IT UNIVERSITY OF COPENHAGEN

DevOps, Software Evolution and Software Maintenance

COURSE CODE: BSDSESM1KU

BACHELOR IN SOFTWARE DEVELOPMENT

# DevOps: ITU-MiniTwit

GROUP R — RHODODEVDRON

IT UNIVERSITY OF COPENHAGEN

| Name | Email |
|---|---|
| Adrian Valdemar Borup | adbo@itu.dk |
| Albert Rise Nielsen | albn@itu.dk |
| Joachim Alexander Borup | aljb@itu.dk |
| Thomas Wolgast Rørbech | thwr@itu.dk |

May 24, 2022

## Table of Contents

# 1 System Perspective

## 1.1 Software license compatibility

To check if the software licenses of the libraries we've used are compatible with the MIT license in our repository, we have used lichen [8] to automatically check the licenses of all used libraries. This process is documented in Appendix B. The results showed that all the libraries were compatible, except one because the tool couldn't find its license. However, checking its repository manually revealed a compatible MIT license.

## 1.2 system dependencies

# 2 Process Perspective

## 2.1 Team interaction

The team consists of 4 developers, who meet once or twice a week to work on the project. At each meeting, the developers discuss the tasks at hand, create issues, divide the work, and then spends the rest of the time working on their tasks. There is no hierarchy - only 4 like-minded developers who makes decisions in unison.

New tasks are created as issues on GitHub, and then assigned to the developer who will be working on it. Maintaining a set of issues on GitHub lets us keep an overview of the planned tasks, and allows for any of the developers to easily add a new task at any time.

## 2.2 Organisation of repositories

The group uses GitHub to manage repositories. On GitHub, we have a dedicated organization that owns all of the project repositories. This way, we can keep related repositories together and reuse permission settings on GitHub. Currently, we have the following repositories:

1. `Devops-2022-Group-R/itu-minitwit`, which is the main project repository. Here, we have the backend web server which handles all business logic.
2. `Devops-2022-Group-R/itu-minitwit-frontend`, containing our frontend web-app.
3. `Devops-2022-Group-R/flag-tool`, which is a rewrite of the original flag tool given with the project template. The tool has been rewritten and moved to a separate repository.
4. `Devops-2022-Group-R/bump-tool`, which is a small tool to help with finding the next version bump based on a pull request's tag (major/minor/patch) and the project's current version.

The general philosophy has been to separate parts that can exist alone with a single responsibility. This way, all issues, pull requests, and releases are related to the topic of the repository. In a monolithic repository, you have to put in more effort to specify which part of the repository is relevant for a PR or issue. This also allows for easier and more simple CI/CD pipelines, and it becomes more obvious what a release changes. One example of this separation is the frontend web application, which is completely separated from the backend, with Kubernetes specs and CI/CD pipelines in its own repository.

However, we have not been as good at following this philosophy as we would like to. For example, our monitoring is a separate entity from our web server, but all the monitoring configuration is stored in the `itu-minitwit` repository [2] [4]. Instead, we should have moved this to a separate repository because it does not have anything to do with how the web server operates.

The same goes for the LaTeX files that this report consists of. We would have liked to have a separate repository for this in order to keep Git history clean and CI/CD more separate, but in this case it is a project requirement to have it in the main repository.

## 2.3 Git branching strategy

We have applied trunk-based development, meaning we branch out from the main branch, have a branch that lives shortly until the changes are implemented, and then merge it directly into the main branch. This keeps unrelated changes separate and makes PRs easier to review and merge. It's worth noting that we do not utilise release branches from trunk-based development, because we release continuously for each merge into the main branch.

## 2.4 CI/CD chains

We use CircleCI as a continous integration and continous delivery platform, and have configured a workflow [3] with jobs to build, test, and deploy our service, generate a changelog, as well as building our LaTeX project. See Appendix A for a full CI/CD workflow diagram.

### 2.4.1 Building, testing, and static analysis

This part of the workflow quite simply installs the dependencies, and then builds and tests the project. Additionally it runs the following three static code analysis jobs:

- A Go linters runner: `golangci-lint`, which runs dozens of linters in parallel.
- A cyclomatic complexity calculator: `gocyclo`, to ensure that the code has low complexity. Otherwise, the job will exit with code 1 until we've refactored the functions in question.
- A developer security platform, `Snyk`, to find and fix security vulnerabilities in code, dependencies, and containers.

