MSc_group_c

ITU MiniTwit Report Skeleton

May 2024

1 Introduction

2 System perspective

This section presents the system.

2.1 Design and architecture

The system is primarily built using Go (Golang) for backend development. The Echo web framework is used for HTTP routing and middleware management. PostgreSQL serves as the database. Additionally, the system uses various Go libraries for security, session management, data serialization, monitoring, system metrics and external systems that will be presented later.

This section presents the architecture of the system by exploring the `src` folder of the repository.

2.1.1 Module diagram

An overview of the modules of the codebase in the `src` folder is presented by the following package diagram.
Note that within the `handlers` folder, the classes `auth.go`, `message.go`, and `user.go` and their dependencies are highlighted, depicting the complexity of this central module. This is thereby not a normal package diagram.

In the diagram it can be seen, that the main.go file orchestrates the system. It (in this context) has the responsibility for: 1. Render the template (frontend) 2. Initialize a new instance of the database object 3. Setup middleware 4. Setup routes, which have the responsibility of exposing the endpoints that further orchestrates to the handlers module for the logic of the API.

2.1.2 Sequence diagrams

Two sequence diagrams have been created to show the flow of information through the system, from a “Follow” request by a user, to the system’s returned

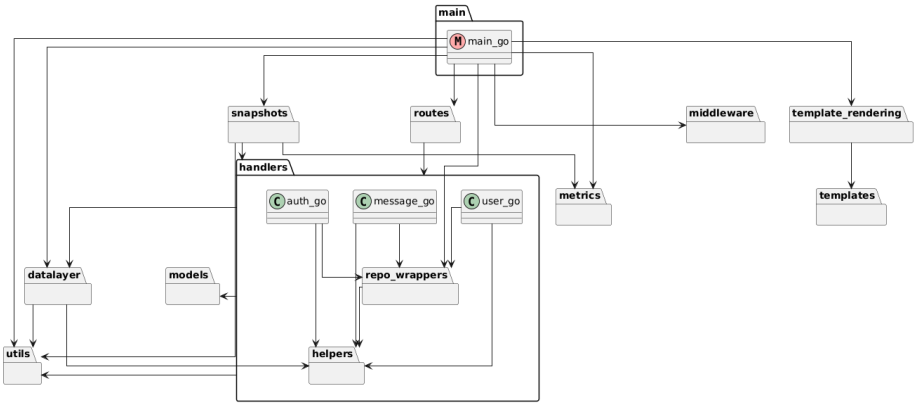

Figure 1: Module diagram

response.

They contain the following high-level lifelines: - User Interface: The minitwit web application - API Handlers: All functions in the `handlers` package, which handle requests to the API endpoints. - Datalayer: The `datalayer`package, which handles database interactions and returns structs (see the `model` package). - Database: The postgres database.

The first version shows the processes involved when the request is sent via. the *UI*, whereas the second version shows the processes involved when sent via. the *API*.

Note that the two versions use different endpoints to interact with the same API.

2.2 Dependencies

Dependency	Description
Go (Golang)	Programming language for backend development.
github.com/labstack/echo/v4	Web framework for routing and HTTP handling.
github.com/gorilla/sessions	Session management with secure cookie support.
github.com/lib/pq	PostgreSQL driver for database connectivity.
PostgreSQL	Relational database storing
golang.org/x/crypto	Cryptographic utilities for security features.
github.com/prometheus/client_golang	Prometheus client library for metrics and monitoring.
github.com/shirou/gopsutil/v4	System utilities and metrics collection for health monitoring.
github.com/klauspost/compress	Compression libraries to optimize data transfer.
golang.org/x/sys	Low-level OS interaction and system calls.
google.golang.org/protobuf	Protocol Buffers support for data serialization.
github.com/gorilla/securecookie	Secure cookie encoding/decoding for session safety.

Dependency	Description
Gravatar	External web service providing avatar images generated from email hashes (used for user profiles).

2.3 Current state of the system

2.3.1 SonarQube analysis summary

The following table summarizes key code quality metrics from SonarQube analysis:

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

2.3.2 Code Climate

The following table summarizes key code quality metrics from Code Climate 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

2.3.3 Overall assessment

Both tools show that the `handlers` module has relatively high complexity, which may require focused attention for maintainability.

2.4 Orchestration

To streamline deployment, Docker, docker-compose, Docker Swarm, and Terraform are used.

The Dockerfile copies all source code from the `src` package to a binary image of the program.

