IT University Copenhagen

DevOps Exam Project

Computer Science

DevOps
MiniTwit - A Go‘ Application

Students:
Christian A. S. Mark - calm@itu.dk
Jonas T. Thomsen - jtth@itu.dk
Malthe A. Nørgaard - asno@itu.dk
Kaare Børsting - boer@itu.dk
Thomas M. Espersen - tesp@itu.dk

Class Code: KSDSESM1KU
May 31, 2022

Organization:
DevelOpsITU

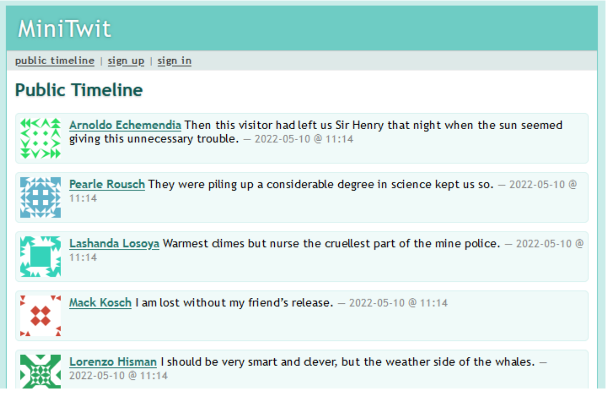
Repositories:
MiniTwit
ServerDeployment
DataScraper

Contents

1 Introduction

This report is based on a project made during the 2022 spring course of DevOps, Software Evolution and Software Maintenance, MSc with the following Course material.

The project's focus was on making software that can evolve, scale and be maintained. An old application copying the social media platform Twitter, that had not been touched for years was provided to us. From this we were to create our scalable, maintainable, and automatically deployed version, which we named MiniTwit, see Figure 1.



MiniTwit

public timeline | sign up | sign in

Public Timeline

Arnoldo Echemendia Then this visitor had left us Sir Henry that night when the sun seemed giving this unnecessary trouble. — 2022-05-10 @ 11:14

Pearle Rousch They were piling up a considerable degree in science kept us so. — 2022-05-10 @ 11:14

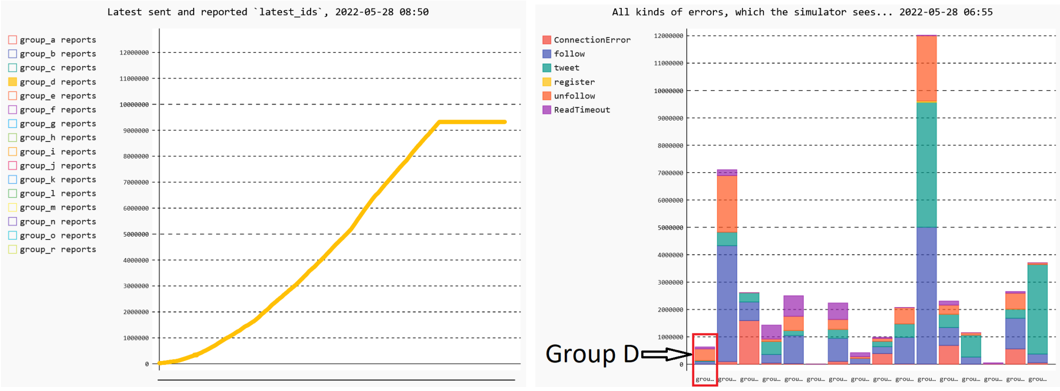
Lashanda Losoya Warmest dimes but nurse the cruellest part of the mine police. — 2022-05-10 @ 11:14

Mack Kosch I am lost without my friend's release. — 2022-05-10 @ 11:14

Lorenzo Hisman I should be very smart and clever, but the weather side of the whales. — 2022-05-10 @ 11:14

Figure 1: Front page of MiniTwit as a public user (Source: Own image)

To generate traffic to the site, the professors have created a simulator see Figure 2 which stress test the system, and simulate a scenario/environment, where downtime is punished.



(a) The flow of requests from the simulator.

(b) Errors seen from the simulator

Figure 2: Feedback loop from the simulator to the groups (Source: Simulator)

To enable discussion of the system from different views the following roles have been defined.

- **Product owners**: The group
- **Stakeholder**: The professors
- **Customers**: The simulator / users of the service.

2 System's Perspective

The system is sectioned into different services. The services are "Application", "Database", "Monitoring/Logging" and "Loadbalancer". The split is done to make each of the services scaleable and maintainable independently of each other [3].

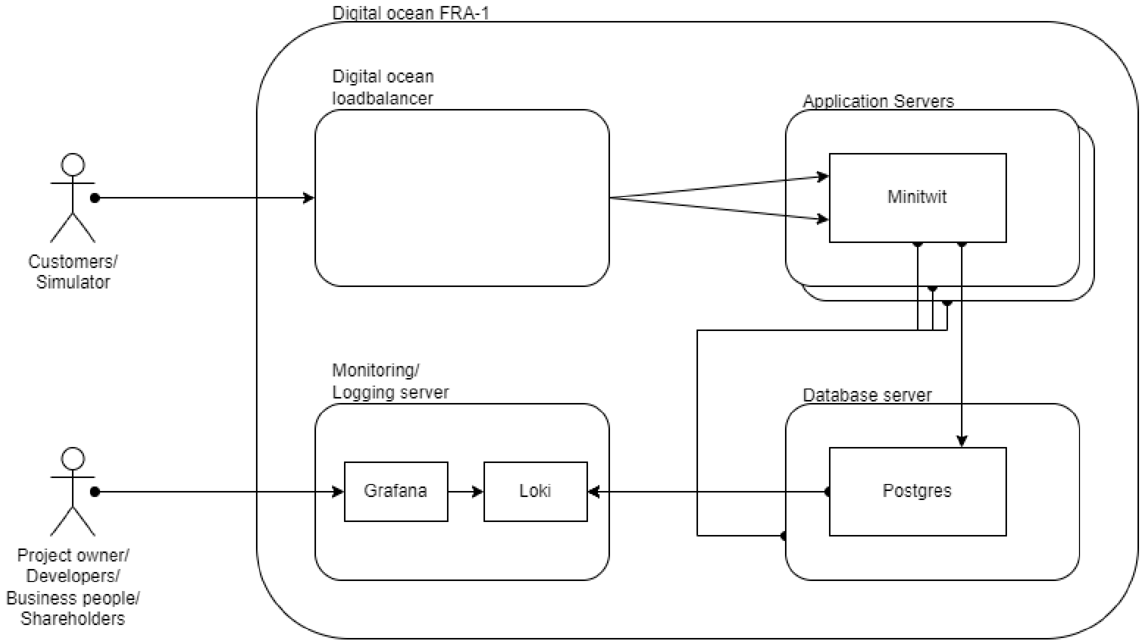

Digital ocean FRA-1
Digital ocean loadbalancer
Application Servers
Minitwit
Customers/ Simulator
Monitoring/ Logging server
Database server
Grafana
Loki
Postgres
Project owner/ Developers/ Business people/ Shareholders

Figure 3: Image of server deployment

To access each of the services and enable the option to use Transport Layer Security (TLS), there have been configured Domain Name System (DNS) records to each of the services, with subdomains to the domain "thomsen-it.dk". The configuration can be seen inTable 1. The DNS records point to floating IPs that the group can reassign through Digital Ocean web interface or API.

Domain	Server(s)
thomsen-it.dk www.thomsen-it.dk	Managed Loadbalancer
database.thomsen-it.dk	Database Server
grafana.thomsen-it.dk prometheus.thomsen-it.dk monitoring.thomsen-it.dk	Monitoring Server

Table 1: DNS record overview

All the servers have been configured to redirect traffic from port 80 to port 443 and the Nginx reverse proxy on the servers terminates TLS traffic [17].

2.1 Design and architecture of ITU-MiniTwit system

The code follows the onion architecture pattern seen in Figure 4. This pattern has been chosen to easily separate segments of the code.

The presentation layer accepts the incoming REST web requests and translates them to method invocations to the application layer. The application layer calls the domain layer and makes sure that the business logic and rules are kept and maintained for each call to

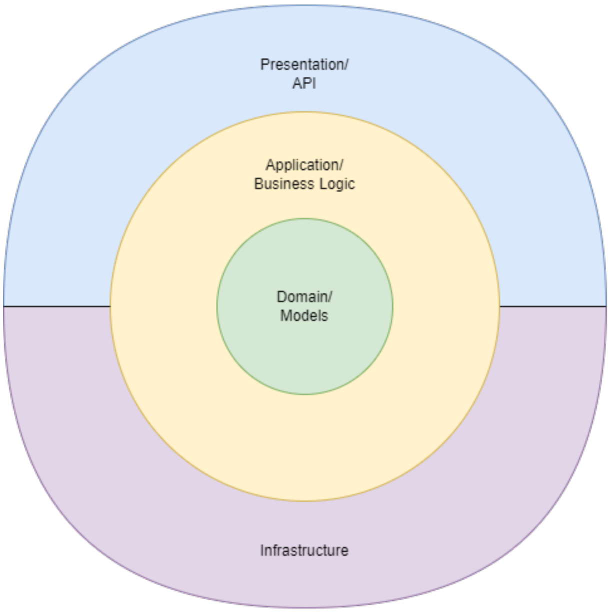


Figure 4: Onion architecture

the database layer and returned to the presentation layer. The communication between the different layers are depicted in Figure 5.

