DevOps Report

Group F
(insert names)

May 2024

Contents

List of Figures

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

Chapter 2

System Perspective

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

2.1   Design and architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

2.2   Systems dependencies

The list below, is the technolgies we are using in our application, group into categories of technogy.

Project Start - choose framework

- Microsoft AspNetCore v2: For basis web framework to handle, request, athentication,

- Dotnet API framework: For handling RESTFull API's

- Swagger: For dotnet API visualization

- Microsoft EntityFrameworkCore version 8.x: For general handling dependency injection to a database connection.

- Postgress v.8: Choosen database servere for this project.

- Docker / Docker-compose / docker-swarm: For containerized applications

- gravatar: For provide user profile images.

- Github / Github actions/ Github secrets: For storing and deploying source code

- DigitalOcean: Production server for our application

Monitoring

- prometheus/prometheus-net/prometheus-net.AspNetCore: For provide metrics to Grafana.

- Grafana: For Data Visualization of data from prometheus and direct data from Postgress database.

Logging

- Serilog AspNetCore v.8: For logging.

- Fluentd: For collecting logs from different sources.

Software quality - Testing

- Sonarqube: Static analyzer for code quality and securit issues

- CodeClimate: Another static analyzer also for code quality and securit issues

- Playwright: UI testing

- Cypress: End-2-end testing

- Xunit: For Basic Unit testing

Chapter 3

Process Perspective

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

3.1 CI/CD pipelines

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

3.2 Monitoring

3.2.1 Prometheus

Decision Log

3.2.2 Grafana

Decision Log

3.3 Logging

3.3.1 fluentd

Decision Log

First we tried to implement our logging with kibana and elastic search, which worked on our local machines. The problem with this was that we only had 1GB RAM on our web server, which wasn't enough to run elastic search. To resolve that issue we looked for other technologies like fluentd.

When facing implementing logging to a system several challenges come up, for example there are disparate log formats of different applications, logs that are scattered across multiple sources (data fragmentation), reliability concerns like network outages and routing different logs to different destinations based on the application or service.

In order to manage these difficulties we decided to use fluentd. Fluentd is an open source data collector which processes the data from various sources, making it easier to manage and analyze. The following aspects show why fluentd is so powerful:

- **Unified Logging Layer:** Fluentd provides a unified platform for collecting, transforming, and routing logs from diverse sources, simplifying the log management process.

- **Data Agnosticism:** Regardless of the data source—be it Prometheus metrics, Grafana dashboards, or application logs—Fluentd seamlessly collects and processes data in a unified format, fostering interoperability and ease of analysis.

- **Configurable Routing:** With Fluentd, we could effortlessly configure log routing based on application or service, directing logs to specific destinations for centralized storage and analysis.

- **Flexibility:** Fluentd's versatility enables us to send data from any source to any destination, empowering us to adapt to evolving logging requirements seamlessly.

Implementation of fluentd

A diagram describing our use and implementation of fluentd can be seen in figure **??**.
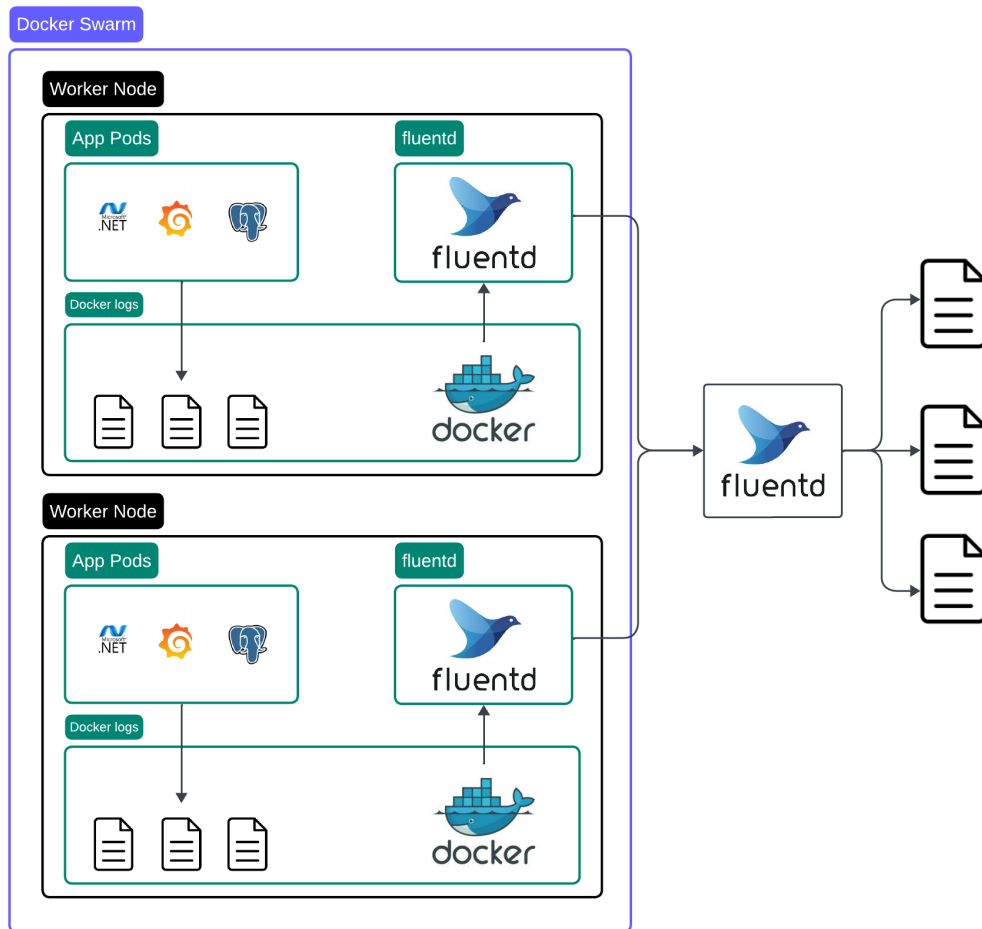