#### 2.4.2 Deploying our service

CircleCI will also:

- Publish a GitHub release
- Push our changes to the four images in Docker Hub (the server, front-end, Prometheus, and Grafana)
- Deploy to their respective images in the managed Kubernetes service in the remote Azure server
- Apply infrastructure via Terraform

#### 2.4.3 Generating a changelog, and building the LaTeX project

We use CircleCI Github Changelog Generator [1] to generate a changelog based on our tags, issues, labels and pull requests on GitHub.

In order to include a build of the LaTeX project in the CircleCI workflow, we've written a script to let CircleCI run a custom Docker image, build a PDF, and then commit it to the branch.

### 2.5 Monitoring

To monitor our systems, Prometheus is used to collect metrics and Grafana is used to visualize them. We chose to monitor the number of requests and request latency for each endpoint on our server, and we monitored the CPU and memory usage of it. We chose these because they give a quick overview of how the system is performing. The remaining information is stored in our logs.

For an example where we had an issue with our system, which was reflected in the monitoring, see Appendix C.

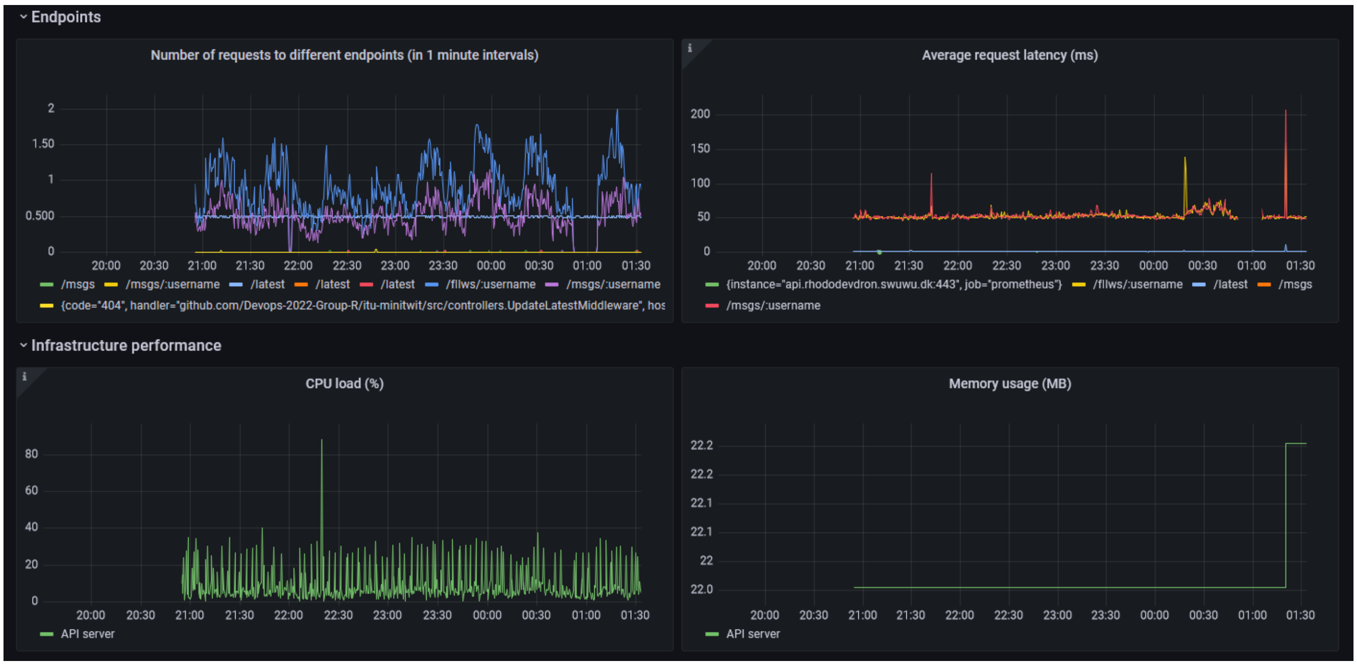

Endpoints
Number of requests to different endpoints (in 1 minute intervals)
/msgs
/msgs/:username
/latest
/fllws/:username
{code="404", handler="github.com/Devops-2022-Group-R/itu-minitwit/src/controllers.UpdateLatestMiddleware", hos
Average request latency (ms)
{instance="api.rhododevdron.swuwu.dk:443", job="prometheus"}
Infrastructure performance
CPU load (%)
API server
Memory usage (MB)

