DevOps, Software Evolution and Software Maintenance, BSc (Spring 2024)

Course code: BSDSESM1KU

May 10, 2024

Exam Assignment by:

Student	Email
Daria Damian	dard@itu.dk
Hallgrímur Jónas Jensson	hajj@itu.dk
Mathias E. L. Rasmussen	memr@itu.dk
Max-Emil Smith Thorius	maxt@itu.dk

Contents

1 System's Perspective

1.1 Design and architecture of the Minitwit system

ITU-MiniTwit is designed as a microservices architecture, leveraging containerization to ensure isolation, ease of deployment, and scalability. The system's architecture facilitates independent development and deployment of services, which enhances maintenance and testing capabilities. Detailed Architecture:

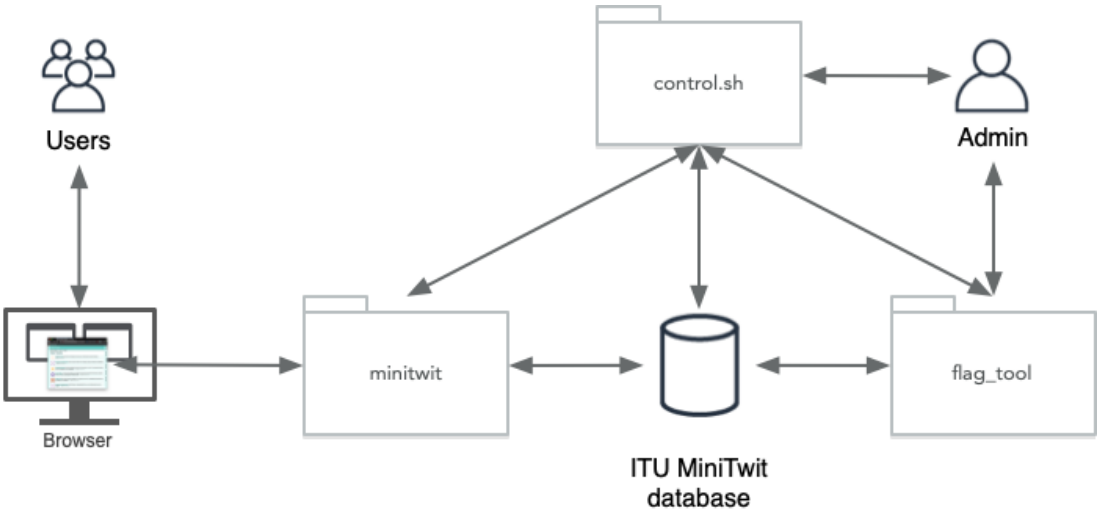

Figure 1.1: High-level ITU-MiniTwit architexture

- API Service:

  Function: Handles all client-server interactions, processing client requests and sending responses back to clients.

  Implementation: Implemented using Go, indicated by go.mod files, which is suitable for creating performant and scalable backend services.

  Files: Located within the api directory, which contains all the routing and logic for handling HTTP requests.

- Database Service:

  Function: Manages data storage and retrieval, serving as the persistent storage layer for the application.

  Implementation: Utilizes SQL, as suggested by the presence of schema.sql and query.sql, indicating the use of relational database management systems.

  Files: The database directory contains scripts and configurations for database setup and management.

- Front-end:

  Function: Provides the user interface through which users interact with the ITU-MiniTwit application.

  Implementation: While specific front-end files weren't listed, typically, this would involve HTML, CSS, and JavaScript files possibly served by the Python service or another static file server.

  Files: Likely included in the root directory or within a static or templates directory that contains HTML templates and JavaScript files.

- Containerization and Orchestration:

  Utilizes Docker, as evidenced by Dockerfile and docker-compose.yml, to containerize the application components, ensuring that each component runs in an isolated environment with its dependencies.

  Docker Compose helps in defining and running multi-container Docker applications, where services such as the API and database are orchestrated to work together.

1.2 Dependencies of the Minitwit system

On the Development Environment:

Vagrant: Used for provisioning virtual environments, ensuring that all developers work within a consistent development environment irrespective of their local machine configurations. Makefile: Simplifies project builds, allowing developers to automate complex sequences of tasks with simple commands. Programming Languages and Frameworks:

Python: Used for scripting and possibly serving web content. The requirements.txt file lists all Python library dependencies. Go: Utilizes modern language features of Go for efficient back-end services. External APIs and Services:

Prometheus: Employed for monitoring the application's performance metrics, helping in proactive management and optimization of the application performance.

1.3 Important interactions of subsystems

1.4 Current state of the system

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec venenatis enim a nulla molestie, ac feugiat justo egestas. Nullam interdum lorem et neque ullamcorper volutpat sed sit amet tortor. Praesent sit amet aliquet risus, et accumsan mi. Sed facilisis condimentum varius. Praesent et nunc cursus, laoreet nisi a, venenatis nisl. Integer diam magna, iaculis at dapibus et, rutrum in leo. Pellentesque feugiat diam felis, quis ornare eros dignissim et. Maecenas sed laoreet nunc. Morbi porttitor massa id dui aliquam, non ultrices dolor malesuada.

