End Report for the Final Project

Instructions for the teaching assistant

This is somewhat of a “work-in-progress submission”. Features were designed with the project requirements in mind, however, due to various factors implementation ran into issues and in the end could not be fully finished.

This report aims to guide the reviewer trough what was implemented and, in the end, briefly analyse what did not work.

NOTE! The exact API spec is not quite followed, see Reflections.

Implemented optional features

None

Instructions for examiner to test the system.

Relevant hardware

- CPU AMD Ryzen 7 5800X3D
- Corsair 1TB Nvme storage
- 32GB DDR4

Platform Info

The system was developed on an x86 system running Ubuntu 24.04, docker version 27.5.1, build 9f9e495 was used during development.

NOTE! The usage of “docker-compose” lead to issues during development, installing the latest version using the package manager and using it resulted in a crash caused by some dependency errors within docker-compose itself.

Following some discussion on this GitHub thread https://github.com/actions/runner-images/issues/9864 “docker-compose” was substituted with “docker compose” without the dash. “docker-compose” may work on the test system, but only “docker compose” was verified.

Aku Suvanto

Description of the CI/CD pipeline

The project continued to utilize git for version management. For this project, due to it being a single developer project, the use of “main” as the development branch was chosen. Once complete, the changes would then be submitted to the “project” branch for review.

The original program was copied from the public GitHub repository:

https://github.com/akusuvanto/devops-docker-compose

And then changes were implemented in the private Gitlab repository:

https://compse140.devops-gitlab.rd.tuni.fi/akusuvanto/finalproject

This repository is assumed to be available for the course personnel (otherwise, work has been mirrored to the public repository as well)

The project runs an automated Gitlab CI/CD pipeline each time code is pushed to the repository. This includes building the application, testing it, and deploying it to the machine the Gitlab CI Runner is installed.

NOTE! This project uses the “shell” type runner for CI, for further development it would likely be better to switch to “docker” runners and use docker within docker to achieve these tasks for better security.

Build

The project builds the containers each time new code is submitted to the repository, but it can also be manually built by cloning the repository and running docker compose.

$ git clone -b project https://github.com/akusuvanto/devops-docker-compose

$ cd devops-docker-compose/

$ docker compose build –-no-cache

Testing

During the project, a test-driven development approach was used. Each implemented function had tests written for it beforehand. The test cases were rather simple to keep the project manageable. Most endpoints are tested for a proper header return type or status code and some test if the system state changes properly or if the return.

The tests were implemented using custom scripts for node.js, written in JavaScript. The tests utilize the Axios library for doing web requests.

If the system has been manually deployed using the above instructions and is ready to be tested, you can run these tests manually by executing the following commands (this requires node.js and npm to be installed on the test system):

$ cd tests/

$ npm install

$ node test.mjs

Deployment

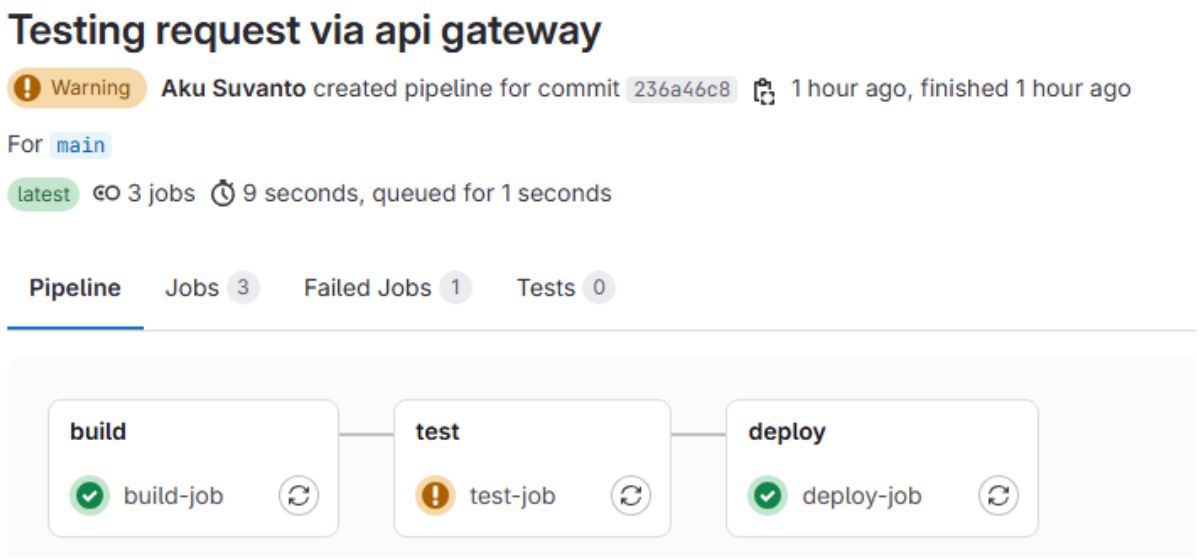
The pipeline automatically deploys the application on the machine the Gitlab Runner is installed, in this case, the development machine. In this way it is easy to test functionality after code is submitted to version control.