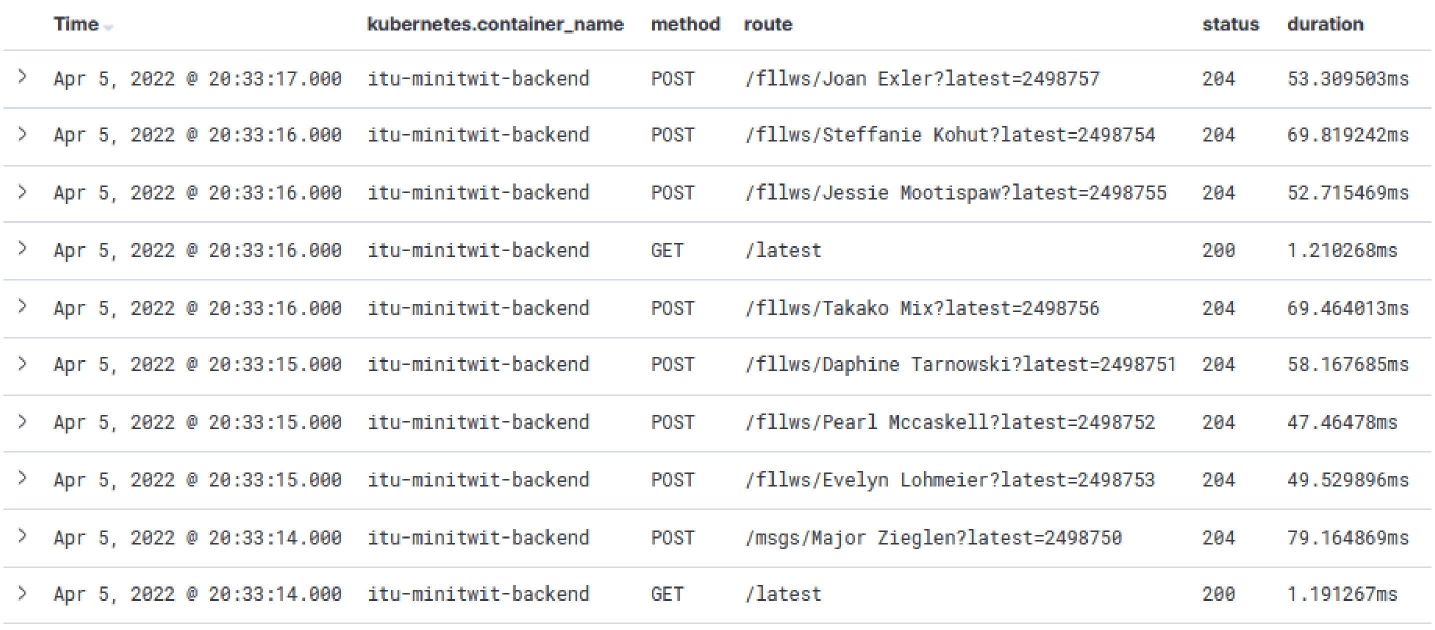
**Figure 1:** *A sample of our Grafana dashboard.*

## 2.6 Logging

To manage logging in our systems, we use the EFK stack (Elasticsearch, Fluentd, and Kibana). It works like this:

1. A pod in the cluster is logging to stdout
2. The stdout is sent to log files by Kubernetes
3. Fluentd picks up on logs in these files and sends them to Elasticsearch
4. Kibana is used to visualize the logs

We log using JSON, which allows us to select specific fields to display in Kibana, making the logs easy to read, filter, and sort by. Figure 2 shows how we display logs in an easy-to-read format.