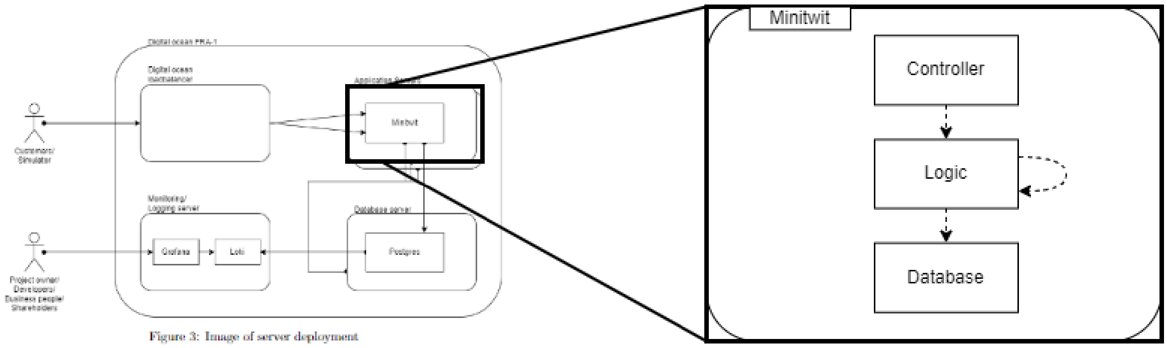


Figure 5: System design of Minitwit Application

2.2 Dependencies of the ITU-MiniTwit system

One of the first important choices were choosing a programming language and suitable web framework as can be seen documented in #33. Here the language Golang and web framework Gin were selected. For package management in Golang, the "go.mod" file contains all the packages that the project needs, and any indirect packages. These are maintained by Go's own package manager. A dependency graph can be seen in Appendix C. This only shows the upper mountain of our dependencies, each package has many deep dependencies. An example of these is given in the same Appendix C, showing the dependencies of the Gin package.

For docker images, the group has based the Go application on an official docker image called "golang:bullseye", which is maintained by the "Docker community"[18]. The rest of the docker image dependencies are maintained by the respective project maintainers and are: "postgres:13.5, prom/prometheus:v2.33.5, grafana/grafana:8.4.3, fluent/fluent-bit:1.9.0-debug, nginx:1.15-alpine, grafana/loki:2.4.2", which can be seen in the "docker-compose.yaml"[6] file.

For hosting, provisioning and management of VM's, the group choose to use Vagrant and the hosting provider DigitalOcean (DO), which both offers services like docker registry, -databases, -Kubernetes and managed apps like Grafana for visualization. None of these services were used however, since the group saw it against the idea of avoiding "Vendor-locking" and makes "Infrastructure-As-Code" harder to make vendor independent. However the group did give in, when the task of scaling came and setting up a managed load balancer just seemed easier to do#174.

Lastly the group depend on Github both for version control and Github Action for Continuous Integration (CI) and pushing to Dockerhub.

2.3 Important interactions of subsystems

Figure 6 shows the interactions between different services, the developer, and the customer. Code is developed on individual developers' computers and is pushed and pulled from and to the different GitHub repositories. From a configured CI pipeline code is pulled from the Minitwit repository, tested and build into docker images, uploaded to Docker Hub. The developer manually runs scripts from the ServerDeployment repository to update the droplets in Digital Ocean.
The customer accesses the application through a managed load balancer that sends the customer to either a running application droplet, which each has a reverse proxy, that finally sends traffic to the minitwit application.
The monitoring server interacts with the other droplets, by pulling information from them.

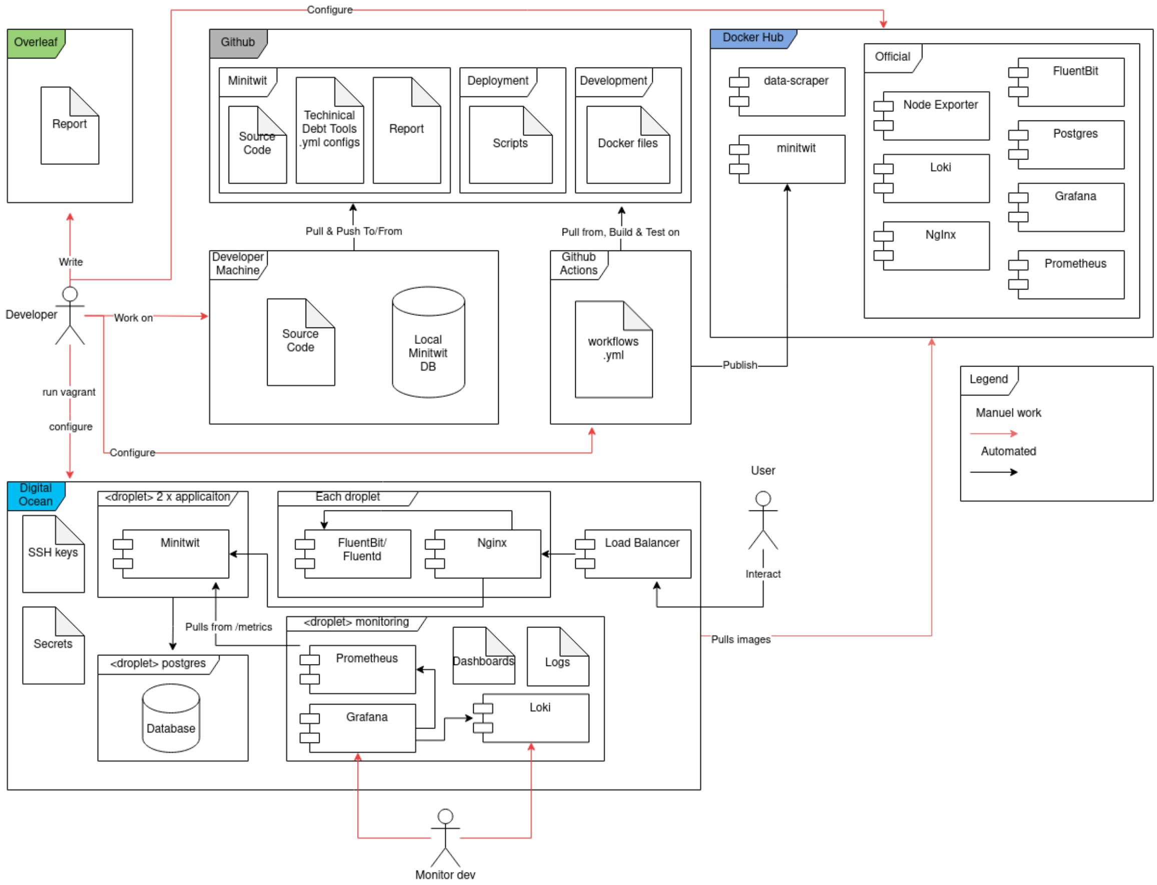
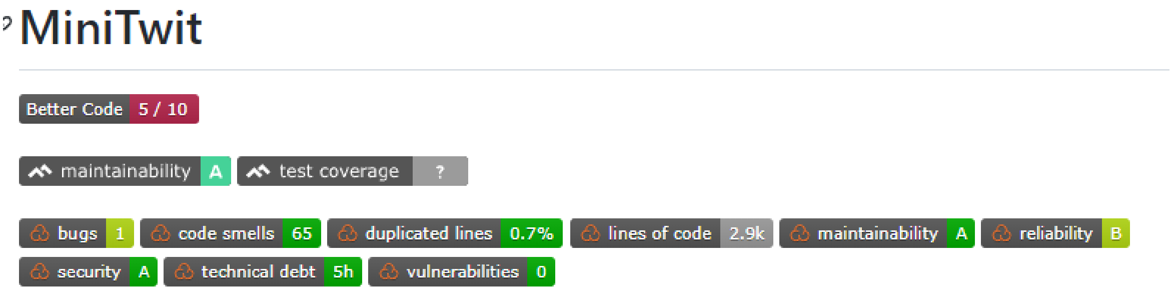


Figure 6: Interactions between services

2.4 Current state of the ITU-MiniTwit system

The current state of the system shown by the static analysis and quality assessment tools can be seen in Figure 7



MiniTwit

Better Code 5 / 10

maintainability A
test coverage ?

bugs 1
code smells 65
duplicated lines 0.7%
lines of code 2.9k
maintainability A
reliability B
security A
technical debt 5h
vulnerabilities 0

Figure 7: Code Quality of Minitiwit Application (Source: own image)

From these quality metrics, it can be seen that the results from SonarCloud returns mostly positive quality, with a bunch of code smells. However, from these code smells only 24 are critical, yet these are not of great importance.
Compared to SonarCloud, Better Code results in a much lower score due to several reasons including too many lines in single functions, code duplication and low test coverage.

2.5 Chosen project license

In order to decide on a project licence, the group had to audit whether it would be compatible with the licenses in our direct dependencies. In our case, these dependencies were Go binaries and it was difficult to find a general-purpose licence scanner for our project (for example, the tool ”scan-code”, which was proposed by our guest lecturer, did not work in our case). However, after doing some research, the group decided that ”Lichen” would be the best option (the group also tried out ”GoLicence”, but couldn’t make it work). As can be seen in issue #113, the group used ”Lichen” to count the number of different direct licences our Go project contained. In total, the project contained 31 MIT, 6 Apache -2.0, 5 BSD-3-Clause and finally one BSD-2-Clause licence. Consequently, the group choose an Apache-2.0 licence, since this is compatible with MIT, BSD-2 and 3-Clause. This decision stems from the fact that the project include Apache-2.0 licences and since Apache-2.0 is less permissive than MIT, BSD-2 and 3-Clause, the license needs to match the Apache-2.0 licence, which is more restrictive than the three aforementioned licences. As a final note, the group also tried other scanning tools for licence detection, but these lie outside the scope of this chapter.