After following the build instructions, in the root directory of the project the following command can be run to deploy locally:

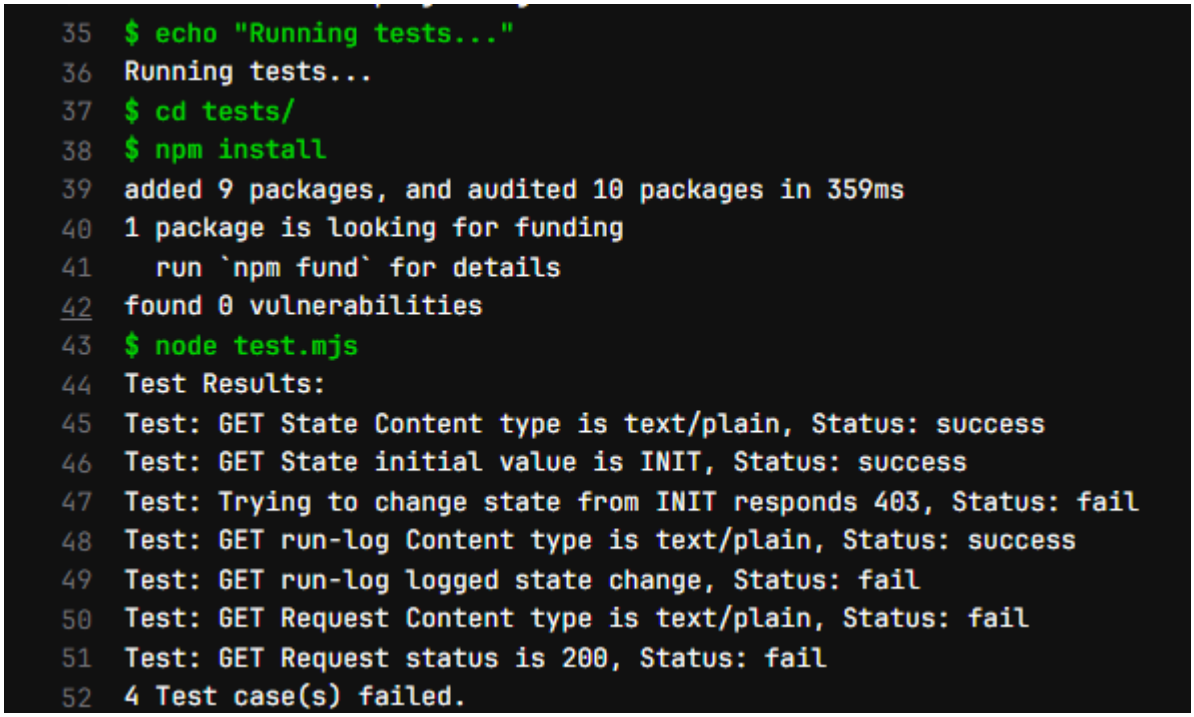
$ docker compose up -d

Example runs of the pipeline

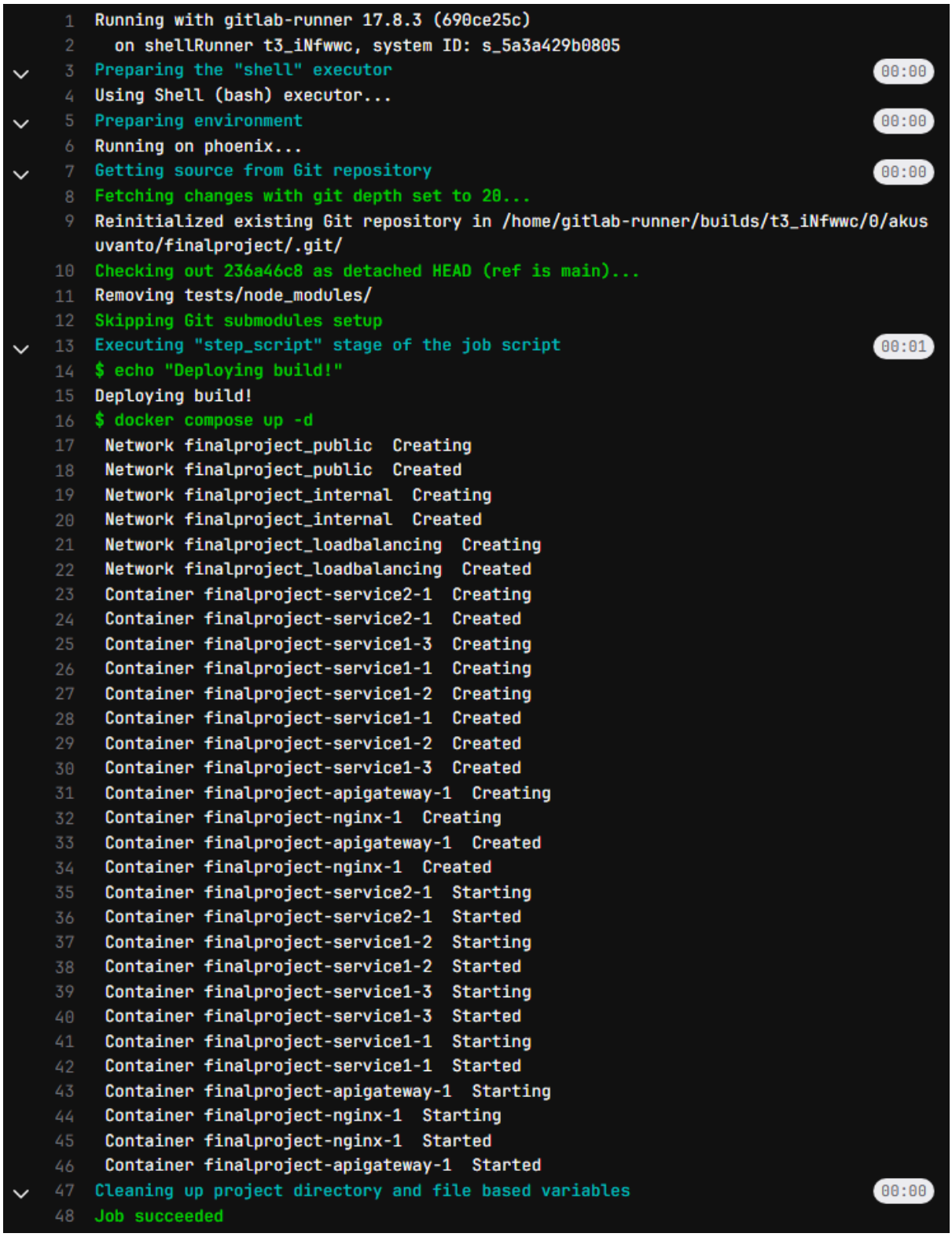
Here is an example screenshot of the pipeline running in Gitlab



Clicking on the text-job, we can see the pipeline is failing due to 4 test cases not passing, however the pipeline has been configured to proceed with a warning (easier for development purposes). Full log in Appendix 1.


$ echo "Running tests..."
Running tests...
$ cd tests/
$ npm install
added 9 packages, and audited 10 packages in 359ms
1 package is looking for funding
run `npm fund` for details
found 0 vulnerabilities
$ node test.mjs
Test Results:
Test: GET State Content type is text/plain, Status: success
Test: GET State initial value is INIT, Status: success
Test: Trying to change state from INIT responds 403, Status: fail
Test: GET run-log Content type is text/plain, Status: success
Test: GET run-log logged state change, Status: fail
Test: GET Request Content type is text/plain, Status: fail
Test: GET Request status is 200, Status: fail
4 Test case(s) failed.

