DevOps Project: Minitwit

MSc. Group C - DuwuOps

Babette Bækgaard (babb@itu.dk)
David Martin Sørensen (daso@itu.dk)
Dea Lærke Thim (death@itu.dk)
Gustav Müller Christoffersen (guch@itu.dk)
Nikoline Burman (nibu@itu.dk)
Simone Haukjær Jakobsen (hasj@itu.dk)

May, 2025

Contents

1 Introduction **2**
1.1 System Depiction . . . 2
1.2 Tech-Stack Overview . . . 2

2 System Perspective **3**
2.1 Minitwit . . . 3
2.2 Orchestration . . . 7
2.3 Deployment . . . 9
2.4 Database . . . 9

3 Process Perspective **11**
3.1 CI/CD . . . 11
3.2 Monitoring . . . 13
3.3 Logging . . . 16
3.4 Strategy for Scaling and Upgrades . . . 17
3.5 AI Use . . . 18

4 Security Assessment (May 24, 2025) **18**
4.1 Risk Identification . . . 18
4.2 Risk Scenarios . . . 19
4.3 Risk Analysis . . . 19
4.4 Mitigation and Remediation . . . 19

5 Reflections **20**
5.1 Evolution, Operation, and Maintenance . . . 20

Bibliography **21**

1 Introduction

1.1 System Depiction

An informal depiction of the system as an initial overview is provided in fig. 1.

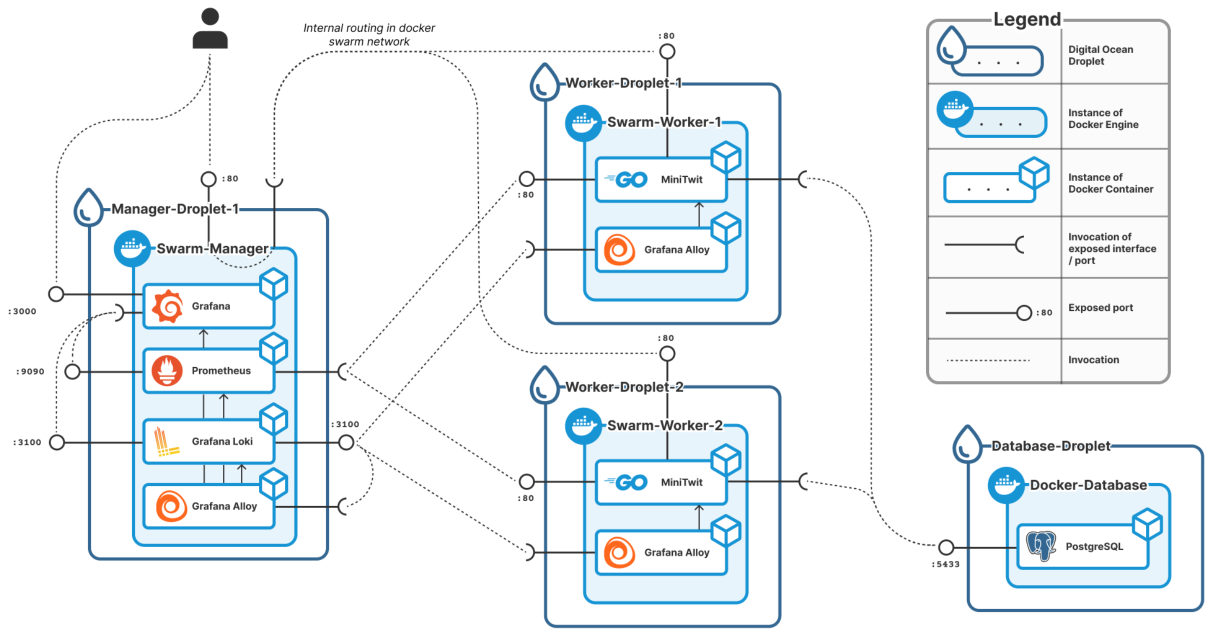

Figure 1: Informal system depiction and legend

1.2 Tech-Stack Overview

Topic	Tech Choice	Section
Refactoring	GoLang, Echo	**2.1 Minitwit**
Orchestrazation	Docker	**2.2 Orchestration**
Deployment	DigitalOcean	**2.3 Deployment**
CI/CD	GitHub Actions	**3.1 CI/CD**
Database	PostgreSQL	**2.4 Database**
Monitoring	Prometheus, Grafana	**3.2 Monitoring**
CI Static Analysis	golangci-lint, Dependabot	**3.1 CI/CD**
Maintainability	SonarQube, CodeClimate	**2.1 Minitwit**
Logging	Loki, Alloy, Grafana	**3.3 Logging**
Scalability	Docker Swarm	**3.4 Strategy for Scaling and Upgrades**
Security	CodeQL, Dependabot	**4 Security Assessment**

Table 1: Overview of tech-stack.

2 System Perspective

2.1 Minitwit

2.1.1 Programming Language

GoLang (Go) was chosen based on documentation [1], community support [2], industry adoption [3], and the notion of being 'lightweight' - both in terms of syntax and performance overhead. The group additionally wanted to prioritize a language they had limited experience with.

The programming languages C#, Java, Go, and Crystal were considered. Java and C# were discarded as candidates as they were considered too verbose as object-oriented languages, and the group had extensive previous experience with them. This led to a comparison between Go and Crystal, outlined in tbl. 2.

Topic / Lang	GoLang	Crystal
Team Competences	Some prior exposure in small capacities	No prior experience
Industry Usage	Extensive adoption [3]	Limited adoption [3]
Performance	Fast	Fast
Concurrency	Yes	Yes
Documentation	Well-documented [1]	Good but less extensive [3]
Community	Large and active [3]	Smaller and less active [3]

Table 2: Comparison between Go and Crystal.

Echo was chosen as the Go web framework for the REST APIs due to its perceived ease-of-use [4], high-performance [5], [6], and its native Prometheus interoperability. Table tbl. 3 outlines the comparison between select web frameworks for Go.

Framework	Gin	Chi	Echo	Gorilla
Prior Experience	Some	None	None	None
Performance	Fast [5]	Fast [5]	Fast [5], [6]	Fast [7]
Features	Moderate [8]	Many [8]	Many [4], [6]	Many [7]
Scalability	Great [8]	Great	Good [6]	Great [7]
Community	Good [6]	Good	Growing [6]	Stale [7]
Ease of use	Good [7], [8]	Complex [9]	Great [4]	Complex [7]
Popularity	High [10]	Low [10], [11]	Medium [11]	Low

Table 3: Comparison of select Go web frameworks.

2.1.2 External dependencies in GoLang

Dependency	Description
labstack/echo/v4	Web framework for routing and HTTP handling.
gorilla/sessions	Session management with secure cookie support.
lib/pq	PostgreSQL driver for database connectivity.
golang.org/x/crypto	Cryptographic utilities for security features.
prometheus/client_golang	Prometheus client for metrics and monitoring.
shirou/gopsutil/v4	System metrics collection for health monitoring.
klauspost/compress	Compression libraries to optimize data transfer.

Dependency	Description
golang.org/x/sys	Low-level OS interaction and system calls.
google.golang.org/protobuf	Protocol Buffers support for data serialization.
gorilla/securecookie	Secure cookie encoding/decoding for session safety.
Gravatar	External web service providing avatar images generated from email hashes.

Table 4: External dependencies for the Go implementation of MiniTwit. (see `go.mod` for more details.)