3 Process' perspective

3.1 Interaction as developers

Communication and interaction between team members have been moderately split between informal, formal, synchronous and asynchronous communication throughout the entire course. Initially Microsoft Teams and Github issues were used. As the group later experienced Microsoft Teams to not be sufficient and fitting for the team, the group moved to using Discord instead of Microsoft Teams.

3.2 Team organisation

The team is organised with all members having equal authority. Project scope and project requirements are set by professors Helge and Mircea, as well as extra needs requested by the group.

3.3 Description of CI/CD chains

GitHub Actions is used as CI. TravisCI was initially used, since this was well-documented and also separated the deployment from the codebase by not having everything at GitHub (see issue #58). However, as the free plan on TravisCI expired, the CI chain was moved to GitHub actions, as developers were familiar with this, as well as GitHub actions providing a faster pipeline (mainly spinning up VM's) (see issue #117).

3.3.1 CI Chain

The CI chain is set up according to the operation on branches (see issues #69 and #117), such as seen in Figure 8:

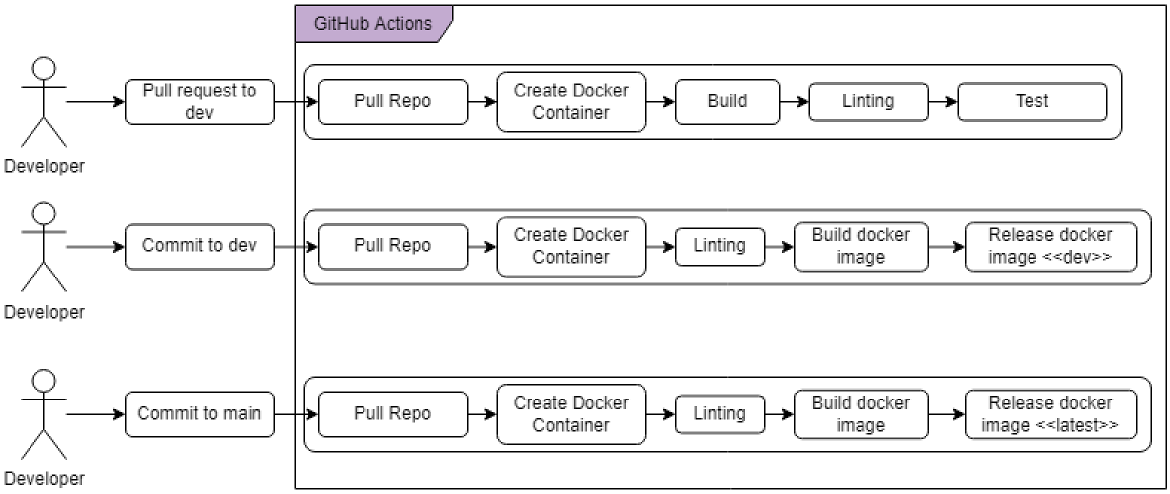


Figure 8: Different GitHub actions

An example of the "pull request to dev" CI run can be seen on GitHub [1]. The Linting tools used are Go lint[10] for linting the application code and Shellcheck [27] for linting all scripts. After Linting and testing, scancode [25] scans for licenses, however, the result is not actually used, since it has been commented out in the yml file[24]. Another commented out "job" is the docker image scan, which is described in #117. For the other chains, if everything passes, a docker image is built, linted using hadolint[14] and released.

Besides these GitHub actions, SonarCloud is set up to run static analysis of the changes, and describe if code has become worse with the pull-request. Git Guardian, also checks for vulnerabilities, leaked passwords etc.

3.4 Organisation of repositories

The project is split up into three repositories.[2]

- **Minitwit:** Main application
- **ServerDeployment:** Files for setup and maintenance of server applications
- **DataScraper:** Application that scrapes the simulator status page

3.5 Applied branching strategy

main	used as base branch, from which releases are created. this must always be buildable and runnable.
dev	used for development. this must always be buildable and runnable. every branch (except main) is derived from this branch.
*	branches derived from dev have the following templates, depending on the issue's topic: - features/#(issueid)-(issue_title / relevant_title) - bugs/#(issueid)-(issue_title / relevant_title)

Table 2: branching strategy [2]

An example of the branching strategy applied, can be seen onFigure 9, where the green, purple and yellow all are feature branches. They are all branched out from the dev branch, and any merge conflicts should be handled in the feature branch, before doing a pull request back to dev.

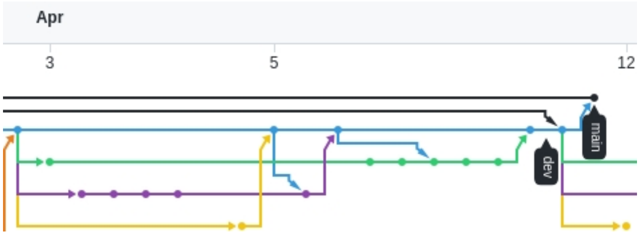


Figure 9: Github network flow graph

3.6 Applied development process

For issues, GitHub's own issue management is used. Most issues are created and managed in *MiniTwit* repository for simplified structure and readability. Issues are then assigned to individual members, who are responsible for the issue (with assistance from other team members).
Commits are expected to happen regularly and to be descriptive. The commits' subject is limited to 50 chars, with few exceptions. Large commits are to be described in bulletpoints. Pull-requests are to not be reviewed by the author, but by another team member. [2]

3.7 Monitoring of MiniTwit

To monitor MiniTwit the visualisation tool Grafana is used. Grafana can pull data from various sources and display it on various dashboards. The reasons for choosing Grafana can be found in issue #100 and is summarised below:

- Free
- Previous experience
- Extensive documentation
- Integrates well with many different data sources
- Works well with Loki subsection 3.8

To gather data from different services Prometheus has been deployed. Prometheus is a tool that pulls data from services that offer a /metrics endpoint and serves the data as a time-series for clients like Grafana. The reason for our choice of Prometheus is described in issue #99, and is summarised below:

- Free
- Previous experience
- Extensive documentation
- Many library Application Programming Interface (API) integrations
- Integrates well with Grafana

The process and flow of data can be seen inFigure 10. The users query Grafana which then queries Prometheus, PostgreSQL or both. Prometheus continuously pulls data from the different servers and applications, which provide a metric endpoint.

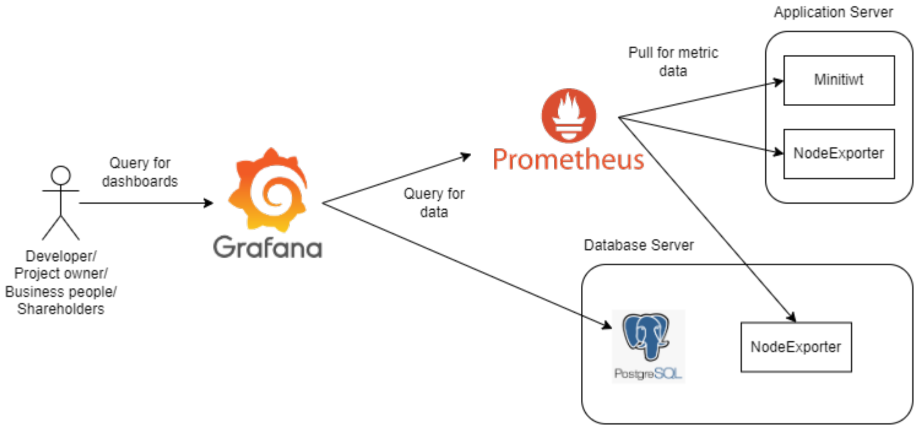


Figure 10: Grafana and Prometheus data sourcing process. (Source: own image)

Monitoring is used to provide information about: Business Intelligence, Endpoints, Server instances, and the Simulator.

All dashboards can be seen inAppendix A, with associated descriptions of panels. Grafana contains the following dashboards:

- **Business intelligence (Figure 14):** About how the product is evolving.
- **Endpoints (Figure 15):** Performance measures of response times and return codes.
- **NodeExporter (Figure 16):** CPU, RAM, Disc, Network usage of hosts/instances.
- **Simulator (Figure 17):** Error rates scraped from the simulator.
- **Heatmaps (Figure 18):** Time of day and week that service is used.

3.8 Logging of MiniTwit

To move logs from different hosts to a centralised server the group uses fluentd[8]/fluentbit[7]. To store the logs and cross-reference them the team chose the service Loki [12]. Loki is developed by Grafana Labs and has a decent amount of documentation [13]. This results in the structure and design seen in Figure 11.

