Group Q DevOps report

Michael Jeppesen Marcin Sroka

micje@itu.dk msro@itu.dk

[1em] Alin Plamadeala Philip Blomholt

alpl@itu.dk phbl@itu.dk

May 15, 2025

Contents

1 Introduction

In this report, we will describe our approach to the tasks given during the DevOps 2025 elective course. Our report will delve into three different perspectives, providing a *System* perspective in section 2, a *Process* perspective in section 3 , and finally we provide a *reflective* perspective in section 4

2 System perspective

2.1 Design and architecture

2.2 Dependencies

Git/Github

Terraform

Docker

S3

PostreSQL

Traefik

Grafana

Alloy

Loki

Prometheus

Golang

Playwright

Ansible

Bash?

2.3 Important interactions

2.4 Current state

We used several static analysis tools and tests to keep the code in good shape. The current, most recent implementation is working correctly and passing all tests: Go linter, Go tests, web end-to-end tests, terraform linter, and terraform observability check.

Below are overviews of two static code analysis tools applied to the current implementation:

- SonarQube: 3.3k lines of code, 0 security issues (grade A), 5 reliability issues (grade C), 17 maintainability issues (grade A), 2.3% code duplication (Below the suggested 3% threshold).
- CodeClimate: maintainability grade A, 5 code smells, 2 duplications, 0 other issues.

Most code duplication and reliability issues can be found in the end-to-end tests.

3 Process perspective

3.1 Stages and tools in CI/CD

3.2 Monitoring

An important part of DevOps relates to operations, and ensuring that existing code is running as intended. Through monitoring, we are able to continuously check if our were active and processed incoming requests. For our setup, we have made use of Grafana as our visualisation tool, scraping information about our mini-twit application using the open-source web application scraper tool Prometheus. Prometheus scrapes information from the "/metrics" endpoint, created using the go package `ginprom`, and provides a multitude of information on the go application behind mini-twit represented using time series data. Prometheus maintains it own storage locally, and to ensure that replacing the mini-twit virtual machine does not lose its time series information, we created a dedicated volume in digital ocean to store this data, alongside the configuration files for Prometheus.

3.3 Logging

While monitoring provides an overview of a system through metrics and health checks, we need to use logging to get more detailed information about events that have transpired, their timestamps and other relevant information. Our logging setup has been based around Grafana as a visualisation tool, Grafana Loki as a log aggregation tool, and Grafana Alloy as the logging tool. Our logging efforts were divided amongst our docker containers, reading their logs. In extension to this, we implemented logs for our mini-twit application using the standard logging library in Golang `slog`. Here we focused on implementing log events for all of our event handlers, since their process information was deemed most important.

Figure 1: Example of logging with FollowHandler showcasing how errors are propagated as a log.

3.4 Security assessment

To identify possible attack surfaces, we made a full port-scan of our server which came up with the following open ports;

22 SSH

80, 443 web-app through traefik

3000 grafana

3100 loki

12345 alloy

9090 prometheus

We reasoned that the following were already secure; SSH, web-app, grafana and prometheus. That left Loki and All

3.5 Scaling strategy

3.6 Use of Artificial Intelligence

In our project, we have used Anthropic's artificial intelligence *Claude* and OpenAI's ChatGPT to assist with certain tasks (Anthropic, 2025; OpenAI, 2025). While ChatGPT was mostly used initially, the output that we received quickly rendered it useless, and as such, we adopted more towards

Claude instead. Claude has mostly been helpful when it came to adopting new technologies that none of us had worked with prior, or adjusting smaller bits of code. However, we often found that Claude tends to miss the bigger picture of a CI/CD setup, outputting wrong solutions since it involves many different source files and contextual parts. Additionally, in several cases, we found that Claude either overcomplicates a solution or does not address security concerns properly, such as access modifiers or port openings. As such, we would argue that the use of artificial intelligence has generally helped adopt new technologies at a faster rate, but this also comes with several issues, including having to address the code that it generates and spending excessive amounts of time prompting.

4 Reflection perspective

4.1 Software Evolution and Refactoring

4.2 Operation

Throughout the project, we learned a major lesson about software. Namely, the key to success is automation and fast feedback loops. If we automate testing, integration, and deployment, then the friction to get from a finished software update to a deployed, working service is minimal. This encourages software evolution in small steps, which in turns makes it much easier to spot bugs and find their cause, and bugs are also found a lot earlier.

We encountered some major issues relating to operations:

- After implementing volumes, we had an issue where the `latest id` entry used by the API was still in a text file in the code, and thus would not be persisted across deployments. We fixed this issue by incorporating `latest id` in the volume as a single-row single-column database table. Related issue: https://github.com/devops-q/devops/issues/250

- We found that we didn't set up authentication for front of loki and alloy. We tried to set it up, but without success. We then realized that loki and alloy are part of the same network and don't need to expose ports, so we stopped port exposition for them, preventing unauthorized access.

4.3 Maintenance

4.4 Our approach to DevOps

Throughout this project, we have sought to implement several DevOps practices to increase the efficiency of our workflow. Here our initial goal was to reduce the amount of time between committing code, code being review and subsequently merged, which we found to be a tedious process. One of the core ideals of DevOps is being able to reduce this lead time between committing and merging, such that developers can work faster and in smaller batches while still maintaining high code quality (Kim et al., 2016). We achieved this principle through the implementation of continuous integration, providing a *standardisation* measure for our coding quality using linters and tests as a requirement for merging a commit. This provided a much lower lead time, since code reviews were no longer being done before a merge

- instead requiring our testing suite to pass. This fostered a culture, where each of us would be able to work on a task without any major repercussions, thereby increasing both risk-taking and experimentation. Another aspect is DevOps relates to our ability to automate repetitive tasks in an effort to reduce the time spent of a task, but in extension also avoid human errors, that might occur when multistep workflows needs are required.

References

Anthropic. (2025). Claude.

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The devops handbook: How to create world-class agility, reliability, and security in technology organizations*. IT Revolution Press.

OpenAI. (2025). Chatgpt.