DevOps24-groupK

Course Code: BSDSESM1KU

GitGurus

Spring 2024

Andreas Guldborg Hansen	aguh@itu
Andreas Severin Hauch Trøstrup	atro@itu.dk
Frederik Petersen	frepe@itu.dk
Mads Aqqalu Roager	mroa@itu.dk
Silke Holme Bonnen	ssbo@itu.dk

Contents

1 Systems' perspective

1.1 Design and architecture

An overview of the architecture of our MiniTwit application can be found in figure 1. The entire system runs on DigitalOcean, with the exception of our monitoring and logging that runs on New Relic. In DigitalOcean, we have 3 primary nodes running. A swarm manager, and two workers. These 3 nodes are a part of a docker swarm network that runs the minitwit web app and minitwit simulator API containers as docker services. The workers and manager also run a New Relic agent that collects logs and system monitoring information, and send it to new relic. The manager node runs a Nginx reverse proxy, that handles https encryption, and load balancing between the workers. Lastly, in DigitalOcean we also have a managed database running, that acts as the database for the application. What the diagram below does not show is that the swarm manager also has a container running the web app and the simulator API. The reason why this is not shown on the diagram is because the Nginx reverse proxy has been set to only forward requests to worker 1 and 2. That means that the web app and the simulator api in the swarm manager will never be used.