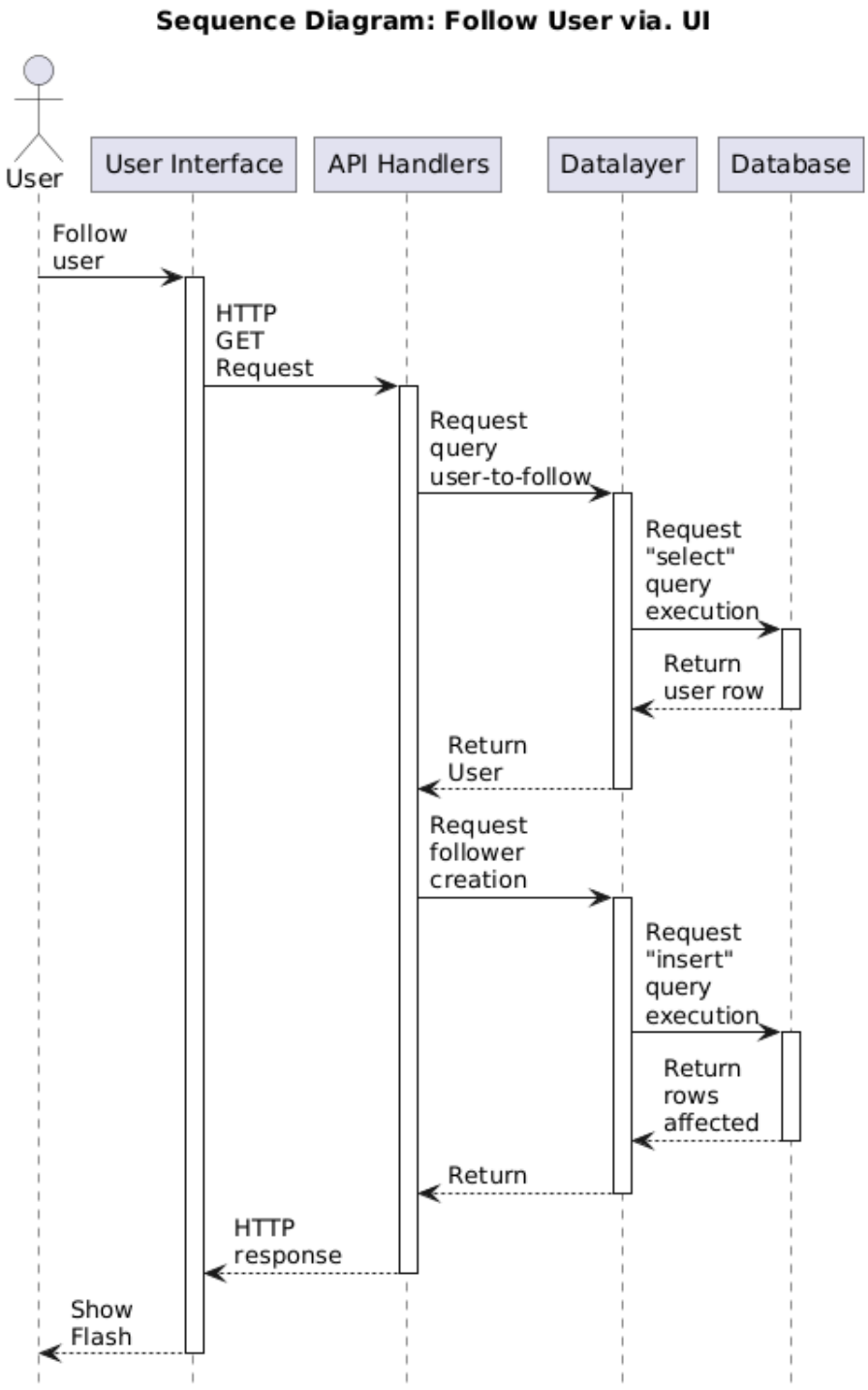

Figure 2: Sequence diagram - Follow request via UI

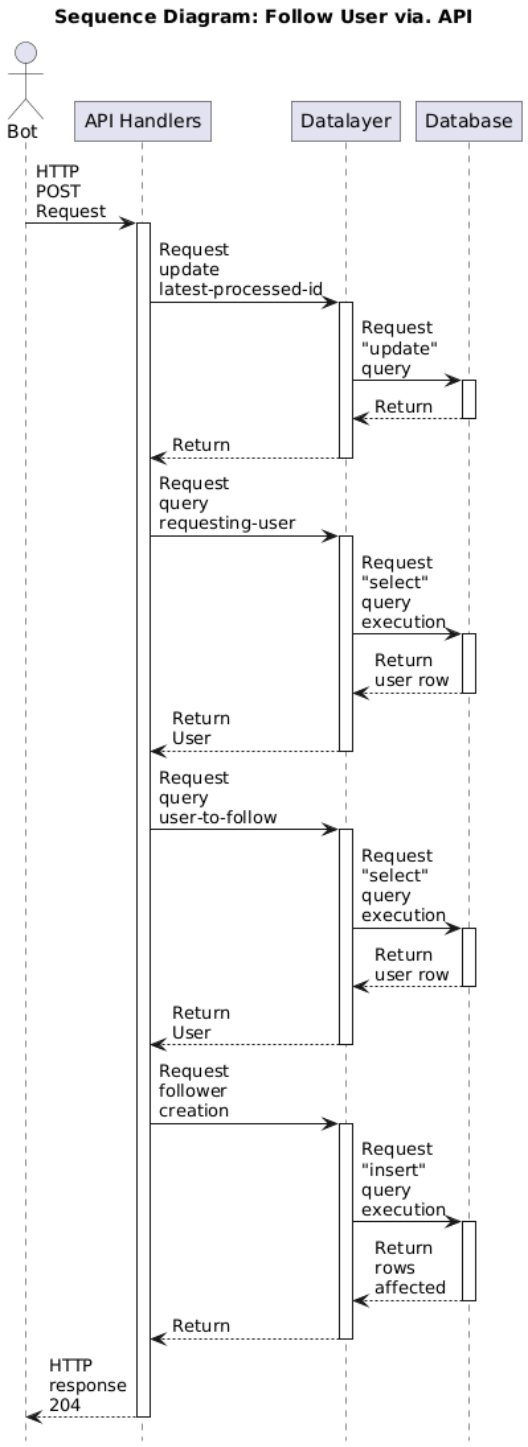

Figure 3: Sequence diagram - Follow request via API

There are two docker-compose files, `docker-compose.yml` and `docker-compose.deploy.yml`. Both define the six central services of the system: app, prometheus, alloy, loki, grafana, and database.

`docker-compose.yml` is needed for local deployment. It uses localhost IP-adresses and has default usernames and passwords.

`docker-compose.deploy.yml` is used for remote deployment. It builds on `docker-compose.yml`, but replaces information where relevant. It specifies the configuration of a Docker Swarm with one manager and two workers: The app runs on two worker replicas, while logging and monitoring services are constrained to only run on the manager node (though alloy collects logs from everywhere). This enables horizontal scaling.

Infrastructure-as-Code is used simplify the setup of the Docker Swarm remotely. Terraform files can be found in `.infrastructure/infrastructure-as-code`. Automatic deployment via. Terraform works as illustrated in the sequence diagram below.

2.5 Deployment

2.5.1 VPS

To host the system on a remote server, DigitalOcean was chosen as the VPS provider. This choice was based on pricing (see @tbl:vps-comparison), its apparent ease-of-use[@Quinn_2022] [@aliamin7] [@Finder_2023], its familiarity to the group.

Table 4: Price comparison of VPS providers. {#tbl:vps-comparison}

VPS	**DigitalOcean**	**Microsoft Azure**	**Oracle**	**AWS (Lightsail)**
Virtual Machine Price	ca. $12/mo [@digitalocean_price]	ca. $15/mo [@azure_price]	$13/mo [@oracle_price]	ca. $12/mo [@aws_lightsail_price]
Storage Price	50GB included [@digitalocean_price]	ca. $5 (64GB) [@azure_price]	ca. $2.5 (50GB) [@oracle_price]	ca. $12/mo [@aws_lightsail_price]
Total Price	ca. $12/mo	ca. $20/mo	ca. $15.5/mo	ca. $12/mo

2.5.2 Infrastructure-as-Code

To ensure a consistent and automatic creation of the infrastructure of the system on DigitalOcean, Terraform was used. Terraform is an infrastructure as code tool[@Terraform_MainPage], which has an easy to use inbuilt provider for