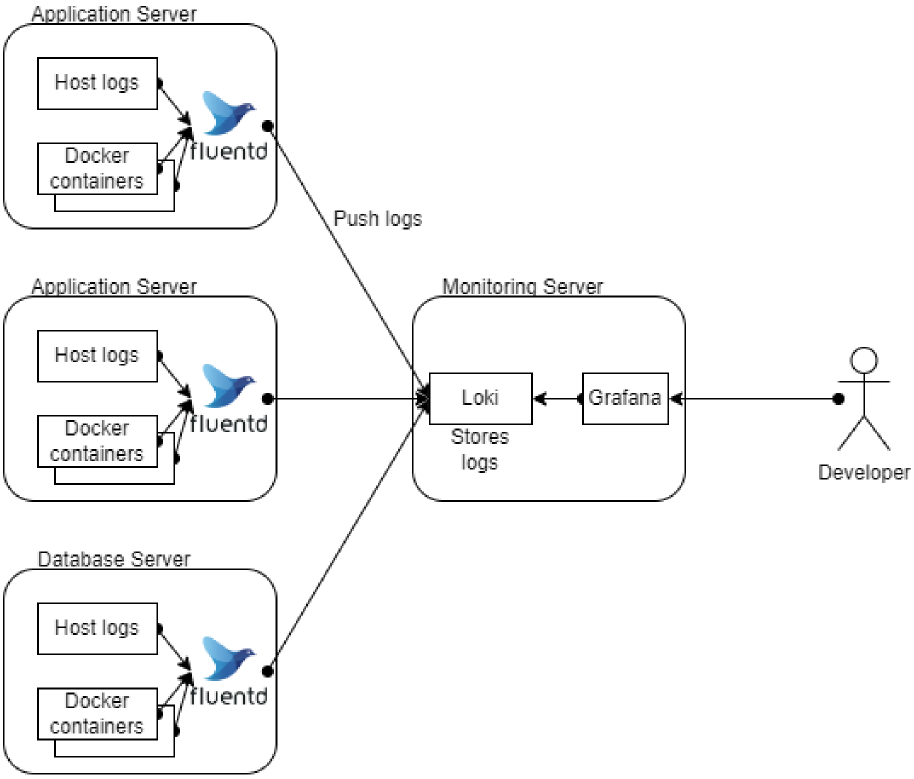


Figure 11: Flow of logs in the system

To better index and search in the logs from the application, structured logging was applied to the application #90 which can be seen in Listing 1.

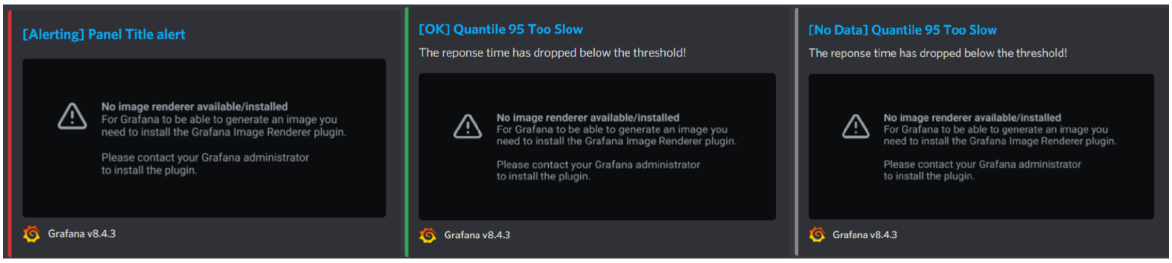
```
{
    "level":"info",
    "time":"2022-03-04T16:48:07Z",
    "message":"Starting: MiniTwit: application: startup: checks"
}
```

Listing 1: Example log of the MiniTwit application

The logging structure of the application can be seen in Figure 20 and Appendix B. Figure 21 shows the view from the log browser in Grafana.

3.9 Alerting

As a combination with the monitoring system, alerts are set up via Discord to notify the group about severe changes in data. Currently only disk space, ram usage, response time and significant increase in errors are setup with alerts (see issue #150 and also visible in Figure 19 in appendix A. When set thresholds were exceeded, when no data was received, and when metrics fell back into the acceptable state, alerts were sent on Discord. The Discord alerts can be seen in Figure 12



[Alerting] Panel Title alert

No image renderer available/installed
For Grafana to be able to generate an image you need to install the Grafana Image Renderer plugin.

Please contact your Grafana administrator to install the plugin.

Grafana v8.4.3

[OK] Quantile 95 Too Slow

The reponse time has dropped below the threshold!

No image renderer available/installed
For Grafana to be able to generate an image you need to install the Grafana Image Renderer plugin.

Please contact your Grafana administrator to install the plugin.

Grafana v8.4.3

[No Data] Quantile 95 Too Slow

The reponse time has dropped below the threshold!

No image renderer available/installed
For Grafana to be able to generate an image you need to install the Grafana Image Renderer plugin.

Please contact your Grafana administrator to install the plugin.

Grafana v8.4.3

Figure 12: Discord alert messages.

3.10 Security assessment

In lecture 9, the group were tasked with doing a security assessment with pen-testing tools, which resulted in the work described in Appendix D and in the Minitwit Github repo's wiki.[16] The result indicates that the group still has some work to do on the security of the application, Postgres server setup and a possible change of the development workflow.

3.11 Applied strategy for scaling and load balancing

The application server is scaled by having two application servers hosted. To manage these two, a load balancer is setup up to direct traffic between them (see issue #175.
The use of load balancer and two application servers also serves the purpose of blue green deployment, such that one server at a time can be taken down, updated and put back online. This guarantees that at least one server will be active at all times [28]. The blue green deployment for our system is depicted in figure 13.

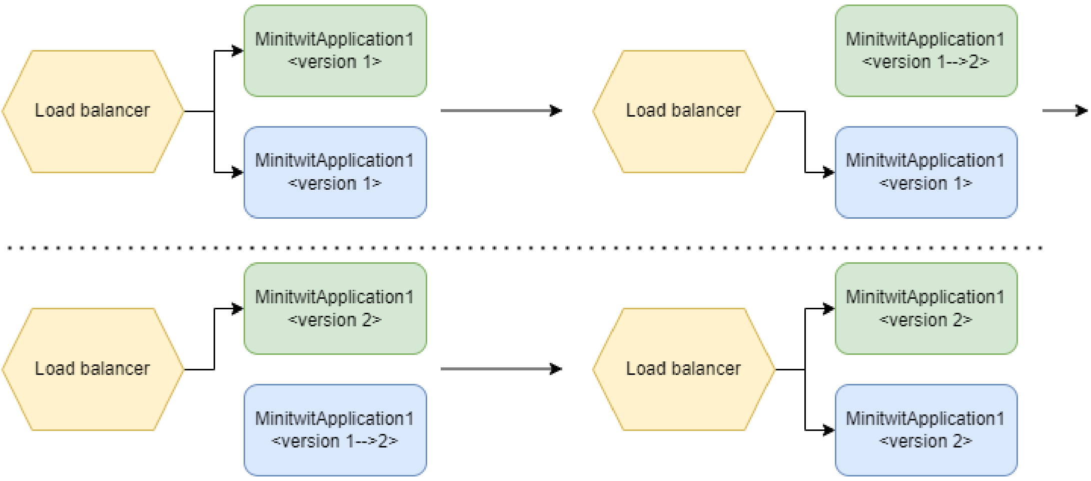

Load balancer
MinitwitApplication1 <version 1>
MinitwitApplication1 <version 1>
Load balancer
MinitwitApplication1 <version 1-->2>
MinitwitApplication1 <version 1>
Load balancer
MinitwitApplication1 <version 2>
MinitwitApplication1 <version 1-->2>
Load balancer
MinitwitApplication1 <version 2>
MinitwitApplication1 <version 2>

Figure 13: Blue green deployment with load balancer

4 Lessons learned perspective

Throughout the project, the group was faced with many difficulties requiring careful considerations, due to our simulated users, needing uninterrupted access to the application.

4.1 Migration of our database

Our largest DevOps challenge throughout the project, was the necessary migration of our database from SQLite to Postgres Our docker container repeatedly crashed due to a memory leak caused by SQLite, sometimes causing everything to disappear, including the current SQLite database file.

The group talked from the beginning of the project about separating the database and introducing a database abstraction layer (see issue #37). However, had not done so, before the simulator had started. Very concerning was the discovery, that all passwords could not be brought over due to them being binary data (BLOB), which has no direct translation into Postgres' types, also discussed in the issue regarding the migration (see issue #92).

The group found no way off translating the BLOB data, and decided to replace all passwords with the same working hash, so the group could move all of the database data. In a real life scenario the group would have decided to ask all users to change their password, which would result in all newly updated passwords to be created and stored compatible with the new database scheme.

4.1.1 Learnings

From the group's unfortunate migration problems, the group learned to never put any database into production before having setup an ORM framework to help make sure a transition between database systems is easy. In a casual project the group would have deleted everything and started over in a fresh new database system with fresh empty tables. However, due to the DevOps part of the course, it became important to the group, to successfully move all the data, and try to have as little downtime as possible.

For the migration the group used the tool PGLoader[22], although in future attempts the group may try something different, e.g. many database systems can save and load database tables as .csv files. This was an idea given by another group later in the course.

4.2 Request latency

Towards the end of the project, the group noticed an unusual long request latency for the front page of the application. It was concluded that the bottleneck was the database (see issue #182). Either an odd query or the lack of an index could cause a slow lookup, causing the request latency.
For future projects, especially ones with a lot of data. More effort will be put into queries and tables in the database, to make sure it scales.

4.3 Memory devouring monitoring server

At the end of the project another maintenance issue was observed. A constant crashing monitoring server. The monitoring server crashes constantly presumably due to missing memory in the container. In future projects, the group would ensure monitoring/logging services are provided more memory, due to a lot of data being held in memory when querying for these services. Additionally, the group did not have much filtering of logging and no set schedule and plan of storing/rotating the logging information, which led to the memory continuously increasing.

References

[1] *CI run - example.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/runs/5634305253?check_suite_focus=true.

[2] *Contributing.md.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/main/CONTRIBUTING.md.

[3] *Deployment.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/tree/docs/Docs/Deployment.

[4] *DigitalOcean is now a GitHub secret scanning partner.* URL: https://github.blog/changelog/2022-05-13-digitalocean-is-now-a-github-secret-scanning-partner/.

[5] *Discord - Four steps to a super safe server.* 2022. URL: https://discord.com/safety/360043653152-Four-steps-to-a-super-safe-server.