| | Time | kubernetes.container_name | method | route | status | duration |
|---|---|---|---|---|---|---|
| > | Apr 5, 2022 @ 20:33:17.000 | itu-minitwit-backend | POST | /fllws/Joan Exler?latest=2498757 | 204 | 53.309503ms |
| > | Apr 5, 2022 @ 20:33:16.000 | itu-minitwit-backend | POST | /fllws/Steffanie Kohut?latest=2498754 | 204 | 69.819242ms |
| > | Apr 5, 2022 @ 20:33:16.000 | itu-minitwit-backend | POST | /fllws/Jessie Mootispaw?latest=2498755 | 204 | 52.715469ms |
| > | Apr 5, 2022 @ 20:33:16.000 | itu-minitwit-backend | GET | /latest | 200 | 1.210268ms |
| > | Apr 5, 2022 @ 20:33:16.000 | itu-minitwit-backend | POST | /fllws/Takako Mix?latest=2498756 | 204 | 69.464013ms |
| > | Apr 5, 2022 @ 20:33:15.000 | itu-minitwit-backend | POST | /fllws/Daphine Tarnowski?latest=2498751 | 204 | 58.167685ms |
| > | Apr 5, 2022 @ 20:33:15.000 | itu-minitwit-backend | POST | /fllws/Pearl Mccaskell?latest=2498752 | 204 | 47.46478ms |
| > | Apr 5, 2022 @ 20:33:15.000 | itu-minitwit-backend | POST | /fllws/Evelyn Lohmeier?latest=2498753 | 204 | 49.529896ms |
| > | Apr 5, 2022 @ 20:33:14.000 | itu-minitwit-backend | POST | /msgs/Major Zieglen?latest=2498750 | 204 | 79.164869ms |
| > | Apr 5, 2022 @ 20:33:14.000 | itu-minitwit-backend | GET | /latest | 200 | 1.191267ms |

**Figure 2:** *Example of how we view logs in Kibana.*

It's not shown on the figure, but we also have a column for error messages, which we've experienced makes it easy to diagnose problems in the system. To diagnose performance issues, the "duration" column has also been helpful. By the end of the project, the mentioned log output has been enough to diagnose bugs and get an overview of what's happening in the system.

### 2.7 Strategy for scaling and load balancing

We have deployed all of our components with Kubernetes deployments [6], and configured them to run behind Kubernetes services [7], this ensures component level load balancing. Using deployments also allows us to manually scale the number of instances of each component. While deployments do have the ability to autoscale, after some configuration, we have decided against it due to cost issues and because our monitoring indicated that the system was not getting enough load to justify scaling.

Deployments also support zero downtime updates. This is achieved by changing settings on the deployment, such as the image to be used, then Kubernetes creates a new pod, and then kills an old pod when the new is ready, repeating the process until all pods have the new configuration.

## 3 References

[1] Comino, Murillo. *CircleCI Github Changelog Generator*. Accessed: May 18 2022. Oct. 2020. URL: https://github.com/onimur/circleci-github-changelog-generator.

[2] Group R. *Configuration files for ITU-MiniTwit monitoring*. Accessed: May 17 2022. Apr. 2022. URL: https://github.com/Devops-2022-Group-R/itu-minitwit/tree/master/monitoring.

[3] Group R. *ITU-MiniTwit CircleCI workflow*. Accessed: May 17 2022. Apr. 2022. URL: https://github.com/Devops-2022-Group-R/itu-minitwit/blob/master/.circleci/config.yml.

[4] Group R. *Kubernetes specs for ITU-MiniTwit monitoring*. Accessed: May 17 2022. Apr. 2022. URL: https://github.com/Devops-2022-Group-R/itu-minitwit/tree/master/.infrastructure/kubernetes/monitoring.

[5] Gene Kim. *The DevOPS Handbook - How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. It Revolution Press, 2016. ISBN: 9781942788003.

[6] Kubernetes Website Contributors. *Deployments | Kubernetes*. Accessed: May 18 2022. Feb. 2022. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

