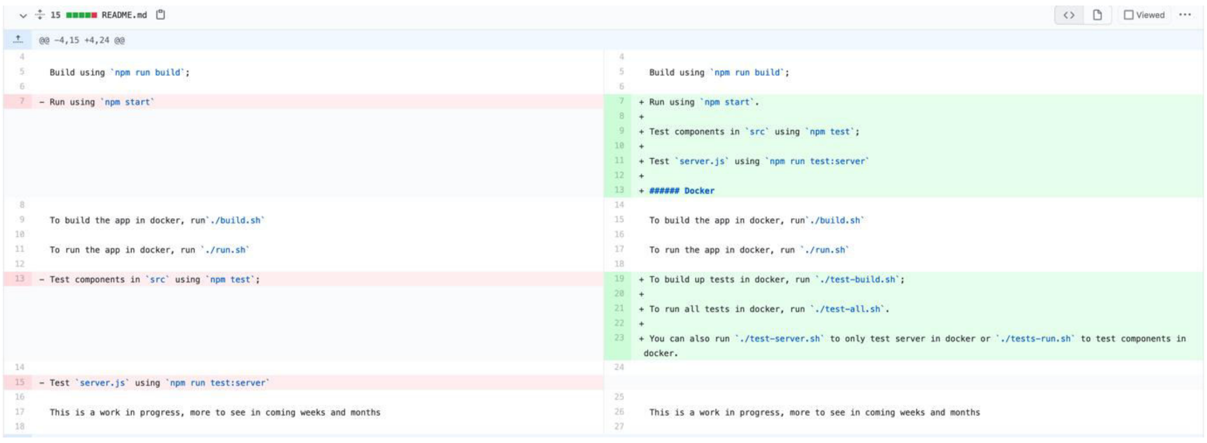
After checking that the scripts are running correctly using the Docker Dashboard (see fig 1.6 below), the only thing left to do, was to document the new testing scripts and how to run them for the benefit of the future development and accessibility for other developers.



```
PASS src/services/authorisation.getUserDetails.test.js
PASS src/services/story.revertToCommit.test.js
PASS src/services/product.test.js
PASS src/services/story.getActiveStoryNumber.test.js
PASS src/components/deployment/deploymentPanel.happyPath.test.js
PASS src/helpers/cookieStorage.happyPath.test.js
PASS src/redux/reducers/selectedProductReducer.test.js
PASS src/services/authorisation.isUserLoggedIn.test.js
PASS src/components/providerDetailsComments/providerDetailsComments.happyPath.test.js
PASS src/services/signin.test.js
PASS src/helpers/interceptor.unhappyPath.test.js
PASS src/redux/actions/selectedProductActions.happyPath.test.js
PASS src/redux/reducers/storiesReducer.test.js
PASS src/redux/reducers/authReducer.test.js
PASS src/components/userDetails/userName.happyPath.test.js
PASS src/redux/reducers/productReducer.test.js
PASS src/components/homePage/homePage.happyPath.test.js
PASS src/components/storyDetails/storyDetails.happyPath.test.js
Test Suites: 80 passed, 80 total
Tests: 340 passed, 340 total
Snapshots: 0 total
Time: 18.11 s
Ran all test suites.

 ┌─────────────────────────────────────────────┐
 │ npm update check failed │
 │ Try running with sudo or get access │
 │ to the local update config store via │
 │ sudo chown -R $USER:$(id -gn $USER) /.config │
 └─────────────────────────────────────────────┘
```

Fig. 1.6, a screenshot of Docker Dashboard after it captured a successful execution of `./tests-run.sh`. As can be seen on the image, it allows to interact with a containerised app and images, allowing for richer and easier experience than the default usage of docker from a command terminal.

I did so by updating the README.md file on GitHub for the project (see the fig. 1.7 below for the changes I made to the file).



```
15 README.md
@@ -4,15 +4,24 @@
```

```

  Build using `npm run build`;

- Run using `npm start`

  To build the app in docker, run`./build.sh`

  To run the app in docker, run `./run.sh`

- Test components in `src` using `npm test`;

- Test `server.js` using `npm run test:server`

  This is a work in progress, more to see in coming weeks and months

```

```

  Build using `npm run build`;

+ Run using `npm start`.
+
+ Test components in `src` using `npm test`;
+
+ Test `server.js` using `npm run test:server`
+
+ ###### Docker

  To build the app in docker, run`./build.sh`

  To run the app in docker, run `./run.sh`

+ To build up tests in docker, run `./test-build.sh`;
+
+ To run all tests in docker, run `./test-all.sh`.
+
+ You can also run `./test-server.sh` to only test server in docker or `./tests-run.sh` to test components in
  docker.

  This is a work in progress, more to see in coming weeks and months

```

Fig. 1.7, GitHub log of changes I made to the README.md file to reflect the new scripts for building and running dockerised tests.

As seen on fig 1.7 above, I opted in for a simple, step-by-step guide on how to run the scripts to simplify the user experience (i.e. developers who will use the scripts locally).

In conclusion

Things I learned and improved upon during the work

While working on the task, I have:
- learned more about docker and the basics of GO language syntax used with it,
- how to write dockerfiles and bash script files,
- practiced writing .md files,
- walked through a process of containerising apps and programs,
- learned why containerisation helps with deployment and application migration to different environments,
- improved my familiarity with Docker Dashboard tool.

Things I would do differently next time

There isn’t much I would change in how I would approach a similar task in the future. I would rather expand on things I have done for this ticket on conducting a similar task by:
- Considering any environment variables that would potentially differ the outcomes of tests,
- Potentially combine build and test scripts into one (which, however, would be up to preferences of developing team).

In fact, as of the time of writing the ticket, after discussing these with Jose, teams in-test engineer, change have been made to accommodate for different environmental variables. Th project is also set to have a script that builds and runs test all together.

Value I brought and impact I made

Due to my work on this ticket, I have:

- Brought in better, more strict development cycle. By containerising tests those were able to be run in pipelines (processes migration), for example after each PR, this has helped to increase the quality assurance for the project.
- Allowed for more realistic testing to take place locally; containerising them allowed for more predictable results as the testing environment became identical as the one in live environments and in pipeline checks.
- Developed appropriate documentation of new process for other developers. With that, I have contributed towards a better training for the scripts’ users (future developers who will need to familiarise themselves with the codebase and its features).