Figure 3.1: Diagram describing our implementation of fluentd

Fluentd Configuration

Fluentd is configured to listen for incoming log data via the forward input plugin on port 24224. Log data from different sources (minitwit-database,

minitwit-service, prometheus, grafana) is matched using the match directive which specifies the destination path and format for the corresponding log data. Log data is buffered using the buffer directive to handle data spikes or network outages, ensuring reliable log collection.

Docker Compose Configuration

A fluentd container is defined using the fluent/fluentd:v1.12-debian image. Each service in the Docker Compose file depends on the fluentd service for logging. Logging is configured to use fluentd as the logging driver, with specific options (fluentd-async-connect, fluentd-retry-wait, fluentd-max-retries) to ensure reliable log transmission. Each service specifies a unique tag to route log data to the appropriate destination, which in our case are simple log files, in fluentd based on the matching rules defined in the fluentd.conf file.

3.4   Security

3.4.1   Risk Identification

Assets

We have different assets that need to be protected:

- The web application: The web app serves and receives content from users.

- The API: The API allows for software developed by third-parties (f.i. the simulator) to obtain content from our application and sent their own.

- The database: The database stores all user information.

- Development, debugging, and deployment tools:

  - SSH: We use SSH to connect to and configure our Ubuntu Server VM(s).
  - Graphana/Prometheus: We use Graphana and Prometheus to monitor our system.
  - GitHub: We use GitHub to store our code, as well as the deployment logic for our application.
  - GitHub Actions: We use GitHub Actions for our CI/CD pipeline.

- GitHub Secrets: Secrets needed by the CI/CD pipeline are stored in GitHub Secrets.
- DigitalOcean: Our VM and database are provided by DigitalOcean and managed via their web interface or API.

Threat sources

Thread sources can be diverse, and attach different layers of our application:

- Vulnerabilities in our code: Vulnerabilities in our code could allow attackers to access user data or send malware to users. Among the vulnerabilities that we **could** have, these are the most relevant ones:

  - SQL injection: SQL injection would allow attackers to access and modify user data.
  - Cross Site Scripting (XSS): Cross Site Scripting would allow attackers to potentially access and modify user data, as well as to sent malicious information to users.

  We need to emphasise that we have checked our code for such vulnerabilities, and we haven't found any. We also follow best practices to avoid such vulnerabilities. However, that does not mean that the vulnerabilities are not present, since it is impossible to prove so.

- Missing HTTPS and firewall: The use of basic HTTP could result on attackers snooping on users and obtaining password, or on man-in-the-middle attacks.

- Misconfigured firewall: A misconfigured firewall could give attackers direct access to, amongst others, the DB.

- Stolen SSH keys for the VM: Stolen SSH keys for the VM would allow attackers to execute arbitrary code in our VM.

- Stolen credentials for Graphana: Stolen credentials for Graphana would allow attackers to obtain access to the monitoring data of our system.

- Public Prometheus metrics: Metrics from the application for Prometheus to scrap are publicly available at */metrics*. Attackers could use this information to gain information about the operations being executed on the system.

- Stolen credentials for GitHub: Stolen credentials or tokens for GitHub would allow attackers to view, modify, and delete our code and/or deployment logic. It would also provide them with access to our secrets and CI/CD pipeline, essentially allowing them to execute arbitrary code in our VM.

- Stolen credentials for DigitalOcean: Stolen credentials or API keys for DigitalOcean would allow attackers to delete and/or access our VM and database.

- Vulnerabilities in the OS or Docker Images used: Vulnerabilities in the OS or Docker images used could allow attackers to take control over our system.

- Supply chain attacks: Supply chain attacks could cause vulnerabilities to be introduced in the dependencies we use.