2.1.3 Design and Architecture

The architecture of `src/` is explored through two views:

1. A module view of the MiniTwit implementation, depicted in a UML module diagram (see fig. 2), and table detailing each module with corresponding description (see tbl. 5).
2. Two UML sequence diagrams (fig. 3 and fig. 4) showcasing the user requests processes of "follow"-interaction through respectively the *UI* and the testing *API* (note: these are separate endpoints).

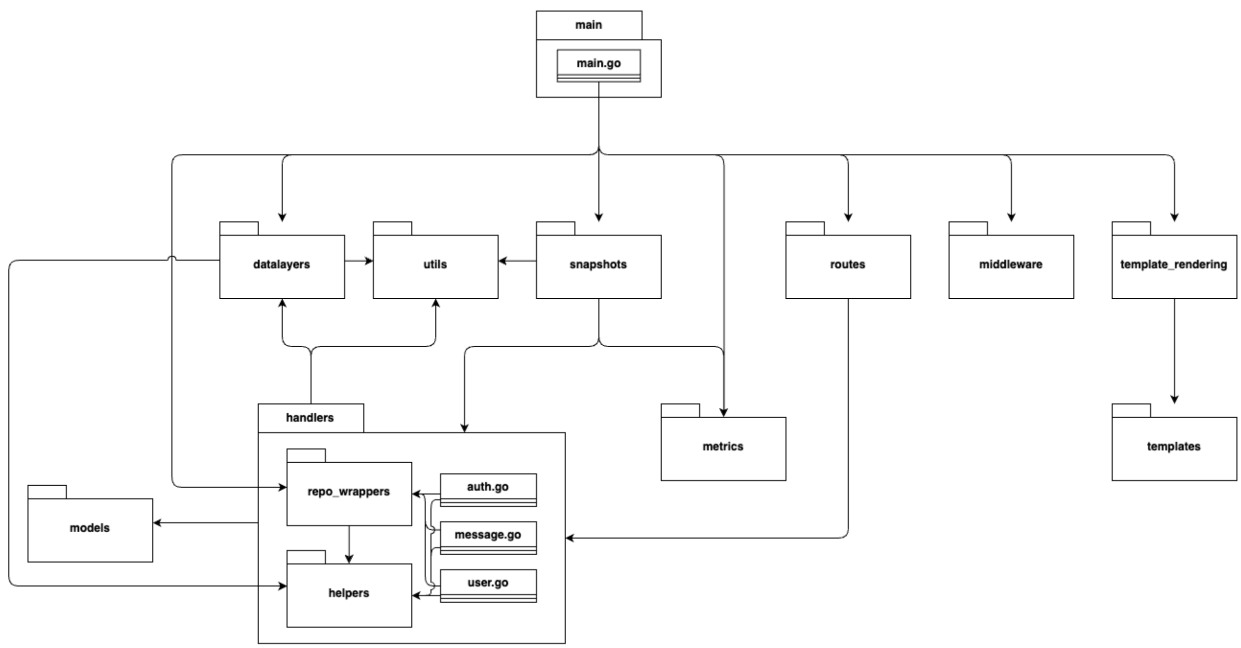

Figure 2: Module (Package) diagram of the GoLang MiniTwit implementation. **Note** `handlers` module is expanded to include GoLang implementations, in order to highlight its complexity.

Module	Description
`datalayer`	Responsible for database connection and initialization. Implements the data access layer through `repository.go` and its interface `irepository.go`.
`models`	Contains data models: `User`, `Message`, `Follower`, and `LatestAccessed`.
`handlers`	Central logic of the system. Orchestrates operations for each model.
`handlers.repo_wrappers`	Utility functions extending repository logic.
`handlers.helpers`	Shared logic.
`routes`	Maps HTTP endpoints to their corresponding handlers.

Module	Description
`metrics`	Registers custom Prometheus metrics to monitor system statistics.
`middleware`	Applies Cross-Site Request Forgery middleware.
`snapshots`	Handles Prometheus snapshots of database.
`template_rendering`	Renders templates used by the frontend.
`templates`	Holds frontend HTML files.
`utils`	Contains shared utility methods used across the codebase.

Table 5: Description of modules in GoLang MiniTwit implementation.

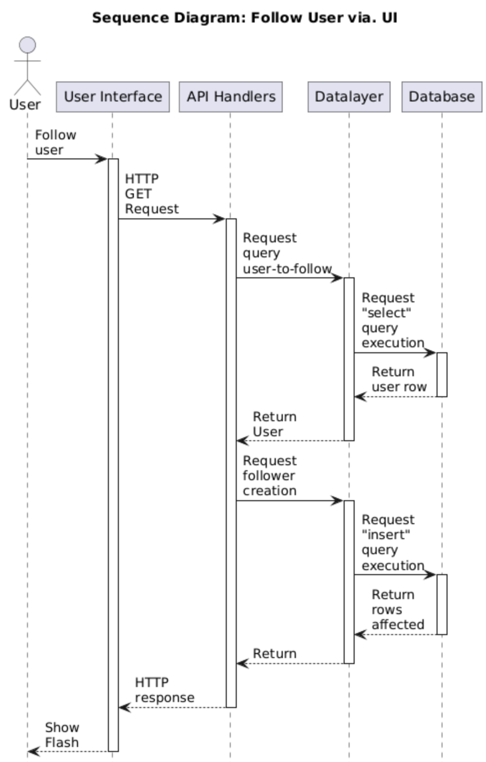

Figure 3: Sequence diagram - Follow request via UI. Note: "API Handlers" refers to files from the `handlers` package.

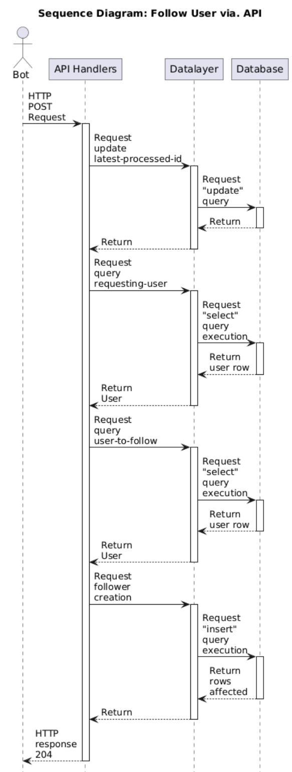

Figure 4: Sequence diagram - Follow request via API. Note: "API Handlers" refers to files from the `handlers` package.

2.1.4 Current State of the System

The analysis tools SonarQube and CodeClimate were utilized to gauge the complexity of the implementation (see tbl. 6 and tbl. 7). Both tools show that the `handlers` module has relatively high complexity, which may require attention for maintainability.

Metric	**Value**
Lines of Code (LOC)	1,591
Code Duplication	4.1%
Security Hotspots	8
Overall Rating	A (Excellent quality)
Cyclomatic Complexity	216 (handlers: 151)
Technical Debt	~1 hour 7 minutes

Table 6: Summarized quality metrics from SonarQube analysis.

Metric	Value
Lines of Code (LOC)	1,912
Code Duplication	0%
Overall Rating	A (Excellent quality)
Complexity	299 (handlers: 196)
Technical Debt	~1 day 2 hours

Table 7: Summarized quality metrics from CodeClimate analysis.

2.2 Orchestration

To streamline deployment, Docker, Docker-Compose, Docker Swarm, and Terraform were utilized.

The implementation contains two separate docker compose files, defining core services (`app`, `prometheus`, `alloy`, `loki`, `grafana`, and `database`). Each service has a corresponding Dockerfile, which details how the image is built. Some services also use custom configuration specifications, found under `/.infrastructure/` (see fig. 5).