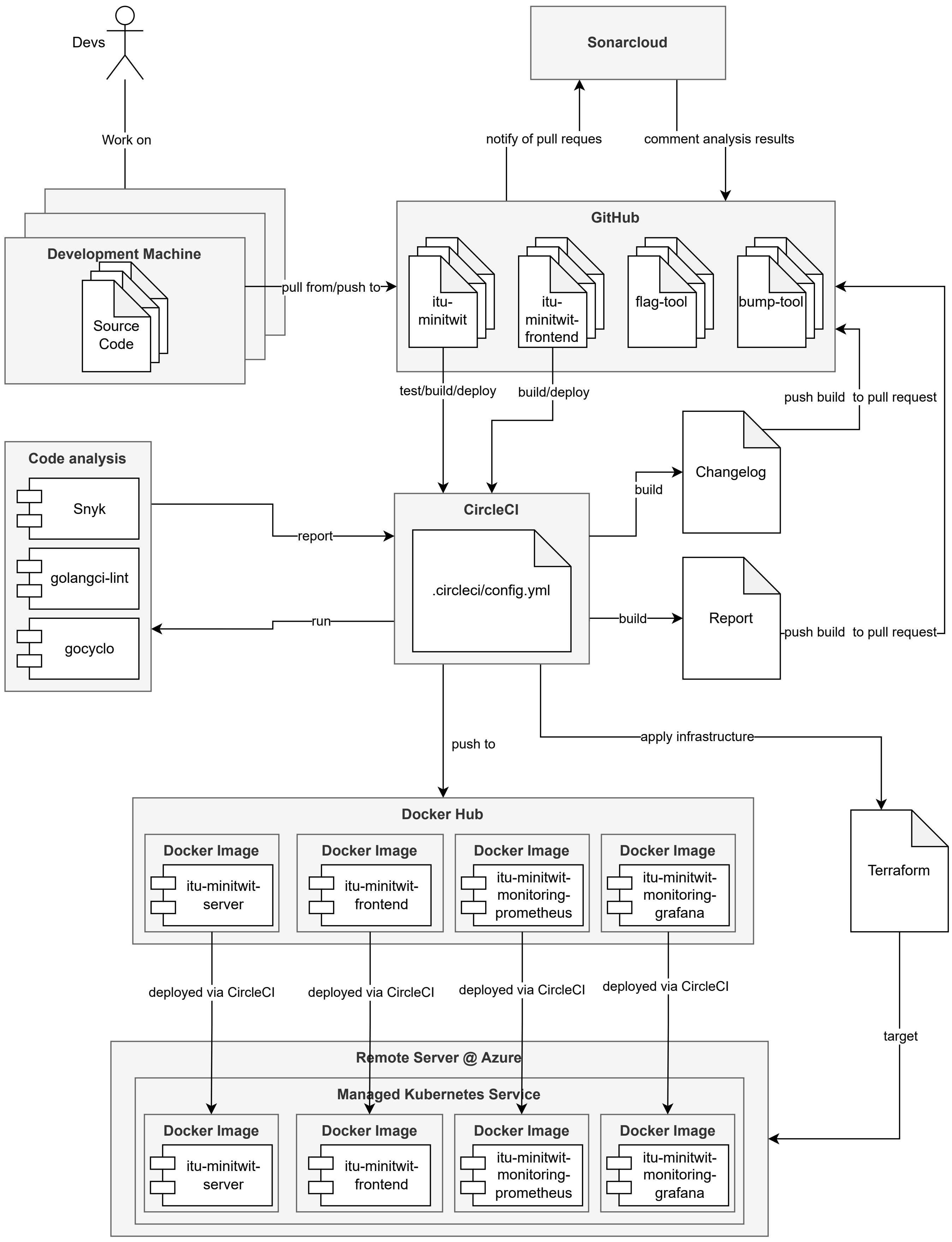
[7] Kubernetes Website Contributors. *Service | Kubernetes*. Accessed: May 18 2022. May 2022. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[8] UW Labs. *lichen*. Accessed: March 20 2022. URL: https://github.com/uw-labs/lichen.

# Appendices

## Table of Contents

## A CI/CD workflow diagram



## B Checking software license compatibility

To run the license scan tool, we used:

```
# Install
go install github.com/uw-labs/lichen@latest
# Run in root of Devops-2022-Group-R/itu-minitwit/
go build -o my-binary ./src
lichen my-binary
```

This reveals that the only disallowed library we use is `Azure/azure-sdk-for-go/sdk/internal` because it's missing a license. However, if you check the repository manually, you can find a MIT license in the root.