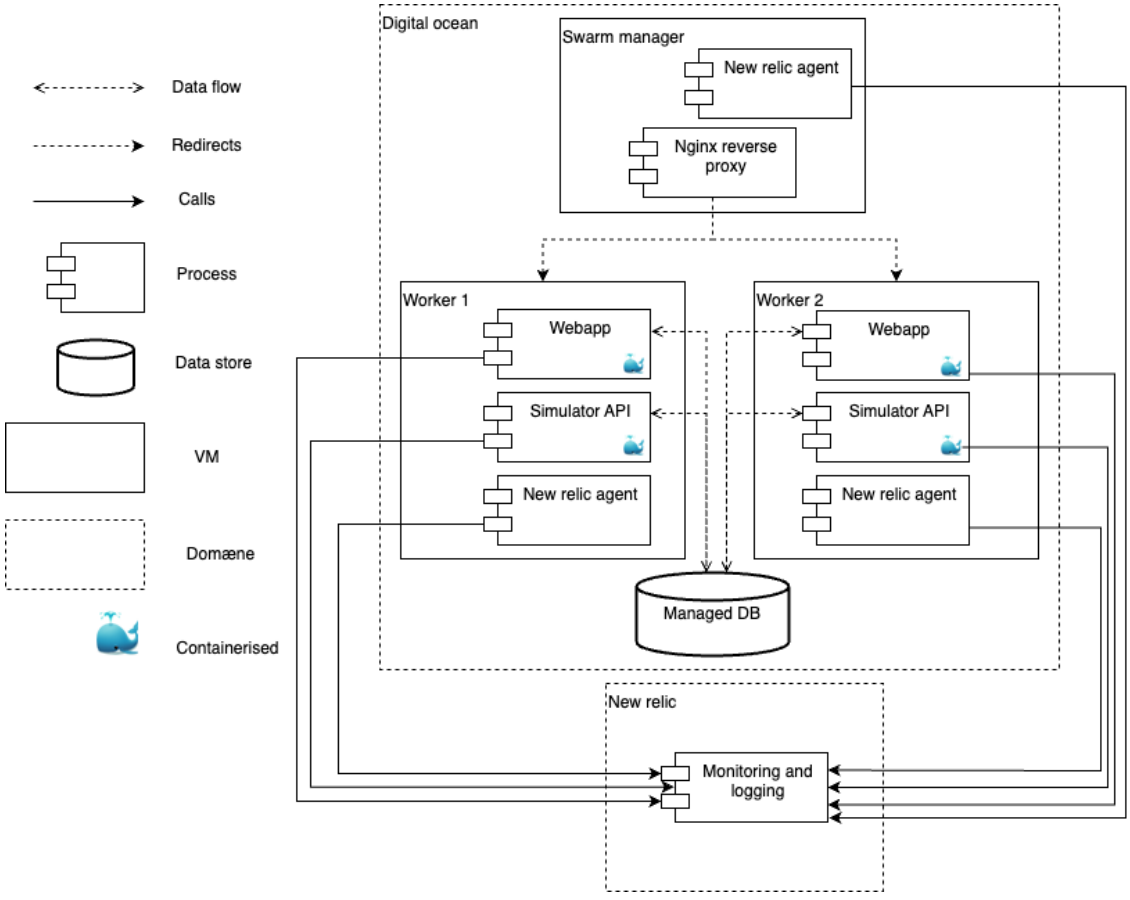

Figure 1: The figure shows a diagram of the architecture of minitwit

1.2 Dependencies

Describe and illustrate: all dependencies of your ITU-MiniTwit systems on all levels of abstraction and development stages. That is, list and briefly describe all technologies and tools you applied and depend on.

DigitialOcean is the only critical dependency for keeping the application running. That is the 3 VM's and the database.

We do not show further dependecies...

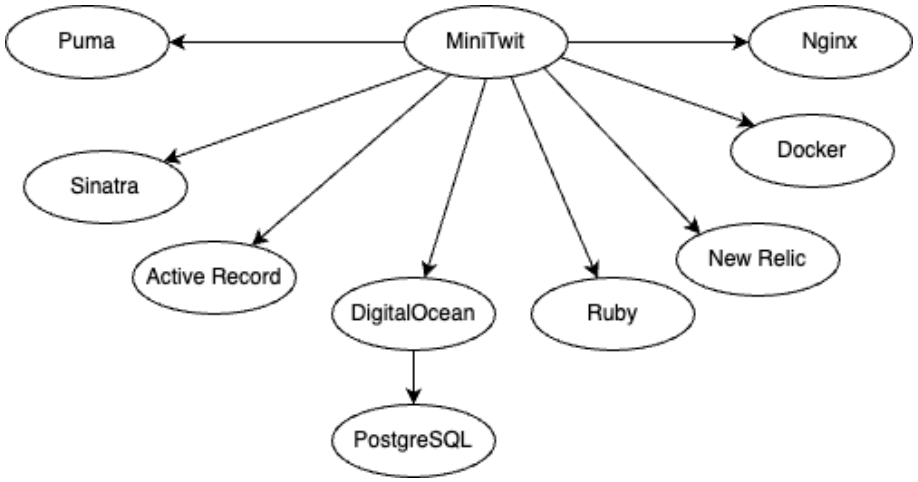

Figure 2: Production dependency graph.

1.3 Interactions of subsystems

For example, via an illustrative UML Sequence diagram that shows the flow of information through your system from user request in the browser, over all subsystems, hitting the database, and a response that is returned to the user. Similarly, another illustrative sequence diagram that shows how requests from the simulator traverse your system.