[6] *docker-compose file in Minitwit repo.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/main/docker-compose.yaml.

[7] *Fluent Bit.* URL: https://fluentbit.io/.

[8] *Fluentd — Open Source Data Collector.* URL: https://www.fluentd.org/.

[9] *gin-csrf middleware.* 2022. URL: https://github.com/utrack/gin-csrf.

[10] *golinter.sh.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/7d425f8a7e31dde29071c
scripts/golinter.sh.

[11] *GORM - ORM for Golang.* 2022. URL: https://gorm.io/index.html.

[12] *Grafana Loki.* 2022. URL: https://grafana.com/oss/loki/.

[13] *Grafana Loki Documentation.* 2022. URL: https://grafana.com/docs/loki/latest/.

[14] *hadolint (docker linting).* 2022. URL: https://github.com/hadolint/hadolint.

[15] *Information gathered from various security assesment tools.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/wiki/Information-gathered-from-various-security-assesment-tools.

[16] *Information gathered from various security assesment tools.* URL: https://github.com/DevelOpsITU/MiniTwit/wiki/Information-gathered-from-various-security-assesment-tools.

[17] *Nginx config.* 2022. URL: https://github.com/DevelOpsITU/ServerDeployment/blob/main/ApplicationServer/data/nginx/nginx.conf.

[18] *Official go docker image.* 2022. URL: https://hub.docker.com/%5C_/golang?tab=tags.

[19] *owasp - Path traversal.* 2022. URL: https://owasp.org/www-community/attacks/Path_Traversal.

[20] *owasp - SQL injection.* 2022. URL: https://owasp.org/www-community/attacks/SQL_Injection.

[21] *OWASP Dos Cheatsheet.* URL: https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html.

[22] *PGLoader.* URL: https://github.com/dimitri/pgloader.

[23] *Postgres Authentication methods.* URL: https://www.postgresql.org/docs/8.3/auth-methods.html.

[24] *pullreq_to_dev.yml.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/main/.github/workflows/pullreq_to_dev.yml.

[25] *Scancode - Dockerfile.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/main/scripts/scancode/Dockerfile.

[26] *Setting up a pre-commit git hook with GitGuardian Shield.* URL: https://blog.gitguardian.com/setting-up-a-pre-commit-git-hook-with-gitguardian-shield-to-scan-for-secrets/.

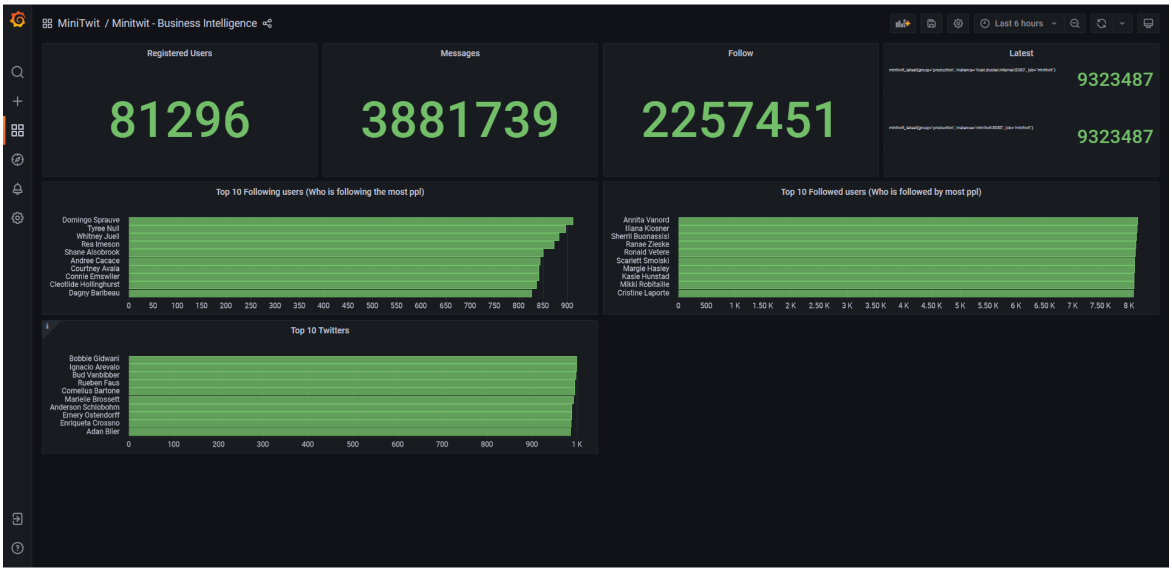
[27] *shellchecker.sh.* 2022. URL: https://github.com/DevelOpsITU/MiniTwit/blob/main/scripts/shellchecker.sh.

[28] *What is blue green deployment?* 2019. URL: https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment.

A Grafana Dashboards

Business Intelligence This dashboard shows:

- How many users are registered to the platform
- How many messages have been posted
- Who follows the most people
- Who are followed by the most people
- Who have posted the most messages.
- What request id are the latest from the simulator



MiniTwit / Minitwit - Business Intelligence

Last 6 hours

Registered Users: 81296

Messages: 3881739

Follow: 2257451

Latest

minitwit_latest{group="production", instance="host.docker.internal:8080", job="minitwit"} 9323487

minitwit_latest{group="production", instance="minitwit:8080", job="minitwit"} 9323487

Figure 14: Business Intelligence - Grafana (Source: Own image)

Endpoints This dashboard shows:

- Different endpoints and their response times
- The summery of different response codes and distributions.

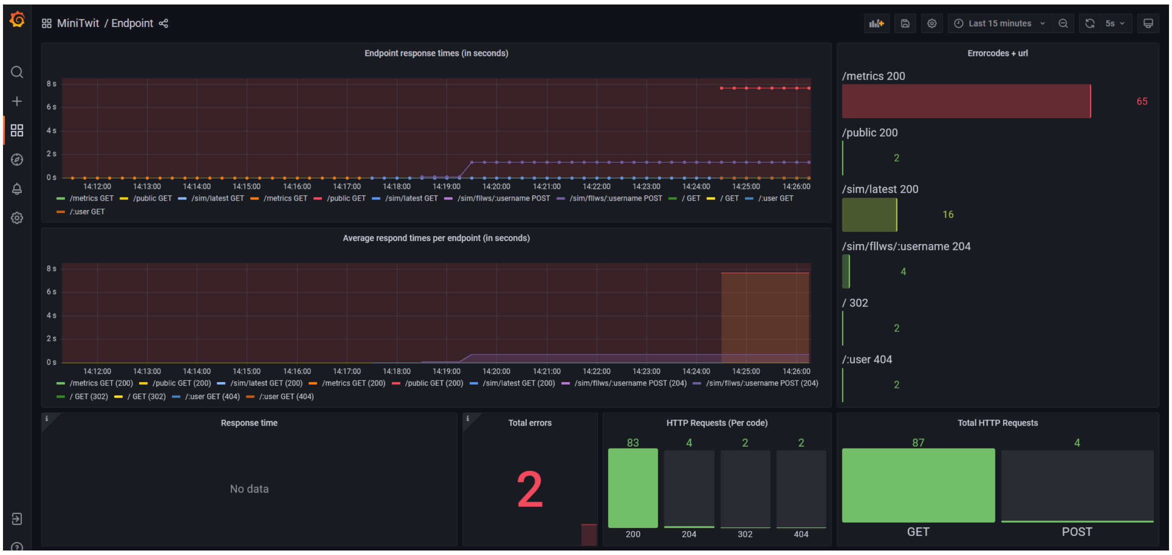
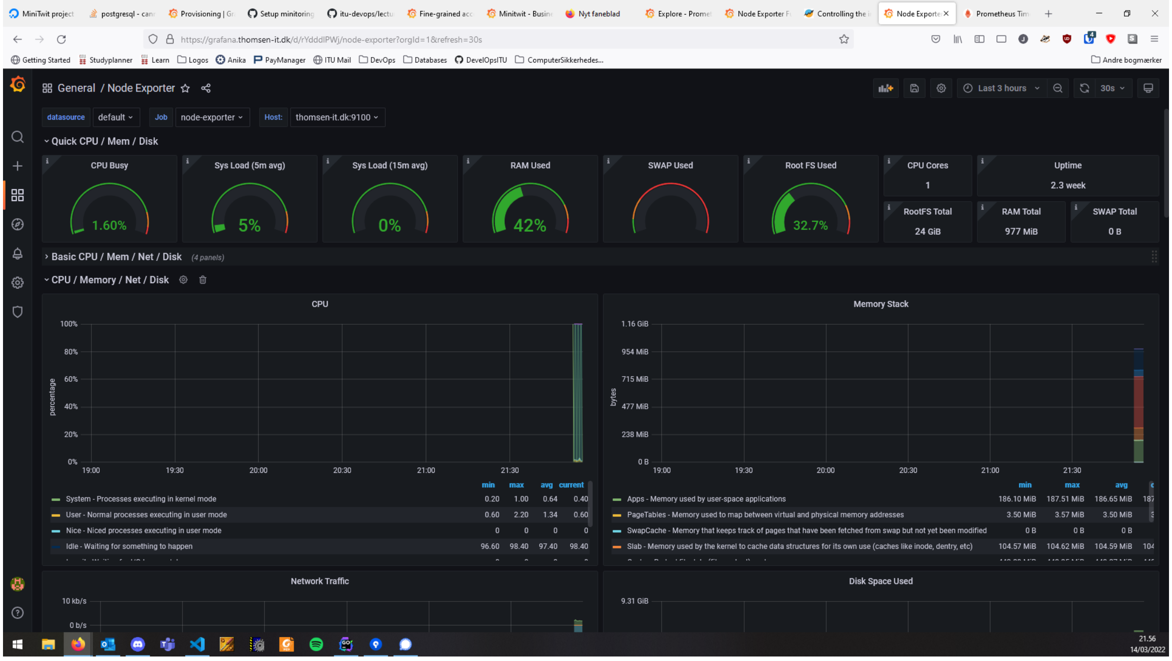


Figure 15: Endpoints - Grafana (Source: Own image)

Hosts/Instances This dashboard shows:

- Average (CPU, RAM, IO, Network) consumption.
- Very detailed information about stats of the Linux host machine.