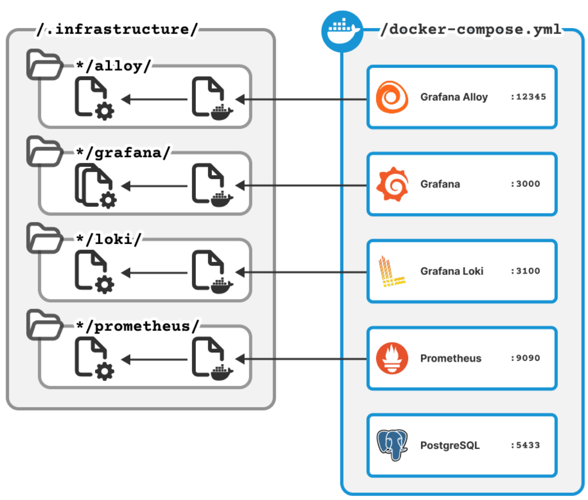

Figure 5: Informal depiction of docker services and respective configurations

- `docker-compose.yml` is used for local deployment and image publishing. It uses `localhost` and includes configurable values (with associative default values) for the system.
- `docker-compose.deploy.yml` is used for remote deployment. It builds on `docker-compose.yml` but overrides relevant configurations. This compose file contains the Docker Swarm setup-specifications, with 1 manager node and at least 1 worker node, which enables horizontal scaling.
  - The Minitwit GoLang application (`app`) runs on every worker node.
  - Metrics aggregation and monitoring services (`prometheus`, `loki`, `grafana`) runs only on the manager node.
  - OpenTelemetry Collector distribution (`alloy`) runs on all nodes.

Infrastructure-as-Code (IaC) is used to simplify the remote setup of the Swarm. Terraform files are located in `.infrastructure/infrastructure-as-code/`. Automatic deployment via Terraform is illustrated in fig. 6.

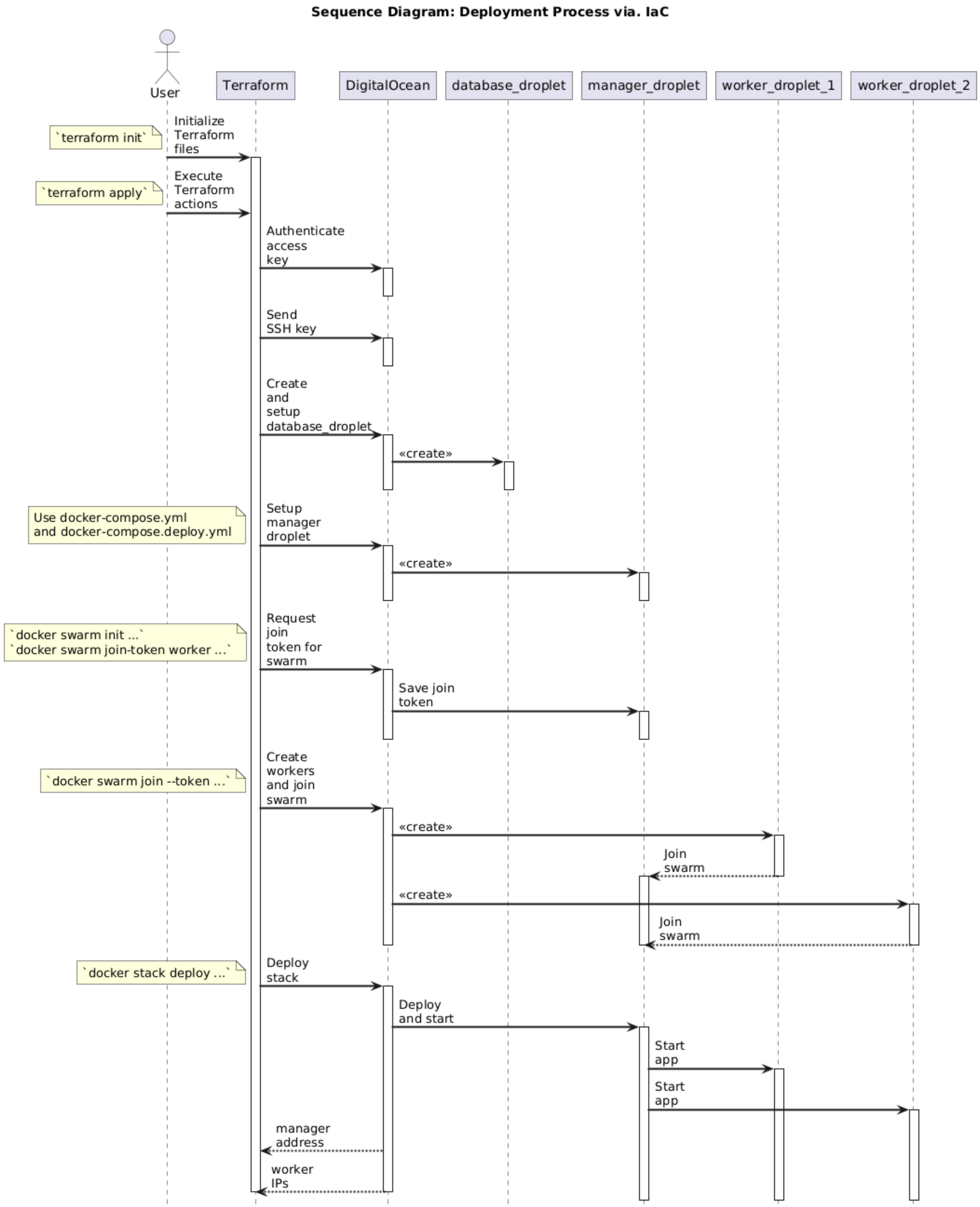

Figure 6: Sequence diagram of Terraform for IaC. Note: Terraform executes the calls to DigitalOcean sequentially, but continuous “OK” responses from DigitalOcean were omitted for brevity.

2.3 Deployment

2.3.1 Virtual Private Server (VPS)

To host the system on a remote server, DigitalOcean was chosen as the VPS provider. This choice was based on pricing (see tbl. 8), its apparent ease-of-use [12] [13] [14], and familiarity to the group through lecture demonstration.

VPS	DigitalOcean	Microsoft Azure	Oracle	AWS (Lightsail)
Virtual Machine Price	ca. $12/mo [15]	ca. $15/mo [16]	$13/mo [17]	ca. $12/mo [18]
Storage Price	50GB included [15]	ca. $5 (64GB) [16]	ca. $2.5 (50GB) [17]	ca. $12/mo [18]
Total Price	ca. $12/mo	ca. $20/mo	ca. $15.5/mo	ca. $12/mo

Table 8: Price comparison of VPS providers.

2.3.2 Infrastructure-as-Code

The Terraform setup ensures a consistent and automatic creation of infrastructure on DigitalOcean. Terraform has an easy-to-use built-in provider for DigitalOcean [19].

