Tampere University   Unit of Computing Sciences
COMP.SE.140 Continuous Development and Deployment

DevOps Final Project Report

By Yousif Al-Baghdadi
Student number: 152106946

Autumn 2024-2025

Contents

System Information

Features

Feature	status	Notes
Basic system requirement	Implemented	Shutdown not implemented
Automated System Tests	Implemented	
CI Pipeline (Build and test)	Implemented	
Test-drive development	Implemented	

Optional Features

Feature	status	Notes
Monitoring and Logging	Not implemented	
Static analysis	Nit implemented	
CI Pipeline (Deploy)	Not implemented	
Test for individual services	Not implemented	

Instructions for examiner to test the system

To run the project simply run:

```
docker-compose up -d
```

The system should run as per the requirements, the API running on port 8197 and the webpage on port 8198.

The default username and password are admin, admin.

Platform and environment information

Docker version (client and server): 26.0.0
Docker Compose version: v2.26.1-desktop.1
Docker server storage driver: overlay2
Host OS: Windows 11 Pro 22631.4602, Running Windows Subsystem for Linux 2
Architecture: x86_64, Processor: AMD Ryzen 7 5800H
Hardware Info: ASUS Vivobook M6500QH

CI/CD Pipeline

CI Workflow

The process starts when code is pushed into the repository. In this project, the Workflow is executed when code is pushed into the main branch, while normally the workflow tests the code when a merge request is opened to the main branch, however since this project is a solo project this is not implemented.

The workflow defined in the file [.gitlab-ci.yml] (file in the repository) is a system/integration test workflow. The workflow deploys the system locally using docker-compose and then runs the test. To run the test, it also needs the dotnet SDK which is being installed. The workflow passes if all tests pass.

The workflow utilizes the Docker-in-Docker paradigm, where we run docker inside of our workflow. To achieve this, the docker image is used as base runner image. It provides the runner with access to the docker client so that we can execute docker commands. Then, we also have docker-dind (docker in docker) service, which runs a separate docker container that acts as the docker server for our runner, the docker client connects to it, to deploy the system.

The system itself is being initialized and built as part of the docker-compose which includes building some images and running them. Check the file [docker-compose.yml].

Build Process

The building process of the services is being done with DockerFiles.

For service-1, the docker file is auto generated with dotnet project, some modifications were needed. There are a few stages in the Docker File. This separation is useful for making slimer Final images as they only contain the runtime tools rather than the SDK, and do not contain any build artifacts. Check the file [/service-1/ Dockerfile].

For Service-2, a single stage is being used because the base image is the same in the runtime and build process. We could have used two stages, the first stage to build with tsc and second stage to run project, this would make it so that the final image will not contain the ‘node_modules’ folder which would save around 25MB. This could be useful if service-2 had more dependency, however the simple docker file is left in this project. Check the file [/service-2/ Dockerfile].

Processes in other Services

For Nginx, we are using the nginx image so the purpose of the docker file is to copy the static page and the configuration into the image. Check the file [/nginx/ Dockerfile].

In addition, the project is using MongoDB to persist state between services. MongoDB is being used for two purposes, storing current state and state transitions.

It is worth noting that, for storing current state, Redis DB would be more ideal as it is in memory database, so it suits the use case more and provides faster performance, however, it is not suitable for logging purposes. Therefore, I opted to use MongoDB for both logging and state for simplicity rather than having both DBs.

I also considered using a SQL DB, however the initial creation of the database would be somewhat problematic as multiple service replicas would try to create the DB at the same time, so I would need a separate service just for DB Schema creation which would add a lot of complexity.

Automated Testing

The automated tests are using the testing framework TUnit, which is a modern, flexible, and fast testing framework with C#, TUnit is designed to aid with all testing types. In this project, simple assertion, logging and test dependencies are being utilized from the test framework, and the rest are features of dotnet with C#.

The tests run part of CI workflow or can be run locally. The test expects the system to be running locally. The tests attempt to send requests to localhost by default, or to the value of the environment variable CICD_APP_SERVICE_HOSTNAME.

The tests send HTTP requests and assert the response status code. The tests are separated into two groups, stateless and stateful tests. Stateless tests are supposed to always pass regardless of system state. While stateful tests affect or be affected by the state of the system and therefore they run in a specific order to assert that the system change states correctly.

Example Run of The Pipeline

Logs of jobs can be found in GitLab (Link in appendix). In addition, a copy of two failing and passing logs included a git gist in the appendix. The failing job log is slightly different from the passing one because the testing framework was changed.

Test driven was followed in the project in most cases. However, the pipeline was often not used, instead tests were run locally because running locally is faster than running the CI.

Reflections

Unimplemented feature – Monitoring and logging

Cloud native applications usually have a feature of being observable. Observability means few things, centralized logs, metrics, and traces.

Logs and metrics could be achieved with a monitoring tools stack like (Prometheus, Grafana and Loki) or (Elasticsearch, Logstash, Kibana). While traces require some special OpenTelemetry instrumentation which is supported by dotnet.

I attempted to implement logging and metrics with Grafana and Loki, but I kept running into problems and could not get them to work correctly. So, I decided to skip this feature in this project given the limited time frame.

Unimplemented feature – Deployment

Deployment to a remote server is certainly an important feature of a CI/CD system.

In theory, the CI deployment could be implemented by doing the following:

First, we need to **condition the deployment server** by doing the following:

- Ensuring that the correct ports are open in the firewall so that our service can be accessed.
- Ensuring that SSH is working, and access is possible through the firewall.
- Ensuring that Docker and Docker-Compose are installed.
- Ensuring that Git is installed so we can run git clone.

Then the CI can deploy by doing the following:

- SSH into the deployment server
- Cloning the git repository locally
- Executing `docker-compose up`

Overlooked challenges – Secrets Management

In this project, default passwords and usernames (secrets/credentials) are used because the system is just for testing and learning. However, in a real application, securing and managing these values is a very challenging task, as they must remain off any git repository and with controlled access.

This mostly means that secrets must be stored in Environment variables, and secret managers could be used to load secrets into apps. For example, the file .htpasswd must not be store in git, instead created dynamically with a script, and loaded from environment variables.

Learning reflection

It was good to learn about GitLab-CI and how it works, and setting-up the runner. Also, creating automated system/integration tests that run on the entire system rather than in-process test are new to me and a good learning experience.

Difficulties

Understanding and Running Docker-in-Docker was a bit tricky, and the requirement of having privileged runner made me question if this would be the right approach. Also, I have read online that Docker-in-Docker is no longer the favored and that it is only exist because it ‘helps in developing docker itself’, I’m still not quite sure if these claims are true, and they also sent me looking for alternative and I spent quite a long time thinking about alternative solutions to run and test the system but could not come up with anything else.

Effort Estimation

Approximately 55 hours were needed to complete all the project parts, excluding the time to implement the first and second assignments.

Appendix A – CI Run Log Failing

https://gist.github.com/Yousif-FJ/73661be9c5a0c700ab3eaa8e9e262c93

Appendix B – CI Run Log Passing

https://gist.github.com/Yousif-FJ/105308654c1cd4fbf1d2f1b21cd98cb3

Appendix C – Repository

https://github.com/Yousif-FJ/docker-compose-experiment

https://compse140.devops-gitlab.rd.tuni.fi/vkyoal/cicd-porject