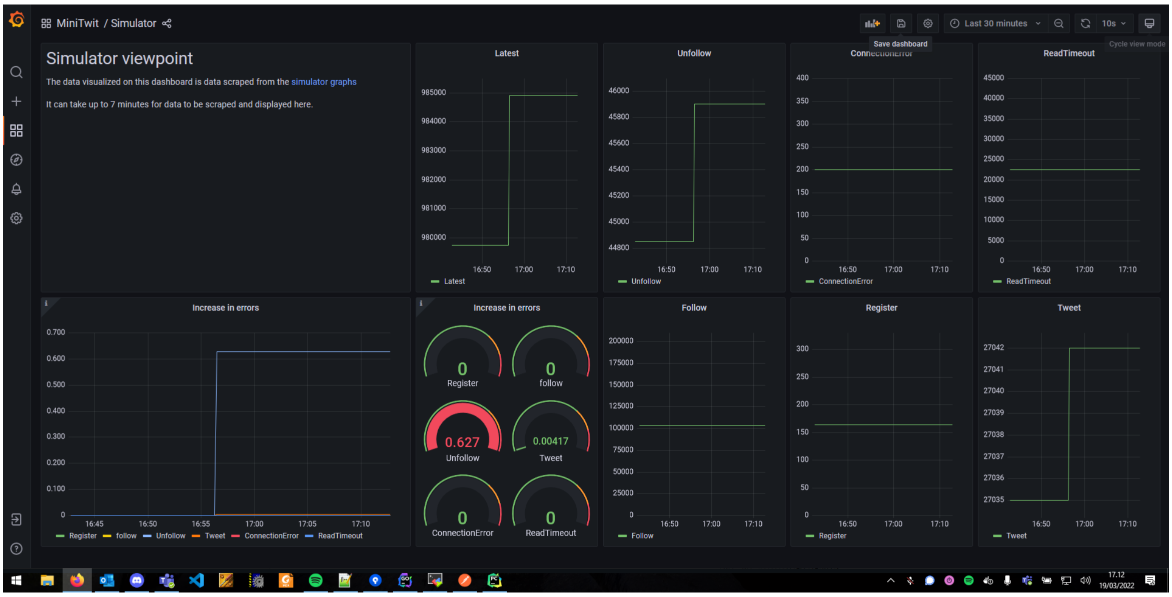
	min	max	avg	current
System - Processes executing in kernel mode	0.20	1.00	0.64	0.40
User - Normal processes executing in user mode	0.60	2.20	1.34	0.60
Nice - Niced processes executing in user mode	0	0	0	0
Idle - Waiting for something to happen	96.60	98.40	97.40	98.40

	min	max	avg
Apps - Memory used by user-space applications	186.10 MiB	187.51 MiB	186.65 MiB
PageTables - Memory used to map between virtual and physical memory addresses	3.50 MiB	3.57 MiB	3.50 MiB
SwapCache - Memory that keeps track of pages that have been fetched from swap but not yet been modified	0 B	0 B	0 B
Slab - Memory used by the kernel to cache data structures for its own use (caches like inode, dentry, etc)	104.57 MiB	104.62 MiB	104.59 MiB

Figure 16: Node Exporter - Grafana (Source: Own image)

Simulator This dashboard shows:

- Errors that the simulator sees.



MiniTwit / Simulator

Simulator viewpoint

The data visualized on this dashboard is data scraped from the simulator graphs

It can take up to 7 minutes for data to be scraped and displayed here.

Figure 17: Simulator - Grafana (Source: Own image)

Heatmaps This dashboard shows:

- Heatmaps of error, ok and total request rate.

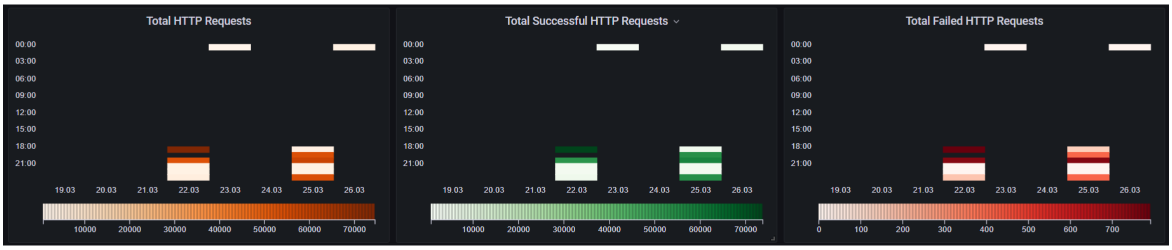


Figure 18: Heatmaps - Grafana (Source: Own image)

Alerts This dashboard shows:

- Alert activity

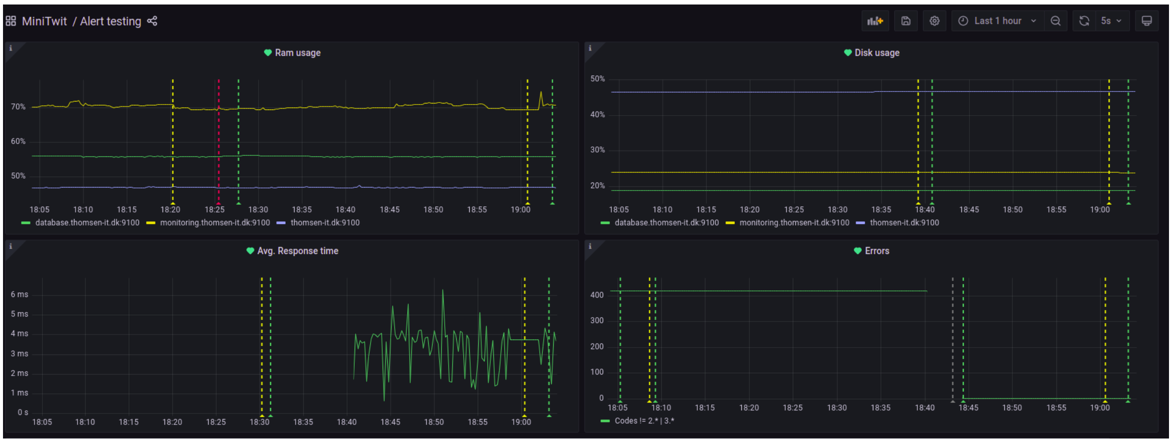
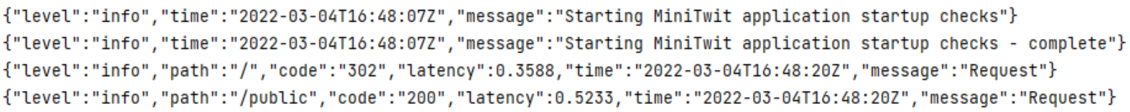


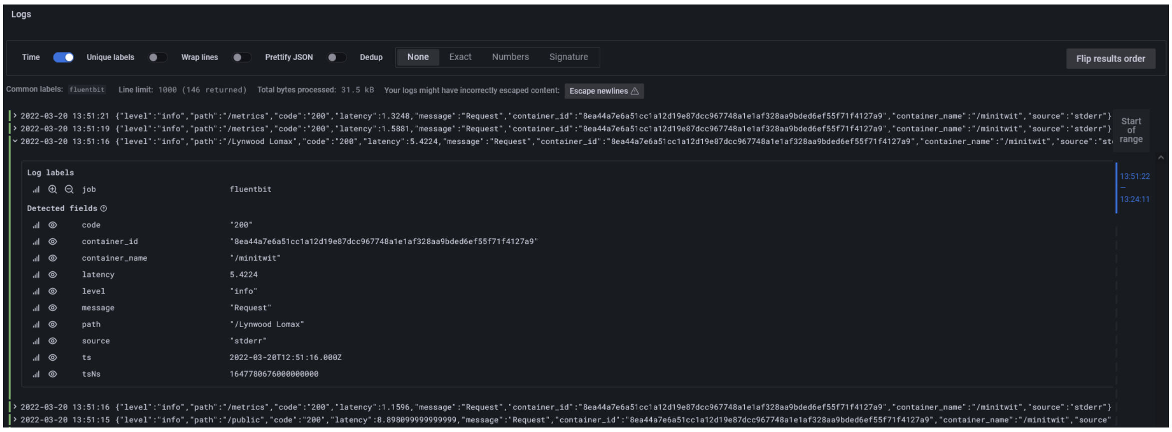
Figure 19: Alerts - Grafana (Source: Own image)

B Logging Structure



```
{"level":"info","time":"2022-03-04T16:48:07Z","message":"Starting MiniTwit application startup checks"}
{"level":"info","time":"2022-03-04T16:48:07Z","message":"Starting MiniTwit application startup checks - complete"}
{"level":"info","path":"/","code":"302","latency":0.3588,"time":"2022-03-04T16:48:20Z","message":"Request"}
{"level":"info","path":"/public","code":"200","latency":0.5233,"time":"2022-03-04T16:48:20Z","message":"Request"}
```