2.3.3 Allocation Viewpoint

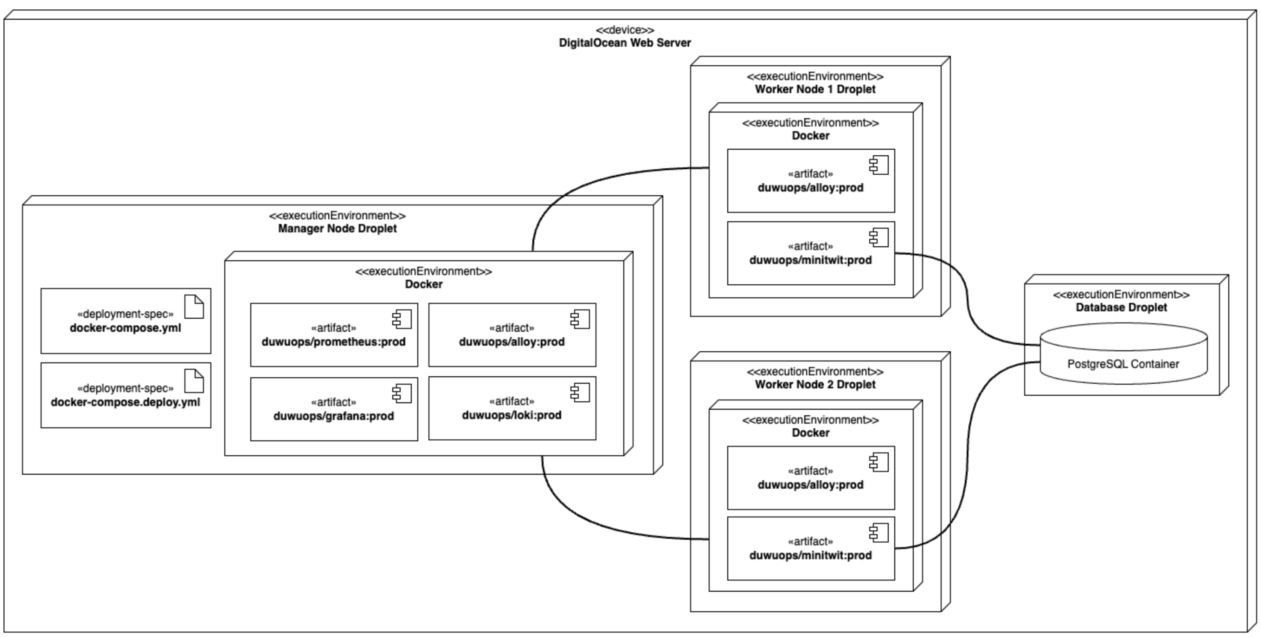

Figure 7: Deployment diagram

2.4 Database

The database runs on a separate, containerized droplet, with restricted access through firewall to ensure security and isolation between environments (see fig. 1).

PostgreSQL was chosen to replace the SQLite setup, due to strong SQL standards compliance [20], high community adoption [3], and advanced features [21], [22].

2.4.1 Choice of Technology - Database

We compared leading relational databases based on the Stack Overflow 2024 Developer Survey [3]. Only open-source, self-hosted Relational Database Management Systems (RDBMSs) were considered. The comparison is shown in tbl. 9.

Database	**SQLite**	**PostgreSQL**	**MySQL**	**Oracle**	**SQL Server**	**MariaDB**
Popularity	33.1% [3]	49.7% [3]	40.3% [3]	10.1% [3]	25.3% [3]	17.2% [3]
License	Public-Domain [23]	Open-Source [24]	Open-Source & Proprietary [25]	Proprietary	Proprietary [26]	Open-Source [27]
Standards Compliance [28]	Low [20]	Compliant [20]	Limited [20]	*Unknown*	*Unknown*	Fork of MySQL; Assumed limited
Max Connections	1	500,000+ [22]	100,000+ [22]	*Unknown*	*Unknown*	200,000+ [22]
Horizontal Scaling	No	Yes [22]	Yes [22]	*Unknown*	*Unknown*	Yes [22]
Concurrency Handling	None	Excellent [22]	Moderate [22]	*Unknown*	*Unknown*	Strong [22]

Table 9: Comparison of RDBMSs.

Note: Performance benchmarks are excluded due to license restrictions placed on benchmarking by licensing of proprietary DBMSs [29].

MySQL was ruled out due to licensing issues and development concerns post-Oracle acquisition [30], [20].

3 Process Perspective

3.1 CI/CD

GitHub Actions was chosen based on its simplicity, familiarity, and free pricing [31], [32]. A motivating factor was the suite of services supported natively in Github, of which the following were utilized:

- GitHub Action Secrets & Variables for storing ssh-keys, passwords, etc.
- GitHub Tags, Releases & Artifacts Storage for artifact versioning of the application.
- GitHub Applications for code quality evaluations with CodeClimate and SonarQubeCloud.
- GitHub Projects, Tasks & Backlog for defining and distributing tasks.

3.1.1 CI/CD Pipelines

In total, **7** pipelines are established (see tbl. 10).

File	Purpose	Invoked on
`continous-development.yml`	Primary CI/CD flow against PROD	Pushing `main`
`codeql.yml`	Analyzes go source code using `CodeQL`	Push & PRs to `main`.
`generate-report.yml`	Generates `report.pdf` from files in `/report/*`	Push to `/report/*`
`linter-workflow.yml`	Runs golangci-lint on go source code.	Push `main` or any PR
`pull-request-tests.yml`	Runs python tests.	All PRs
`test-deployment.yml`	Secondary CI/CD flow against TEST.	Tag `test-env*`
`sonarcube_analysis.yml`	Analyses go source code using SonarCloud.	PRs to `main`

Table 10: List of GitHub Actions workflows employed.

3.1.2 CI/CD Specific Technologies

- The `golangci-lint` linter is implemented in `linter-workflow.yml` (see tasks #119 and #129)
- The `pandoc` library is used to generate laTeX reports from markdown in `generate_report.yml`
- The `CodeQL` code analysis engine is used in `codeql.yml` to check for security vulnerabilities.
- Original `pytest` files are used in `continous-development.yml`–now functioning as a `Test` stage (see `minitwit_tests.py` and `sim_api_test.py`).