Risk scenarios

Some possible scenarios of attacks are as follows:

1. Attacker performs SQL injection on web application to download sensitive user data.

2. Attacker performs SQL injection on API to download sensitive user data.

3. Attacker performs XSS on web application to download sensitive user data.

4. Attacker performs XSS on web application to deliver malware.

5. Attacker performs XSS on web application to download sensitive user data.

6. Attacker snoops on user traffic and obtains a user's password, which then uses to impersonate them.

7. Attacker performs a man-in-the-middle attack on a user and provides them with false information.

8. Attacker steals SSH keys, connects to the VM, from there to the DB, and downloads sensitive user data.

9. Attacker steals SSH keys, connects to the VM, and uses it to deliver malware.

10. Attacker steals Graphana credentials, connects to it, and learns more information about the usage of our system.

11. Attacker connects to */metrics* and learns more information about the usage of our system.

12. Attacker steals GitHub credentials and modifies our code to introduce vulnerabilities.

13. Attacker steals GitHub credentials and modifies our CI/CD pipeline to introduce vulnerabilities.

14. Attacker steals GitHub credentials and modifies our code or CI/CD pipeline to execute arbitrary code in our system and download sensitive user data.

15. Attacker steals GitHub credentials and modifies our code or CI/CD pipeline to execute arbitrary code in our system and deliver malware.

16. Attacker steals DigitalOcean credentials and downloads sensitive user data from the DB.

17. Attacker steals DigitalOcean credentials and uses our VM to deliver malware.

18. Attacker steals DB credentials and connects directly to it, due to a firewall misconfiguration.

19. Attacker uses vulnerabilities our the OS or Docker images to connect to the DB and download user data.

20. Attacker uses vulnerabilities our the OS or Docker images to connect to the VM and uses it to deliver malware.

3.4.2 Risk Analysis

Risk Matrix

A risk matrix helps visualize the possible risks in terms of impact and probability. We classified the above scenarios according to their likehood

and probability on the following matrix. Note that the numbers on the matrix correspond to the numbers on the list of risk scenarios above.

	High Impact	Medium Impact	Low Impact
Low Likelihood	8,9,12,13,14,15,16,17,18	-	-
Medium Likelihood	19,20	1,2,3,4,5	10
High Likelihood	-	6,7	11

Steps to take

In order to minimize our risk, we are going to take the following steps:

- Encrypt our SSH key with a passphrase in order to make it more difficult to steal.

- Hide Graphana behind a firewall.

- Limit the access to /*metrics*.

- Use two-factor-authentication in all GitHub accounts of the team members.

- Use two-factor-authentication in all DigitalOcean accounts of the team members.

- Continue using best practices when developing, regarding SQL injection and XSS attacks.

- Be aware of the dependencies we are using and their potential vulnerabilities, and keep them updated.

3.4.3 Steps taken

We encrypted our SSH key with a strong passphrase, which teams members keep separate from the key. Most GitHub and DigitalOcean accounts already had 2FA enabled, and we enabled it in all the remaining ones. We decided against hiding Grafana behind a firewall since it needs to be access by third parties, namely the professors.

3.5 AI-Systems used during the project

3.5.1 Reflection on ChatGPT

People would assume that using AI like ChatGPT would make developing software much easier and faster, which we think is only partially true. A

positive aspect of using ChatGPT is for example it assists very well when learning a new programming language. It is much faster to learn the right syntax by asking ChatGPT rather than googling. It is also very good at explaining code, which makes it easier to understand what other people did. Another positive use is for fixing simple errors by just copying the code with the error message. Usually it resolves the mistake pretty good which also safes some time then.

The problem is when there are more complex errors, especially when it is between two or more systems. ChatGPT makes a first guess and then tends to try to find solutions within the area of the first suggestion. This on the other hand can lead to lose a lot of time because the fix of the problem lies somewhere completely else. An example for this was the implementation of our e2e tests with Cypress. When mounting the docker for cypress we had an error that it could not find the test files, which were in the correct directory. ChatGPT suggested that the volume mounting was wrong and kept suggesting to fix this issue within our compose file. However after some quick research on stackoverflow it was clear that the naming of the test files were wrong, which then could be fixed pretty quick.

As a conclusion, the use of ai-systems can safe time and make development efficient, if used in the right places. When ChatGPT doesn't give the correct answer within the first tries it is a good advice to stop using it and try to resolve the issue by your self.

Chapter 4

Lessons Learned

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.

Chapter 5

Reflection and conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed efficitur nunc non tempor pulvinar. Quisque congue orci non porta feugiat. Morbi vitae nulla libero. Nulla a sem diam. Pellentesque ullamcorper, est sed posuere eleifend, ante elit imperdiet lorem, ac scelerisque lacus ante id ipsum. Vestibulum ac metus tempor sem mattis ornare at eget massa. Sed ut aliquam lorem.