Figure 20: Logging Structure Examples (Source: Own image)



Common labels: fluentbit Line limit: 1000 (146 returned) Total bytes processed: 31.5 kB Your logs might have incorrectly escaped content:

```
> 2022-03-20 13:51:21 {"level":"info","path":"/metrics","code":"200","latency":1.3248,"message":"Request","container_id":"8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9","container_name":"/minitwit","source":"stderr"}
> 2022-03-20 13:51:19 {"level":"info","path":"/metrics","code":"200","latency":1.5881,"message":"Request","container_id":"8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9","container_name":"/minitwit","source":"stderr"}
˅ 2022-03-20 13:51:16 {"level":"info","path":"/Lynwood Lomax","code":"200","latency":5.4224,"message":"Request","container_id":"8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9","container_name":"/minitwit","source":"st

  Log labels
      job                     fluentbit
  Detected fields
      code                    "200"
      container_id            "8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9"
      container_name          "/minitwit"
      latency                 5.4224
      level                   "info"
      message                 "Request"
      path                    "/Lynwood Lomax"
      source                  "stderr"
      ts                      2022-03-20T12:51:16.000Z
      tsNs                    1647780676000000000

> 2022-03-20 13:51:16 {"level":"info","path":"/metrics","code":"200","latency":1.1596,"message":"Request","container_id":"8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9","container_name":"/minitwit","source":"stderr"}
> 2022-03-20 13:51:15 {"level":"info","path":"/public","code":"200","latency":8.898099999999999,"message":"Request","container_id":"8ea44a7e6a51cc1a12d19e87dcc967748a1e1af328aa9bded6ef55f71f4127a9","container_name":"/minitwit","source"
```

Figure 21: Logging - Grafana (Source: Own image)

C Dependencies

All dependency graphs are available as .svg files at github for greater visibility.

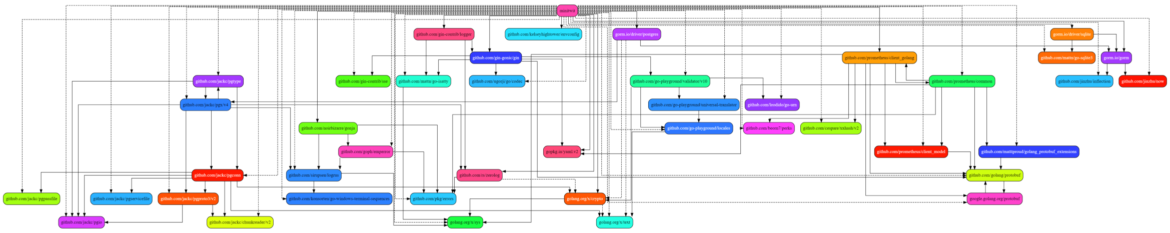

minitwit
github.com/gin-contrib/logger
github.com/kelseyhightower/envconfig
gorm.io/driver/postgres
gorm.io/driver/sqlite
github.com/gin-gonic/gin
github.com/prometheus/client_golang
github.com/mattn/go-sqlite3
gorm.io/gorm
github.com/jackc/pgtype
github.com/gin-contrib/sse
github.com/mattn/go-isatty
github.com/ugorji/go/codec
github.com/go-playground/validator/v10
github.com/prometheus/common
github.com/jinzhu/inflection
github.com/jinzhu/now
github.com/jackc/pgx/v4
github.com/go-playground/universal-translator
github.com/leodido/go-urn
github.com/noirbizarre/gonja
github.com/go-playground/locales
github.com/beorn7/perks
github.com/cespare/xxhash/v2
github.com/goph/emperror
gopkg.in/yaml.v2
github.com/prometheus/client_model
github.com/matttproud/golang_protobuf_extensions
github.com/jackc/pgconn
github.com/sirupsen/logrus
github.com/rs/zerolog
github.com/golang/protobuf
github.com/jackc/pgpassfile
github.com/jackc/pgservicefile
github.com/jackc/pgproto3/v2
github.com/konsorten/go-windows-terminal-sequences
github.com/pkg/errors
golang.org/x/crypto
google.golang.org/protobuf
github.com/jackc/pgio
github.com/jackc/chunkreader/v2
golang.org/x/sys
golang.org/x/text

Figure 22: Go mod dependencies, tool used gomod

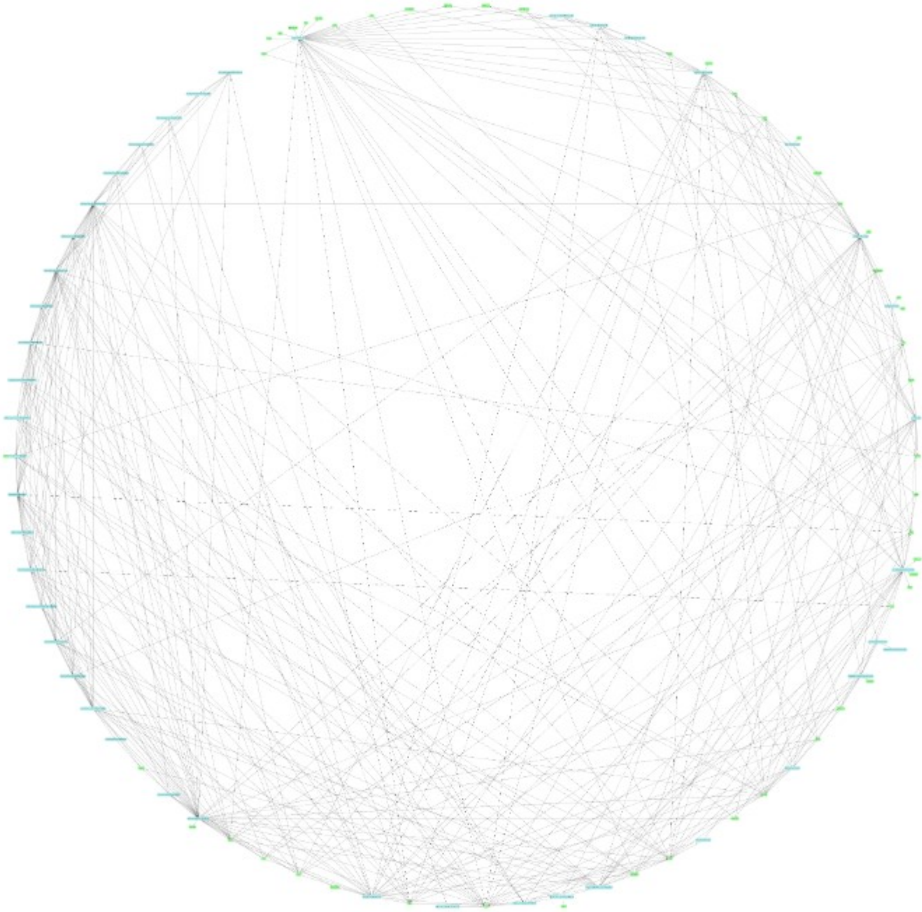


Figure 23: Gin dependencies, tool used godepgraph

D Security assessment

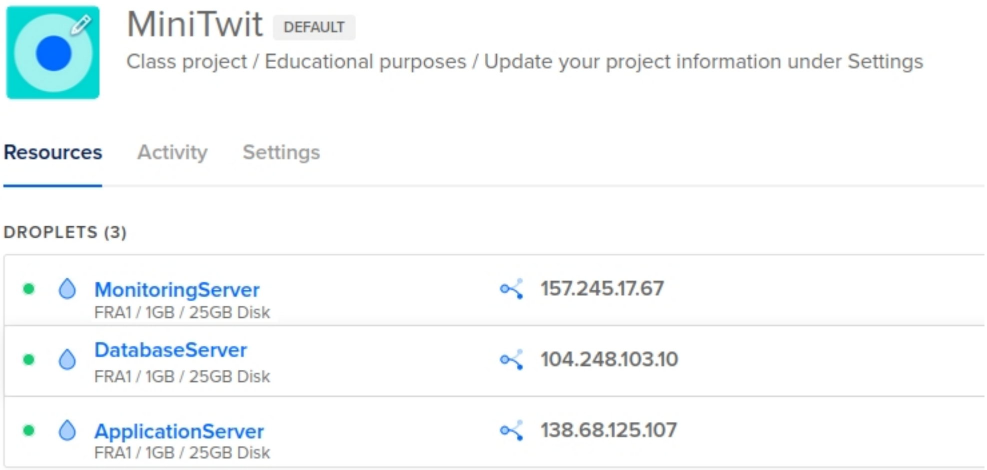
D.1 Risk Identification

First all of Develops ”assets” are identified, to define the scope of the assessment.

D.1.1 Identifiy assets

1. Minitwit Web Application Server
2. Postgres Database Server
3. Logging and Monitoring Server
4. DigitalOcean as Cloud Provider
5. Dockerhub as Docker Registry Provider
6. Github as Code Version Control, SCM, project documentation and CI/CD provider.
7. Discord Channel as secret sharing and communication.

The first 3 are servers monitoring, running and serving the Minitwit Web Application to customers. They are all small VMs hosted at Digital-Ocean, with 1 GB of RAM, 25 GB disk and a single CPU core (as can be seen on Figure 24). The 4., 5. and 6. are crucial 3rd party service providers, that are used in the minitwit project. The 7. is the team’s communication and secret sharing channel, which both contains passwords and keys, but also monitor alert notifications from the ”Logging and Monitoring Server”.