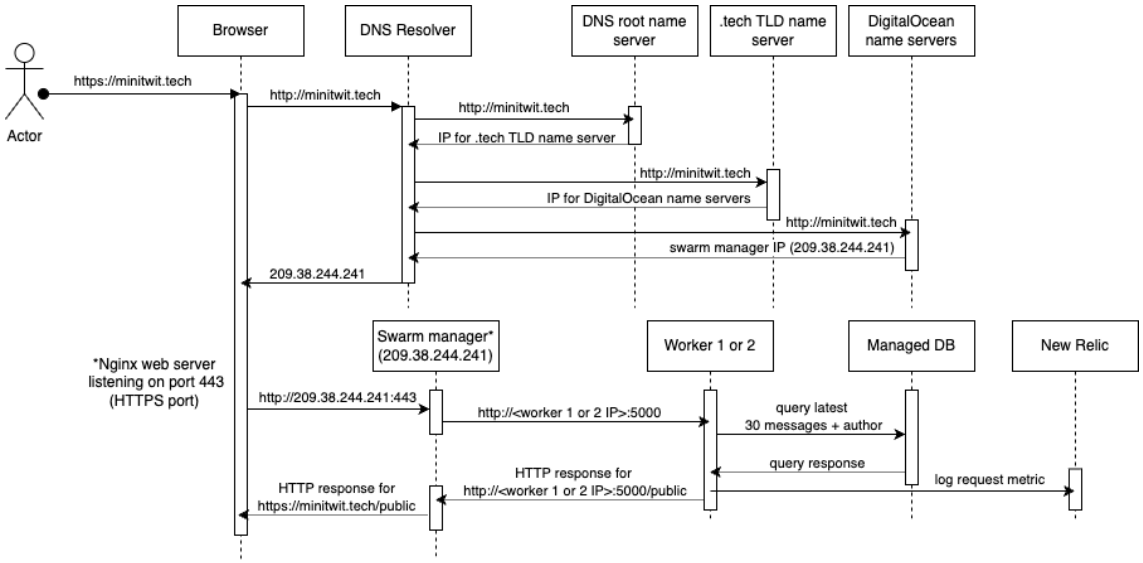

Figure 3: A sequence diagram describing the process of calling "minitwit.tech/public"

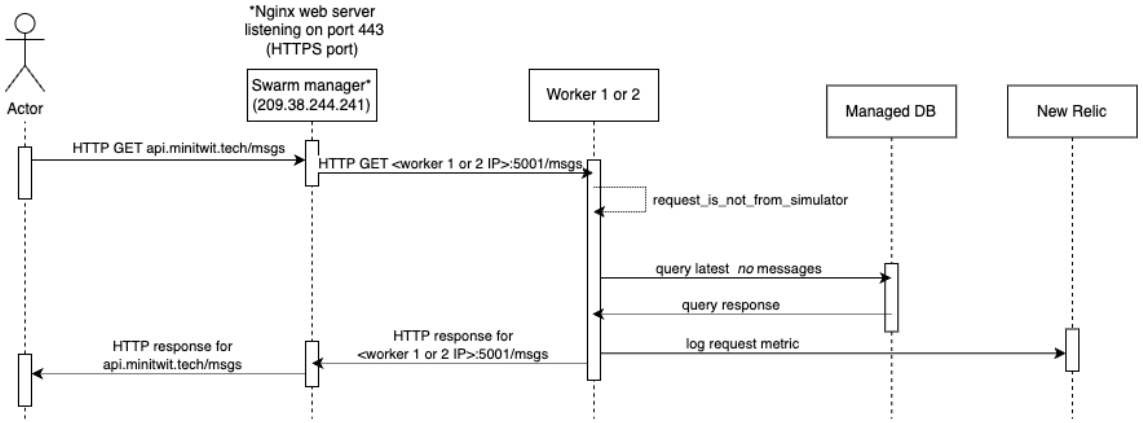

Figure 4: A sequence diagram describing the process of calling "api.minitwit.tech/msgs"

1.4 Current state

Describe the current state of your systems, for example using results of static analysis and quality assessments.

2 Process' perspective

This perspective should clarify how code or other artifacts come from idea into the running system and everything that happens on the way. In particular, the following descriptions should be included:

2.1 CI/CD chain

A complete description of stages and tools included in the CI/CD chains, including deployment and release of your systems.

2.1.1 Deployment workflows

Our deployment workflows runs in GitHub LSP

2.1.2 Report workflows

To ensure our report in our repository is always up to date, we utilise a built-in synchronization between Overleaf and GitHub. Once changes are made that a group member wish to save, one click from within Overleaf synchronizes the contents to a separate repository in our GitHub organisation, which triggers two Github Actions workflows:

1. A workflow from within this report repository that notifies our main repository of any changes made to our report repository.
2. A workflow from within the main repository that is called by the previously mentioned workflow, which triggers a git submodule update and commits it to the repository.