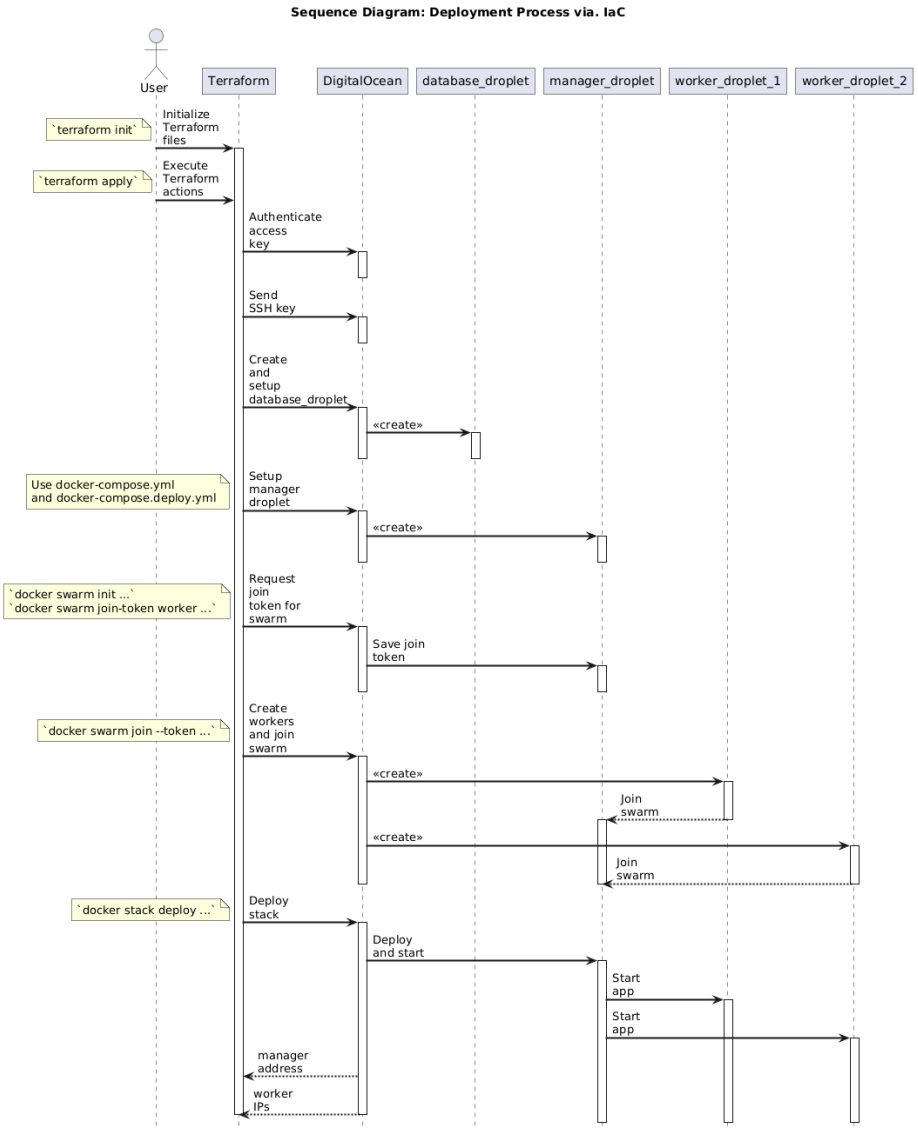

Figure 4: Sequence diagram of IaC

DigitalOcean[@Anicas_Hogan_2022]. Please see figure **ref** for an overview of how Terraform builds the infrastructure of the system on DigitalOcean.

2.5.3 Allocation viewpoint

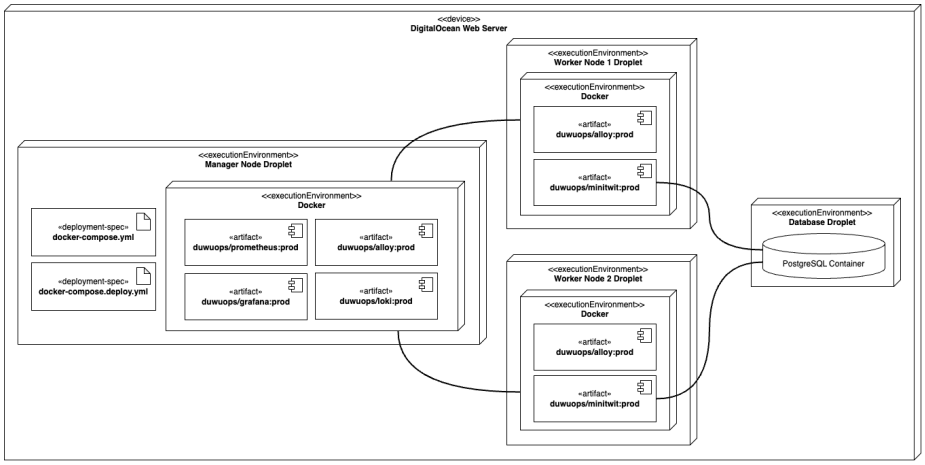

Figure 5: Deployment diagram

3 Process perspective