MiniTwit DEFAULT

Class project / Educational purposes / Update your project information under Settings

DROPLETS (3)

MonitoringServer FRA1 / 1GB / 25GB Disk	157.245.17.67
DatabaseServer FRA1 / 1GB / 25GB Disk	104.248.103.10
ApplicationServer FRA1 / 1GB / 25GB Disk	138.68.125.107

Figure 24: Caption

D.1.2 Identify threat sources

Minitwit Web Application Server - A vulnerability assessment with tools was done, with several tools. The full report[15] showed that we had several high risk threats. These threats include:

1. Path Traversal
2. SQL Injection

3. Absence of Anti-CSRF Tokens
4. CSP Header Not Set
5. Missing Anti-clickjacking Header

A path traversal attack aims to gain access to directories and files, which are not stored in the web root folder[19]. When running the automated vulnerability assessment, patch traversal was highlighted as a high risk threat. However, this does not seem to be relevant for our web-application, since it is impossible to access sensitive directories and files with "dot-dot-slash" sequences.

SQL injection attacks consists of injecting SQL queries through input fields in a web-application[20]. This attack can be used to maliciously modify database columns or even execute administrator operations on the database itself. However, this is not an issue for our application, since we're using an object-relational-mapping (ORM) to handle all database operations. Thus, we're not using dynamic queries to execute operations on the database, but delegate this to the ORM. This means that the efficacy of any SQL injection attack depends on the integrity of the ORM that we're using for our MiniTwit application. In this case, we're using GORM.[11]

Adding Anti-CSRF Tokens is also suggested by the vulnerability scanner. This makes sense, since this does not seem to be handled by default by Gin. In this case, it is worth to think about the impact an CSRF-attack would have on our MiniTwit application. A successful CSRF-attack is only limited to the privileges of the user and in this case, these include tweeting, following and unfollowing users.

Lastly it is important to note, that the application does not have any rate limiting. So it would be possible to deploy a Denial-of-Service (DoS) brute-force attack on the /login page, to obtain valid username and password combinations.

Postgres Database Server The database server has several services running, with WAN exposed ports like the Postgres server through NGINX and the node exporter service, with only the postgres server really being of interest. This can be seen on Figure 25

Threats to the postgres server could be if it was used to do privilege escalation on the machine that it is running on, which seems pretty unlikely. In a real business scenario data exfiltration could also be a threat, if any customer GDPR or business crucial data was present in the database.

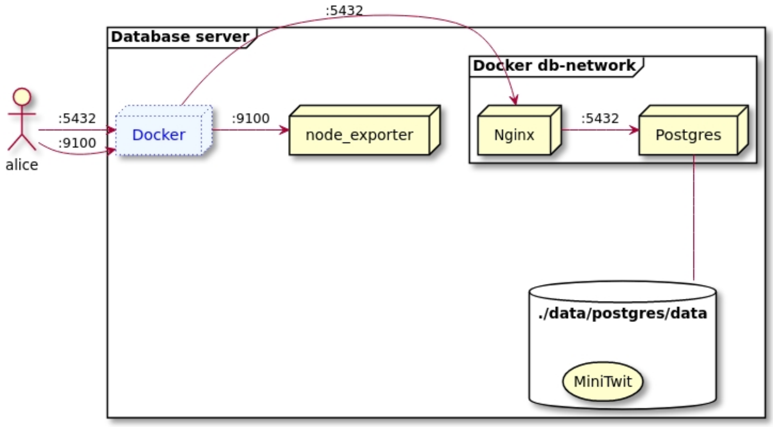


Figure 25: Database server diagram

Logging and Monitoring Server
The Logging and Monitoring server also has several services running, with WAN exposed ports through NGINX to: Loki, Grafana and Prometheus. Prometheus also talk to a node exporter and the groups own datascraper service. This can be seen on Figure 26. Loki is used to gather logging information from the application and Prometheus is gathering infrastructure metrics for monitoring.

Threats to these services:

1. Access to the Grafana dashboard might reveal business or deployment sensitive information.
2. Privilege escalation vulnerabilities to the machine in any of the service.

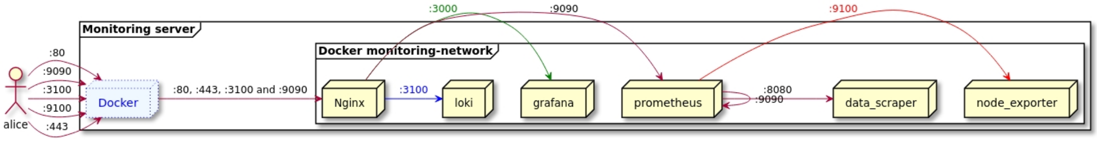


Figure 26: Monitoring and Logging server diagram

DigitalOcean as Cloud Provider DigitalOcean is used by the group as the Cloud provider, which offers a website for teams to configure their cloud infrastructure. The configuration can also be accessed through a REST API with an API key, which can be used to provision and configure current or new VMs. The website also provides a ”web console”, which gives the authorised user a web terminal, with access to the root user of the VMs.

Github as Code Version Control, SCM, project documentation and CI/CD provider The group uses Github for both its version control and doing Continues Integration with Github Actions. Since all the code is in public repositories, the group has to be extra cautious whenever pushing and dealing with secrets in the codebase. Another point, which was mentioned by Helge was that it can be a bit of a risk to depend on a single service provider like Microsoft's Github for multiple functions/roles.

Discord Channel as secret sharing and communication - Is the group's chosen method of communication and sharing secrets like Digital Ocean API keys, database passwords and etc. in cleartext. Discord is a communication platform, which is mostly used by the gaming community and thus not directly targeted to enterprise use. The platform has access control and with the use of a private server, should make it so it is for invited members only. However, scams[5] that lead to account takeovers or misconfigured server permissions, can lead to outsiders accessing "private" channels without being authorized and viewing secrets.

D.1.3 Risk scenarios, their impact and likelihood

Minitwit Web Application Server An adversary could in case of a SQL injection vulnerability get access to mess with the database server.(1) Not very likely, but could have high impact.

Another scenario would be that the response time of service would be degraded due to a password brute-force or a dedicated DoS attack.(2) Highly likely with medium impact, depending on the degree of degradation.

Postgres Database Server The current configuration of the server leaves the database open to anyone trying to brute-force access to the postgres database, which is only protected by a password.(3) Highly likely with high impact.

Logging and Monitoring Server A successful bruteforce attack of the grafana login page could lead to people getting access to business data and possibly get information about the database server setup.(4) Not very likely with low impact.

DigitalOcean as Cloud Provider The API key is leaked by a member through the Discord channel or in a commit to the public Github repository. The API key could then be used to destroy/remove the group's infrastructure. A worse scenario would be, that the intruder spins up powerful VM's (driving up our bill) and use them for cryptocurrency mining, connect them to a botnet or use them for criminal activity.(5) Not very likely with High impact.

Another scenario would be that one of the team members does not follow common password-policy practices, and therefore has the same passwords for several accounts with the same email address. Then in an event that there is a database/password-leak on any service that the member uses, an intruder could get access to all additional services like the member's DO account. By having access to the account the intruder could upload their own keys to the servers or simply use the web-based terminal to gain root access to the servers. (6) Not very likely with High impact.

Github as Code Version Control, SCM, project documentation and CI/CD provider The group uses Github for both its version control and doing Continues Integration with Github Actions. Since all the code is in public repositories, the group has to be extra cautious whenever pushing and dealing with secrets in the codebase. (7) Highly likely, since the group has already tried it with medium-high impact.

Another point, which was mentioned by Helge was that it can be a bit of a risk to depend on a single service provider like Microsoft's Github for multiple functions/roles.

Discord Channel as secret sharing and communication Someone gets access to one of the developers account, who accidentally also uses their Discord account in their spare time.(8) Or access could be given through an invitation link, by adding a malicious Discord

Bot or by using other social-engineering tricks. Not very likely with High impact.

D.2 Risk Analysis

From the previous section, the following risk matrix can be created.

Likelihood/Impact	Low	Medium	High
Likely		2,7	3
Possible			1
Unlikely	4		5,6,8

From this, we can sort the scenarios in a prioritised order: ”3,2,7,1”, where likelihood is more important than impact. The following mitigation plan based on discussion and research was decided on.

Postgres Database Server password bruteforce (3) Instead of having the postgres server open up for password-authentication, the team could setup an alternative authentication method like Kerberos, described in the postgres documentation[23]. Alternatively setting up SSH connection or simply setup a VPN between the application server and the database server.

Minitwit Web Application Server DoS or bruteforce (2) Setup rate limiting in the post message and login endpoints, will result in it becoming harder to do those attacks. Anti DDos solutions like proxying traffic through Cloudflare or use a CDN for static content, are also popular mitigation strategies.[**owasp˙dos˙mitigation**]

Github as Code Version Control, SCM, project documentation and CI/CD provider (7) There are several mechanisms that tries to automate secret detection like gitguardian[26], which uses git-hooks that can catch any secret leaking before committing or pushing leaking code. Running these scans and check, could be enforced by a company policy and don't have to run on each developer's machine. Other companies and the project's hosting provider DigitalOcean also scan open repositories, and tries to warn the owners, that a potential API key has been leaked.[4] The group even managed to receive an email from DigitalOcean, about leaking of the groups API key, at some point.

Minitwit Web Application Server DoS or bruteforce (1) As described in the subsubsection D.1.2, SQL injections happens often when the programmer is creating dynamic queries, typically using simple string concatenation. These are mostly seen in very naive implementations, where an ORM nor any input validation are present. Lastly using an ORM correctly, using prepared statements or stored procedures are usually recommended to avoid creating an SQL injection vulnerability.