We can also see that the deploy-job completed without errors, following is the full screenshot of that log.



Reflections

Main learnings and worst difficulties

Regrettably a lot of issues could have of course been solved with better time management, but let's focus on the technical issues faced.

A lot of time was spent thinking about how the project should handle the communication between the API gateway and the previously created services. It was chosen that for best results the API gateway should send requests trough the Nginx load balancer. But an issue that remains unfixed was encountered where the nginx configuration would not allow requests from the gateway application. We suspect this is due to doing authentication wrong on the API gateway side, but the issue was not able to be resolved in time. As such the authentication on the gateway side is also insecure, as is the main auth (but that is more of a project definition issue.)

Another issue (unclear if this is technically an issue as the spec is kind of vague on the details) is that the state change only works when sending the status as json due to some parsing issues.

Important! To test state change, you can send the PUT Requests to the /state endpoint as a raw body containing for example "{"status":"PAUSED"}" without the outer quotation marks in Postman.

One more major issue that took a lot of hours was trying to get communication between the API Gateway and the Nginx proxy working. Since the state of the application needed to initially be changed on a successful login, it was decided that nginx should send a PUT request to the API Gateway when that is detected. After researching how this could be achieved, we determined we should use the Lua plugin for nginx. However, we were never able to get the http package for Lua to install properly inside the docker container. We ran into multiple version conflicts and missing dependencies and in the end some other errors we were never quite sure how to solve.

We think the approach to completing the project tasks was sufficient and out design was not unreasonable, and that all of these issues could have been solved with the approach we had but, in the end, we ran out of time and the project will need to be submitted as is.

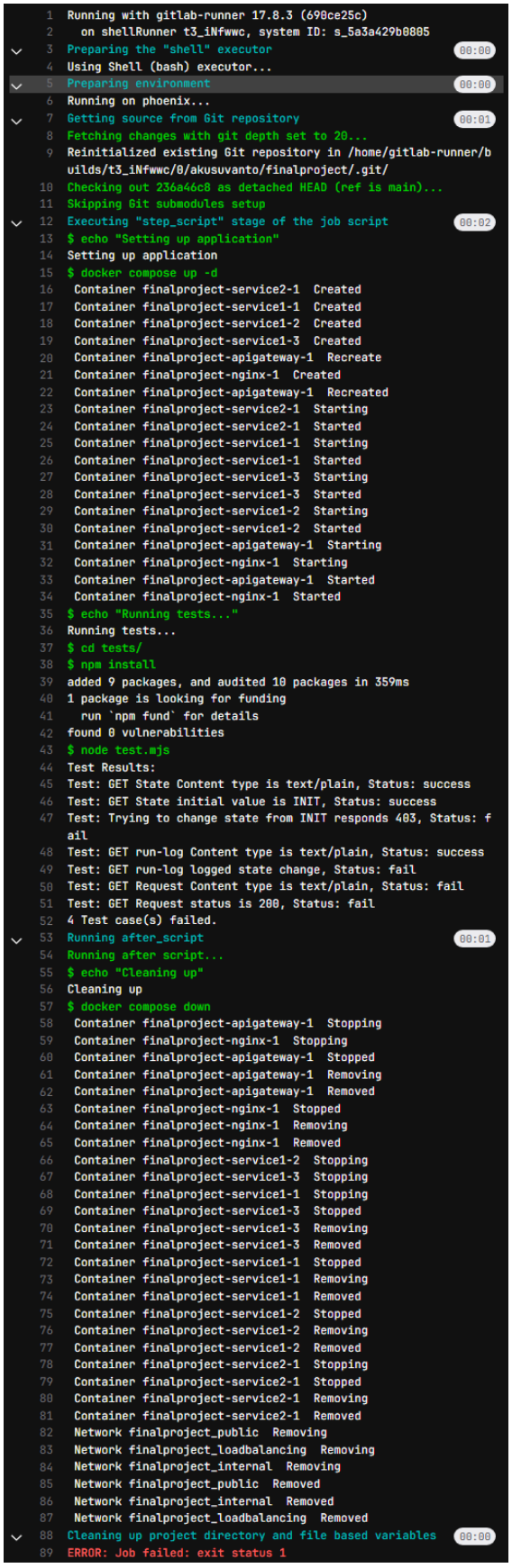
What was learned: CI/CD is very useful, and automation of processes leads to easier development. Personal learnings toward better time management.