The output of the command is:

```
github.com/Azure/azure-sdk-for-go/sdk/azcore@v0.19.0: MIT (allowed)
github.com/Azure/azure-sdk-for-go/sdk/azidentity@v0.11.0: MIT (allowed)
github.com/Azure/azure-sdk-for-go/sdk/internal@v0.7.0:  (not allowed - unresolvable license)
github.com/beorn7/perks@v1.0.1: MIT (allowed)
github.com/cespare/xxhash/v2@v2.1.2: MIT (allowed)
github.com/denisenkom/go-mssqldb@v0.12.0: BSD-3-Clause (allowed)
github.com/gin-contrib/sse@v0.1.0: MIT (allowed)
github.com/gin-gonic/gin@v1.7.7: MIT (allowed)
github.com/go-playground/locales@v0.14.0: MIT (allowed)
github.com/go-playground/universal-translator@v0.18.0: MIT (allowed)
github.com/go-playground/validator/v10@v10.10.0: MIT (allowed)
github.com/golang-sql/civil@v0.0.0-20190719163853-cb61b32ac6fe: Apache-2.0 (allowed)
github.com/golang-sql/sqlexp@v0.0.0-20170517235910-f1bb20e5a188: BSD-3-Clause (allowed)
github.com/golang/protobuf@v1.5.2: BSD-3-Clause (allowed)
github.com/jinzhu/inflection@v1.0.0: MIT (allowed)
github.com/jinzhu/now@v1.1.4: MIT (allowed)
github.com/leodido/go-urn@v1.2.1: MIT (allowed)
github.com/mattn/go-isatty@v0.0.14: MIT (allowed)
github.com/mattn/go-sqlite3@v1.14.11: MIT (allowed)
github.com/matttproud/golang_protobuf_extensions@v1.0.1: Apache-2.0 (allowed)
github.com/pkg/browser@v0.0.0-20180916011732-0a3d74bf9ce4: BSD-2-Clause (allowed)
github.com/prometheus/client_golang@v1.12.1: Apache-2.0 (allowed)
github.com/prometheus/client_model@v0.2.0: Apache-2.0 (allowed)
github.com/prometheus/common@v0.32.1: Apache-2.0 (allowed)
github.com/prometheus/procfs@v0.7.3: Apache-2.0 (allowed)
github.com/shirou/gopsutil@v3.21.11+incompatible: BSD-3-Clause (allowed)
github.com/sirupsen/logrus@v1.6.0: MIT (allowed)
github.com/tklauser/go-sysconf@v0.3.10: BSD-3-Clause (allowed)
github.com/tklauser/numcpus@v0.4.0: Apache-2.0 (allowed)
github.com/ugorji/go/codec@v1.2.6: MIT (allowed)
github.com/zsais/go-gin-prometheus@v0.1.0: MIT (allowed)
```

```
golang.org/x/crypto@v0.0.0-20220210151621-f4118a5b28e2: BSD-3-Clause (allowed)
golang.org/x/net@v0.0.0-20211112202133-69e39bad7dc2: BSD-3-Clause (allowed)
golang.org/x/sys@v0.0.0-20220209214540-3681064d5158: BSD-3-Clause (allowed)
golang.org/x/text@v0.3.7: BSD-3-Clause (allowed)
google.golang.org/protobuf@v1.27.1: BSD-3-Clause (allowed)
gopkg.in/yaml.v2@v2.4.0: Apache-2.0, MIT (allowed)
gorm.io/driver/sqlite@v1.2.6: MIT (allowed)
gorm.io/driver/sqlserver@v1.3.1: MIT (allowed)
gorm.io/gorm@v1.23.1: MIT (allowed)
2022/03/20 14:23:40 1 error occurred:
      * github.com/Azure/azure-sdk-for-go/sdk/internal@v0.7.0: not allowed - unresolvable
        license
```

## C Monitoring example: High CPU usage

Figure 4 and Figure 5 show the status of our system in Grafana and Azure when we tried to implement logging. The initial implementation used up high amounts of our CPU resources, which is reflected in the response times and CPU recording metrics.