Cras et porta ex. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam erat volutpat. Phasellus eget ipsum sit amet nulla porttitor rhoncus at eget elit. Aenean vulputate, urna sed lacinia luctus, sem nisi iaculis nunc, vitae mattis neque sem in felis. Sed tempor tincidunt dapibus. Morbi porta ex erat, sed ornare mi gravida nec. Phasellus ut nunc venenatis, mollis est vestibulum, laoreet nibh. Mauris in vulputate diam. Nunc ullamcorper vestibulum velit, eget volutpat leo vulputate at. Nam in tortor id dolor elementum lobortis at ut elit. Nulla in interdum mi. Sed aliquam ullamcorper blandit.

2 Process' perspective

2.1 CI/CD Chain

We implemented a GitHub Actions workflow to automate the process of testing, building and deploying the most recent version of Minitwit. It is set to execute on each push to the 'Main' branch of our GitHub repository. The workflow is separated into three main jobs: 'BuildAndTest', 'Deploy', and 'Release' which are executed sequentially, each dependent on the last executing successfully.

2.1.1 Triggers

The workflow is triggered on two specific GitHub events: manual triggers via workflow_dispatch and on pull requests to the main branch.

2.1.2 Jobs

BuildAndTest

In this step all relevant images are built and pushed to the Docker hub and then run and tested to ensure that the system will work as expected. The key steps are as follows:

1. **Checkout:**

- Uses `actions/checkout@v2` to fetch the codebase from the repository.

2. **Environment Setup:**

   - Dynamically creates a `.env` file with database configurations sourced from GitHub secrets.

3. **Docker Operations:**

   - Logs into Docker Hub using credentials from GitHub secrets to push built images.
   - Builds and pushes multiple Docker images, including:
     - The application image
     - API image
     - A test database image

4. **Python Setup and Dependency Installation:**

   - Configures the Python environment.
   - Installs necessary dependencies to ensure consistent testing conditions.

5. **Testing:**

   - Executes integration tests by setting up the application and its dependencies in Docker containers.
   - Conducts API tests and application-specific tests to ensure functionality and reliability.

Deploy

The **Deploy** job is activated on successfully completing the **BuildAndTest** job. This job manages the deployment of the application to the remote server where the application is hosted. The steps are:

1. **Checkout:**

   - Retrieves the latest codebase from the repository, uses `actions/checkout@v2` similar to the first job.

2. **SSH Configuration:**

- Prepares SSH keys to establish a secure connection to the deployment server.

3. **Deployment Execution:**

   - Updates environmental configurations and transfers necessary files to the server using SSH.
   - Employs Docker Compose on the server to pull the latest Docker images.
   - Deploys the images using Docker Stack, which effectively updates the running application on the server.

Release

After the previous jobs have been successfully completed, this job manages software versioning and public release.

1. **Version Calculation:**

   - Executes a script to determine the new version number for automated version tracking.

2. **GitHub Release Creation:**

   - Uses `actions/create-release@v1` to create a formal release on GitHub.
   - Tags the release with the new version number and provides release notes outlining the changes in this version.

2.2 Monitoring

For monitoring

2.3 Logging

2.4 Security assessment

2.5 Scaling strategy

3 Lessons Learned Perspective

3.1 Evolution and refactoring

3.2 Operation

3.3 Maintenance

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec venenatis enim a nulla molestie, ac feugiat justo egestas. Nullam interdum lorem et neque ullamcorper volutpat sed sit amet tortor. Praesent sit amet aliquet risus, et accumsan mi. Sed facilisis condimentum varius. Praesent et nunc cursus, laoreet nisi a, venenatis nisl. Integer diam magna, iaculis at dapibus et, rutrum in leo. Pellentesque feugiat diam felis, quis ornare eros dignissim et. Maecenas sed laoreet nunc. Morbi porttitor massa id dui aliquam, non ultrices dolor malesuada.

Cras et porta ex. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam erat volutpat. Phasellus eget ipsum sit amet nulla porttitor rhoncus at eget elit. Aenean vulputate, urna sed lacinia luctus, sem nisi iaculis nunc, vitae mattis neque sem in felis. Sed tempor tincidunt dapibus. Morbi porta ex erat, sed ornare mi gravida nec. Phasellus ut nunc venenatis, mollis est vestibulum, laoreet nibh. Mauris in vulputate diam. Nunc ullamcorper vestibulum velit, eget volutpat

leo vulputate at. Nam in tortor id dolor elementum lobortis at ut elit. Nulla in interdum mi. Sed aliquam ullamcorper blandit.