Worst difficulties: Building more advanced docker images (including the Lua plugin in the nginx image and then building the http module along with all it's dependencies)

Amount effort (hours) used

The work of the project was spread over five days, with development of the application landing on the final three. Two first days were spent researching and planning approaches as well as familiarization with the technologies used. In total the time in hours should be about 20 to 25, most of it burned on debugging and environment issues. Should the full estimate (50 hours) the project description gave, I believe the project could have been completed fully, possibly with some of the additional features.

Appendix 1 (failed test-job)



```
Running with gitlab-runner 17.8.3 (698ce25c)
  on shellRunner t3_iNfwwc, system ID: s_5a3a429b0805
Preparing the "shell" executor 00:00
Using Shell (bash) executor...
Preparing environment 00:00
Running on phoenix...
Getting source from Git repository 00:01
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /home/gitlab-runner/builds/t3_iNfwwc/0/akusuvanto/finalproject/.git/
Checking out 236a46c8 as detached HEAD (ref is main)...
Skipping Git submodules setup
Executing "step_script" stage of the job script 00:02
$ echo "Setting up application"
Setting up application
$ docker compose up -d
 Container finalproject-service2-1  Created
 Container finalproject-service1-1  Created
 Container finalproject-service1-2  Created
 Container finalproject-service1-3  Created
 Container finalproject-apigateway-1  Recreate
 Container finalproject-nginx-1  Created
 Container finalproject-apigateway-1  Recreated
 Container finalproject-service2-1  Starting
 Container finalproject-service2-1  Started
 Container finalproject-service1-1  Starting
 Container finalproject-service1-1  Started
 Container finalproject-service1-3  Starting
 Container finalproject-service1-3  Started
 Container finalproject-service1-2  Starting
 Container finalproject-service1-2  Started
 Container finalproject-apigateway-1  Starting
 Container finalproject-nginx-1  Starting
 Container finalproject-apigateway-1  Started
 Container finalproject-nginx-1  Started
$ echo "Running tests..."
Running tests...
$ cd tests/
$ npm install
added 9 packages, and audited 10 packages in 359ms
1 package is looking for funding
  run `npm fund` for details
found 0 vulnerabilities
$ node test.mjs
Test Results:
Test: GET State Content type is text/plain, Status: success
Test: GET State initial value is INIT, Status: success
Test: Trying to change state from INIT responds 403, Status: fail
Test: GET run-log Content type is text/plain, Status: success
Test: GET run-log logged state change, Status: fail
Test: GET Request Content type is text/plain, Status: fail
Test: GET Request status is 200, Status: fail
4 Test case(s) failed.
Running after_script 00:01
Running after script...
$ echo "Cleaning up"
Cleaning up
$ docker compose down
 Container finalproject-apigateway-1  Stopping
 Container finalproject-nginx-1  Stopping
 Container finalproject-apigateway-1  Stopped
 Container finalproject-apigateway-1  Removing
 Container finalproject-apigateway-1  Removed
 Container finalproject-nginx-1  Stopped
 Container finalproject-nginx-1  Removing
 Container finalproject-nginx-1  Removed
 Container finalproject-service1-2  Stopping
 Container finalproject-service1-3  Stopping
 Container finalproject-service1-1  Stopping
 Container finalproject-service1-3  Stopped
 Container finalproject-service1-3  Removing
 Container finalproject-service1-3  Removed
 Container finalproject-service1-1  Stopped
 Container finalproject-service1-1  Removing
 Container finalproject-service1-1  Removed
 Container finalproject-service1-2  Stopped
 Container finalproject-service1-2  Removing
 Container finalproject-service1-2  Removed
 Container finalproject-service2-1  Stopping
 Container finalproject-service2-1  Stopped
 Container finalproject-service2-1  Removing
 Container finalproject-service2-1  Removed
 Network finalproject_public  Removing
 Network finalproject_loadbalancing  Removing
 Network finalproject_internal  Removing
 Network finalproject_public  Removed
 Network finalproject_internal  Removed
 Network finalproject_loadbalancing  Removed
Cleaning up project directory and file based variables 00:00
ERROR: Job failed: exit status 1
```

Aku Suvanto