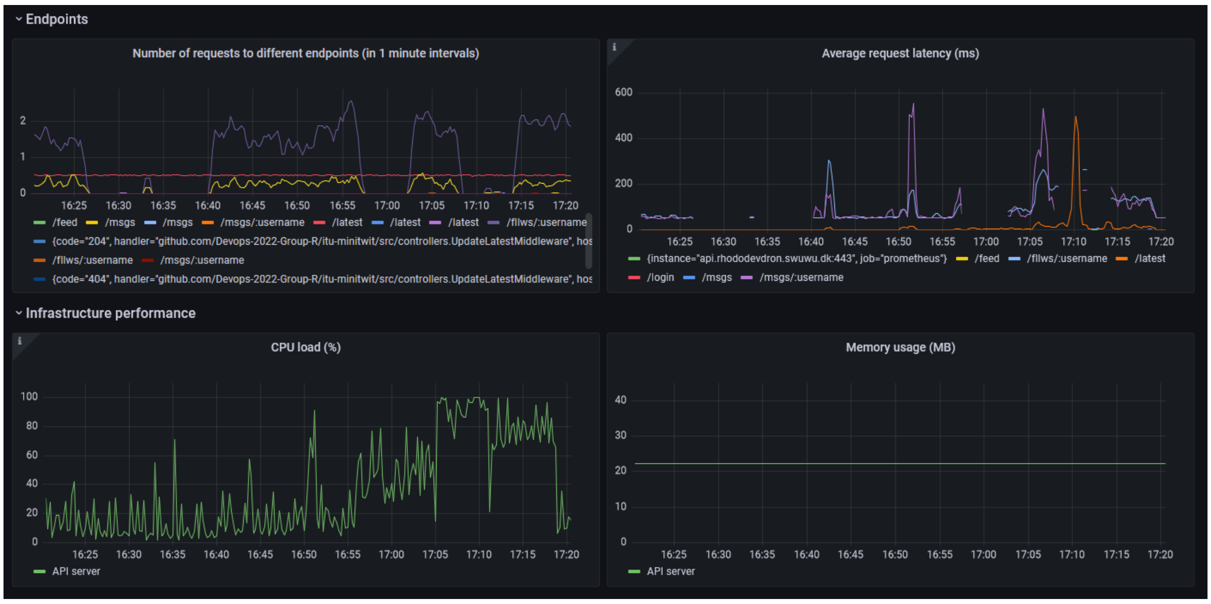


**Figure 4:** *A snapshot of our Grafana monitoring dashboard.*

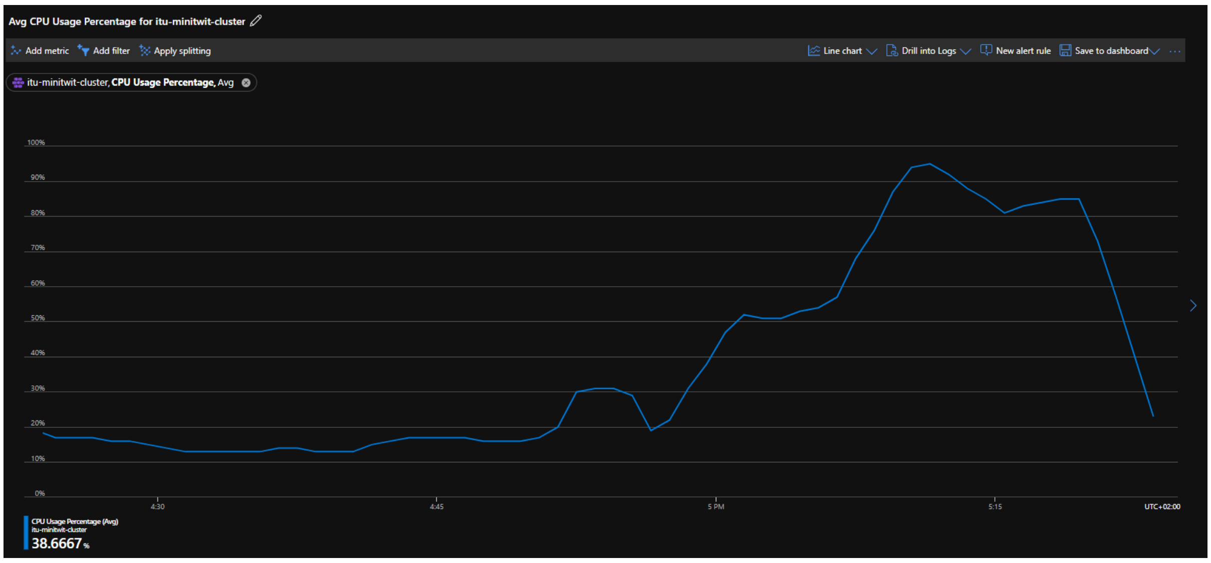


**Figure 5:** *A graph from Azure's monitoring showing the CPU usage of our cluster.*