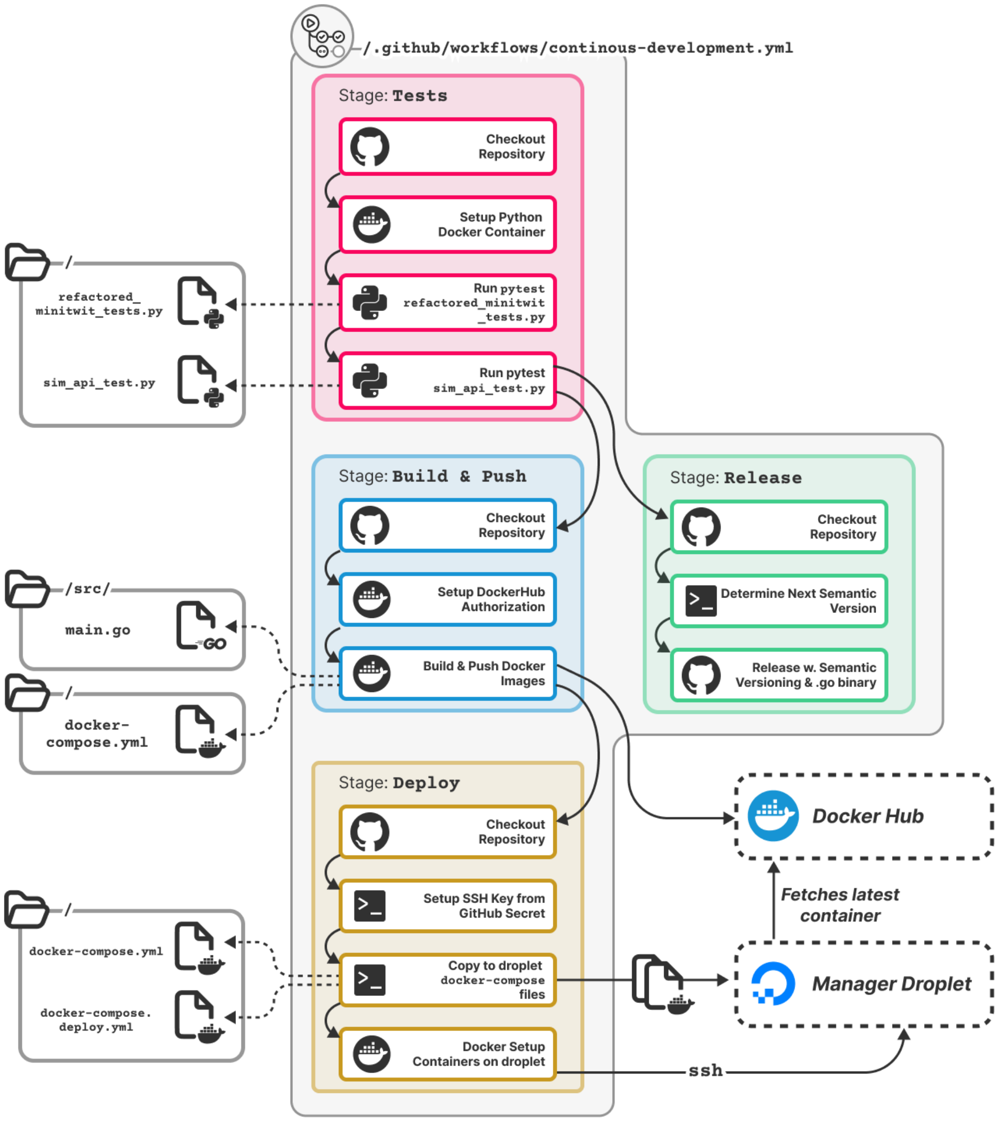

Figure 8: Informal visualization of `continous-development.yml` (primary pipline), with stages `Tests`, `Build & Push`, `Release`, and `Deploy`

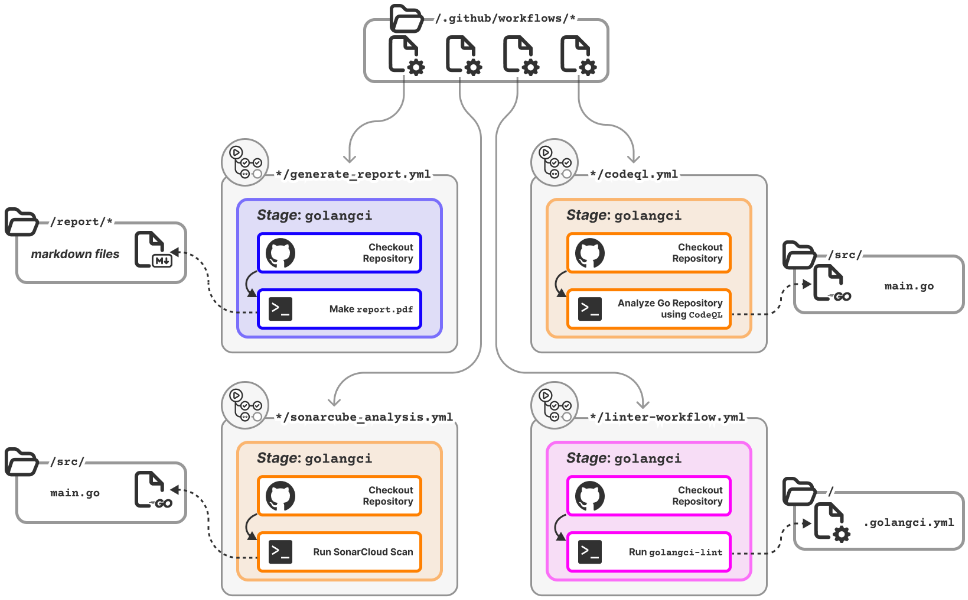

Figure 9: Informal visualization of other pipelines

3.1.3 Choice of CI/CD

A comparison of CI/CD systems was performed, and the results can be seen in tbl. 11.

- Since GitHub was chosen, GitLab CI/CD and BitBucket Pipelines were discarded, as they are specific to alternative git repository management sites.
- Commercial automation tools such as Azure DevOps and TeamCity were discarded due to pricing.

As such, the choice was between GitHub's GitHub Actions or a CI/CD system agnostic to repository management sites.

The self-hosted automation system Jenkins was considered, but the perceived learning curve along with the self-hosted infrastructure setup [32] dissuaded us from this choice, as time-to-production for *establishing* CI/CD pipelines was an important factor for us.

CI/CD Tool / Platform	GitHub Actions	Jenkins	Azure DevOps	TeamCity (JetBrains)
Ease-of-use	Simple [31]	Medium [31]	*Undetermined*	*Undetermined*
Version Control	Native GitHub Integration [32]	Agnostic [32]	Agnostic [32]	Agnostic [32]
Hosting	Primarily cloud-based [32]	Self-hosted [32]	Cloud-based [32]	Cloud-based or self-hosted [32]
Pricing Model	Free for public repositories, tiered for private [32]	Open-source (MIT License), only cost is for hosting [32]	Commercial with a limited free tier [32]	Commercial [32]

Table 11: Comparison between CI/CD systems.

3.2 Monitoring

3.2.1 Prometheus

Prometheus is used to collect and store metrics, and is invoked as a middleware service of Echo. Prometheus was chosen due to its familiarity from class, native integration with Echo [33], inferred popularity, integration with Grafana, and open-source license [34].

In our implementation, Prometheus scrapes application every 5 seconds (see `prometheus.yml`). Custom-made metrics are implemented in Echo to expose specific information from the GoLang implementation (see `src/metrics/`), these are outlined in tbl. 12

Operation	Type	Purpose
User follower	Gauge	Tracks the number of followers a user has
User followees	Gauge	Tracks the number of users a specific user is following
VM CPU usage	Gauge	Monitors real-time CPU usage on a virtual machine
Messages posted (by time)	Counter	Counts the total number of messages posted over time
Messages posted (by user)	Gauge	Tracks the message count for individual users
Messages flagged (by user)	Gauge	Tracks how many messages a user has flagged
New user	Counter	Counts the number of new users registered over time
Total users	Gauge	Tracks the total number of users in the system

Table 12: Custom-made metrics for Prometheus. **Note:** See `src/metrics/` for implementation.