2.2 Monitoring & Logging

How do you monitor your systems and what precisely do you monitor? Both monitoring and logging of this project is done using New Relic.

What do you log in your systems and how do you aggregate logs?

2.3 Security

Brief results of the security assessment and brief description of how did you harden the security of your system based on the analysis

2.4 Scaling and upgrading

Applied strategy for scaling and upgrades

2.5 Use of AI tools

In case you have used AI-assistants during your project briefly explain which system(s) you used during the project and reflect how it supported/hindered your process.

In the beginning of the project, not all team members were familiar with the programming language Ruby. In order to get familiar, some of us had the Github Copilot extension enabled, allowing us to participate on a similar level as those who were more familiar with the language.

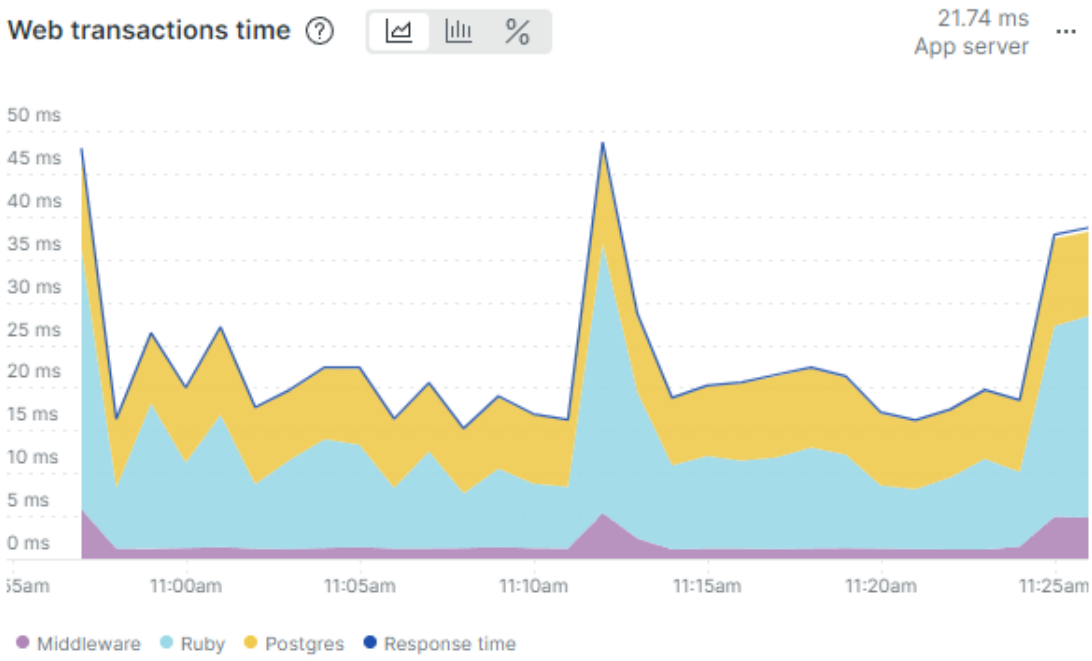

Figure 5: The newrelic graph depicting web transaction times by application layer

In the review process, ChatGPT has been used as a tool to interpret changes made by other members, by prompting ChatGPT to help understand the effects and consequences of the changes.

3 Lessons learned perspective

Describe the biggest issues, how you solved them, and which are major lessons learned with regards to: evolution and refactoring, operation, and maintenance of your ITU-MiniTwit systems. Link back to respective commit messages, issues, tickets, etc. to illustrate these. Also reflect and describe what was the "DevOps" style of your work. For example, what did you do differently to previous development projects and how did it work?

3.1 Evolution and refactoring

3.2 Operation

3.3 Maintenance