3.2.2 Grafana

Grafana was chosen due to its familiarity from class, rich visualisation and open-source license. In Grafana, two users are configured: An admin user and a specific login for Helge and Mircea. When Docker Swarm was implemented, the created dashboards unfortunately became non-functional. As such, the presented pictures show what they *used to* look like (see fig. 10, fig. 11, fig. 12, fig. 13)

Figure 10: Grafana Dashboard: Whitebox Request and Response Monitoring Dashboard (Timeframe: last 30 mins)

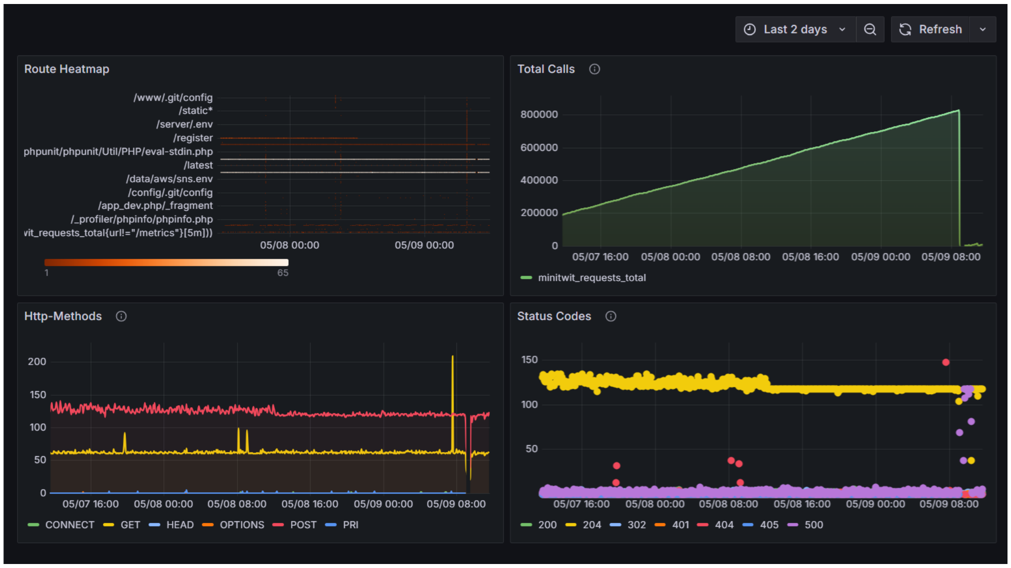

Figure 11: Grafana Dashboard: Whitebox Request and Response Monitoring Dashboard (Timeframe: last 2 days)

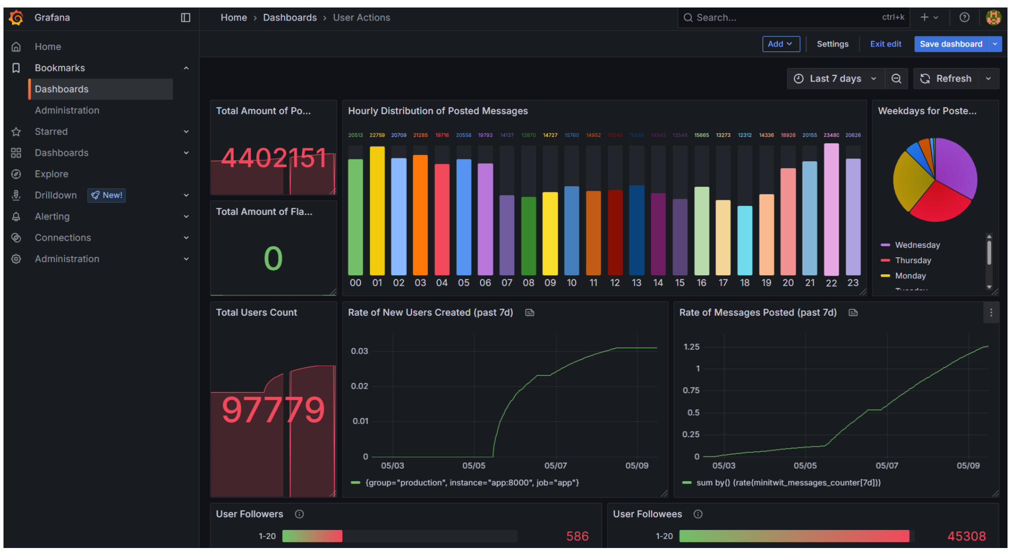

Figure 12: Grafana Dashboard: Whitebox User Action Dashboards Monitoring (Timeframe: last 7 days)

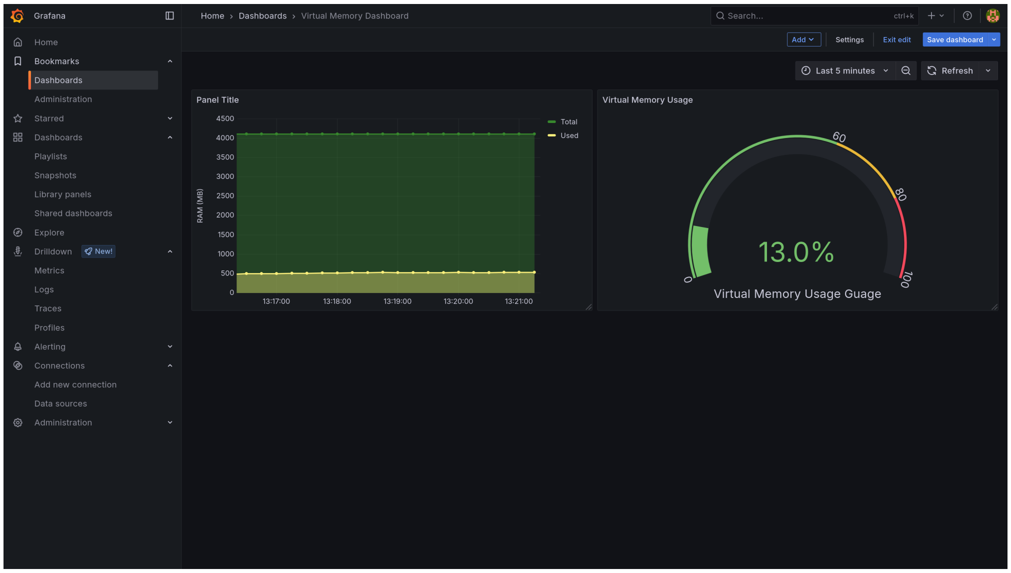

Figure 13: Grafana Dashboard: Whitebox Virtual Memory Dashboard Monitoring (Timeframe: last 5 mins)

3.2.3 Other types of monitoring

- **Black Box Monitoring:** Via the Status and Simulator API errors graf from class
- **DigitalOcean Monitoring:** DigitalOcean shows CPU usage, Bandwidth, and Disk I/O.
- **Alert System:** An alert system was set up via a cron-job that checks the web-application every 5 minutes. If the application is not up, it activates a Discord bot that sends an alert and tags everyone (see fig. 14).

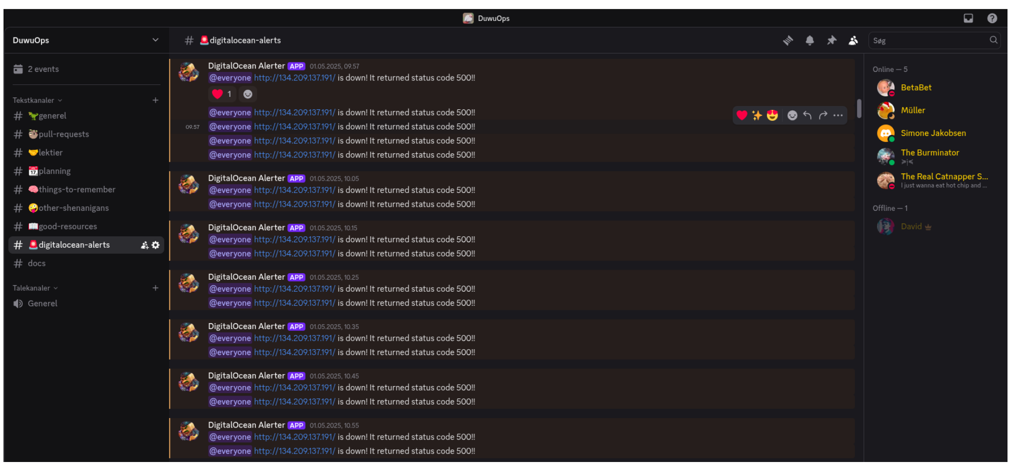

Figure 14: Discord Alert System (Bot) example

3.3 Logging

Grafana Alloy, Grafana Loki and Grafana were chosen to handle the collection, aggregation, and presentation of logs.

To ensure application log messages are usable, logs are created at different levels of severity. To ensure they are readable at a glance, emojis are used (see fig. 15)

Figure 15: Emojis used in logging for observability

Grafana Alloy collects logs by gathering data from containers on the same docker environment. The gathered logs are sent to Grafana Loki. One instance of Alloy exists on each worker node.

To ensure that logs are centralised, Loki only runs on the manager node, but collects data from all Alloy instances. The collected logs can be found via. Grafana->Drilldown->Logs.

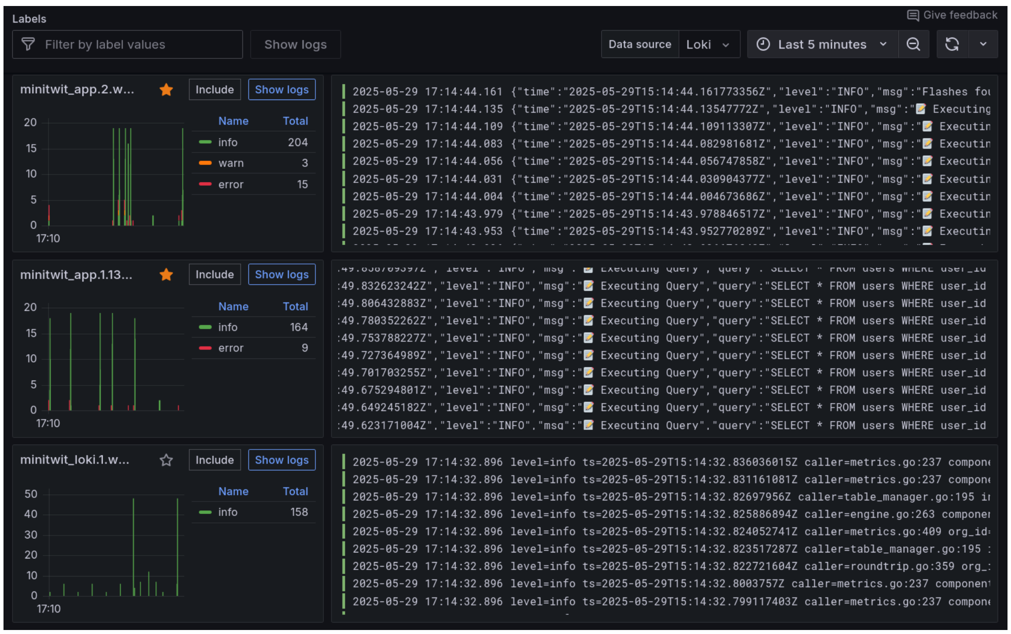

Figure 16: Logging dashboard.

Loki is configured to store logs in a folder called `tmp`. While this approach provides reliable log persistence, transitioning to an unbuffered stdout stream would be better to align with the principle that processes should not manage their own storage.

3.4 Strategy for Scaling and Upgrades

We used Docker Swarm with Docker `stack` command to reuse the already existing Docker compose configurations. However, some changes were necessary to accommodate the Docker `stack` specifications and issues related to splitting the services onto different droplets.

The changes included:

- Setting up an overlay network
- Specifying the number of replicas for each service
- Assigning monitoring services to specific droplets
- Adjusting configurations across various technologies

Docker has been configured to do rolling updates, as this is nativly supported on Docker Swarm. Additionally, Docker has been configured to rollback if a minitwit-container crashes whithin 30 seconds of deployment.

3.5 AI Use

Throughout the development process, the team used the AIs: ChatGPT, Claude, DeepSeek, and GitHub Copilot.

These were used to:

- Understand and fix code issues
- Help format and phrase code and text
- Provide inspiration during development

They were especially helpful for bug-fixing, and were credited as co-authors on relevant commits.

4 Security Assessment (May 24, 2025)

Note: This assessment was conducted before Docker Swarm was implemented and is therefore outdated. An updated security review is a high priority for future development.

4.1 Risk Identification

Our public-facing asset is a single web-app, while the database instance is protected by a firewall. To identify the attack surface, we performed a TCP SYN scan of the most common ports against web's IP. The scan revealed open ports:

- SSH (port 22)
- HTTP (port 80)
- Grafana (port 3000)
- Prometheus (port 9090)

Ports 3000 and 9090 are default monitoring services, and should be privated. Exposing SSH on port 22 is expected for maintenance access, and HTTP on port 80 is the web-app's interface and should remain open.

To uncover vulnerabilities, we used Nmap's vulnerability scripts against ports 22 and 80, which identified exposure to cross-site request forgery (CSRF) and Slowloris denial-of-service attacks. Given prior incidents of idle-connection exhaustion, the Slowloris finding was expected. A subsequent Nikto scan revealed missing security headers to prevent clickjacking and content sniffing.

4.2 Risk Scenarios

- A successful CSRF attack could trick authenticated users into unknowingly executing malicious actions.
- A Slowloris attack could exhaust server memory and trigger an OOM kill; because Docker's restart policy does not recover from OOM kills[35], the service would require manual intervention.
- Clickjacking could be achieved by embedding malicious code in transparent frames, deceiving users into performing unintended actions.
- Content sniffing attacks could exploit the browser's tendency to reclassify responses based on payload content, potentially executing embedded malicious scrips within user-submitted posts.

4.3 Risk Analysis

	Impact: Low	**Impact: Medium**	**Impact: High**
Likelihood: Low	Clickjacking		
Likelihood: Medium			Content Sniffing
Likelihood: High	CSRF		Slowloris

Table 13: Overview of likelihood and impact-level of identified scenarios.

Based on this analysis, we prioritized patches in the following order: Slowloris protection, CSRF mitigation, content-sniffing prevention, and clickjacking hardening.

4.4 Mitigation and Remediation

- Slowloris attacks: Configure Read, Write, and Idle connection timeouts on the web-server and impose limits on header size (see PR #160).
- Slowloris attacks: Enforce maximun database connections, with reduced lifetimes, to prevent resource exhaustion (see PR #160).
- CSRF: Integrate middleware that issues and validates per-request tokens for all form submissions (see PR #152).
- Content sniffing: Add response headers instructing browsers not to infer MIME types (see PR #157).
- Clickjacking: Set the `X-Frame-Options: DENY` header on all responses (see PR #157).

5 Reflections

5.1 Evolution, Operation, and Maintenance

The group applied DevOps strategies to deliver a product with fast feedback loops to improve quality in our work. This is summarized in three key areas:

5.1.1 Visible Work

- **Kanban board**: Provided an overview of work and progress (using GitHub Backlog).
- **Issue assignment**: Clarified responsibility (using GitHub Tasks).
- **Acceptance criteria**: Set clear goals for each task (using GitHub Tasks)
- **Conventions**: Standardised commits, PRs, and reviews (see `CONTRIBUTING.md`).
- **Knowledge exchange**: Ensured shared understanding across the team.

5.1.2 Minimal Branch Sizing

- Used small branches for faster reviews.
- Reduced merge conflicts and integration delays.

5.1.3 Pipeline

Pipelines helped shorten lead time by:

- **CI**: A autonomous process that provided quick feedback on code changes.
- **Security alerts**: Using a dedicated discord alert bot (see fig. 14).
- **Automatic updates**: Reduced manual maintenance.
- **Automated tests**: Ensured confidence during development and refactoring.

Bibliography

[1] Go team, "Documentation." Accessed: May 29, 2025. [Online]. Available: https://tip.golang.org/doc/?utm_source=chatgpt.com

[2] Go team, "Go developer survey 2024 H2 results." Accessed: May 29, 2025. [Online]. Available: https://go.dev/blog/survey2024-h2-results?utm_source=chatgpt.com

[3] Stack Overflow, "Stack Overflow developer survey 2024." Accessed: May 29, 2025. [Online]. Available: https://stackoverflow.blog/2024/08/06/2024-developer-survey/

[4] J. Azar, "Choosing a go framework: Gin vs. echo," Mattermost. Accessed: Mar. 20, 2025. [Online]. Available: https://mattermost.com/blog/choosing-a-go-framework-gin-vs-echo/

[5] C. Mayank, "Go: The fastest web framework in 2025," *Tech Tonic*, 2025, Accessed: Sep. 30, 1999. [Online]. Available: https://medium.com/deno-the-complete-reference/go-the-fastest-web-framework-in-2025-dfa2ddfd09e9

[6] N. Kramer, "Top 8 go web frameworks compared 2024." Accessed: Mar. 20, 2025. [Online]. Available: https://daily.dev/blog/top-8-go-web-frameworks-compared-2024

[7] Y. Israni, "Go gin vs gorilla mux: Which one is right for your web project?" Accessed: Mar. 20, 2025. [Online]. Available: https://yashisrani.hashnode.dev/go-gin-vs-gorilla-mux-which-one-is-right-for-your-web-project

[8] M. M. Shahjahan, "A deep dive into gin, chi, and mux in go," Medium. Accessed: Mar. 20, 2025. [Online]. Available: https://medium.com/@hasanshahjahan/a-deep-dive-into-gin-chi-and-mux-in-go-33b9ad861e1b

[9] B. Scheufler, "Choosing the right go web framework." Accessed: Mar. 20, 2025. [Online]. Available: https://brunoscheufler.com/blog/2019-04-26-choosing-the-right-go-web-framework

[10] Awesome Go, "Gin VS chi." Accessed: Mar. 20, 2025. [Online]. Available: https://go.libhunt.com/compare-gin-vs-chi

[11] Awesome Go, "Echo VS chi." Accessed: Mar. 20, 2025. [Online]. Available: https://go.libhunt.com/compare-echo-vs-chi

[12] C. Quinn, "Should i pick digitalocean or AWS for my next project?" Last Week in AWS. Accessed: May 28, 2025. [Online]. Available: https://www.lastweekinaws.com/blog/should-i-pick-digitalocean-or-aws-for-my-next-project/

[13] A. Amin, "Why do some people use digitalocean when other major cloud providers are so much more featureful?" Quora. Accessed: May 28, 2025. [Online]. Available: https://qr.ae/pAeHzC

[14] D. P. Finder, "Digitalocean reviews - pros & cons, ratings & features," Medium. Accessed: May 28, 2025. [Online]. Available: https://digitalproductsfinder.medium.com/digitalocean-reviews-pros-cons-ratings-features-19481335e8ae

[15] DigitalOcean, "Droplet pricing calculator." Accessed: Mar. 20, 2025. [Online]. Available: https://www.digitalocean.com/pricing/calculator

[16] Microsoft Azure, "(Azure) linux virtual machines pricing." Accessed: Mar. 20, 2025. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing

[17] Oracle, "(Oracle cloud infrastructure) cost estimator." Accessed: Mar. 20, 2025. [Online]. Available: https://www.oracle.com/cloud/costestimator.html

[18] Amazon Web Services, "Amazon lightsail pricing." Accessed: Mar. 20, 2025. [Online]. Available: https://aws.amazon.com/lightsail/pricing/

[19] M. Anicas and B. Hogan, "How to use terraform with digitalocean," *DigitalOcean*. DigitalOcean, Mar. 2022. Accessed: May 28, 2025. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-terraform-with-digitalocean

[20] Mark Drake and ostezer, "SQLite vs MySQL vs PostgreSQL: A comparison of relational database management systems." Accessed: Mar. 20, 2025. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[21] A. Regu, "MariaDB vs PostgreSQL: Detailed comparison for developers." Accessed: Mar. 20, 2025. [Online]. Available: https://blog.tooljet.ai/mariadb-vs-postgresql-a-detailed-comparison-for-developers/

[22] P. Abedinpour, "MariaDB vs MySQL vs PostgreSQL vs SQLite: A comprehensive comparison for web applications." Accessed: Mar. 20, 2025. [Online]. Available: https://medium.com/@peymaan.abedinpour/mariadb-vs-mysql-vs-postgresql-vs-sqlite-a-comprehensive-comparison-for-web-applications-0523cc3bc9d8#:~:text=PostgreSQL%20tends%20to%20perform%20better,applications%20with%20stringent%20data%20requirements.

[23] SQLite, "SQLite is public domain." Accessed: Mar. 20, 2025. [Online]. Available: https://www.sqlite.org/copyright.html

[24] The PostgreSQL Global Development Group, "PostgreSQL license." Accessed: Mar. 20, 2025. [Online]. Available: https://www.postgresql.org/about/licence/

[25] Oracle, "MySQL - commercial license." Accessed: Mar. 20, 2025. [Online]. Available: https://www.mysql.com/about/legal/licensing/oem/

[26] Microsoft, "Microsoft SQL server - commercial license." Accessed: Mar. 20, 2025. [Online]. Available: https://www.microsoft.com/en-us/licensing/product-licensing/sql-server

[27] MariaDB, "MariaDB - license." Accessed: Mar. 20, 2025. [Online]. Available: https://mariadb.com/kb/en/mariadb-licenses/

[28] B. Kelechava, "The SQL standard – ISO/IEC 9075:2023 (ANSI X3.135)." Accessed: Mar. 20, 2025. [Online]. Available: https://blog.ansi.org/sql-standard-iso-iec-9075-2023-ansi-x3-135/

[29] Oracle, "Oracle technology network license agreement." Accessed: Mar. 20, 2025. [Online]. Available: https://www.oracle.com/downloads/licenses/standard-license.html

[30] The Fedora Project, "Issue: Replace MySQL with MariaDB." Accessed: Mar. 20, 2025. [Online]. Available: https://fedoraproject.org/wiki/Features/ReplaceMySQLwithMariaDB

[31] F. Dinu, "GitHub actions vs. Jenkins: Popular CI/CD tools comparison," spacelift. Accessed: Mar. 20, 2025. [Online]. Available: https://spacelift.io/blog/github-actions-vs-jenkins

[32] S. N. Flavius Dinu, "20+ best CI/CD tools for DevOps in 2025," spacelift. Accessed: Mar. 20, 2025. [Online]. Available: https://spacelift.io/blog/ci-cd-tools#2-azure-devops

[33] Echo Community Contributors, "Prometheus middleware for echo." 2025. Available: https://echo.labstack.com/docs/middleware/prometheus

[34] Prometheus Authors, "Prometheus - monitoring system and time series database." 2012. Available: https://github.com/prometheus/prometheus

[35] "Docker docs - resource constraints." Accessed: May 29, 2025. [Online]. Available: https://docs.docker.com/engine/containers/resource_constraints/#understand-the-risks-of-running-out-of-memory