6. DevOps Implementation

Version Control Practices

Git Workflow Git is an open-source version control system that can be used between developers to help manage and track changes in their repositories. It has become the standard tool that developers use to manage software development projects ranging from small personal projects to huge projects in big organizations. The tool helpås track changes to files over time, allows several developers to collaborate on the same project while ensuring that they dont accidentally overwrite each others code, and helps provide a history of changes which developers can also use to revert to previous iterations of the same file if problems arise. Git is also being used in this project. Currently the project consists of three repositories. One for the frontend, backend and the report. In each of these repositories each group member has their own branch in which they can make changes without interfering with the other group members work. When a group member has then fully developed or written something, it can be merged into the dev branch.

GitHub Integration Github is an extension to the git version control system, that also serves as a hosting platform for git repositories. It comes with alot of features that can aid the collaboration between developers to a great extend. For this project, a github organization has been made. That way the group can have their three repositories in a shared space. To each repository the group also has a work board, where tasks can be created such that each group member can see what needs to be done. This helps create an overview of the project status. When group members have developed or written something, they can create a pull request, "asking" for their branch to be merged into a different branch. It is essentially an invitation for the rest of the group to review what has been committed to the repository before merging it into the development branch. This is a way for developers to collaborate and catch errors before they reach a shared branch. Pull requests can also be used to setup continious integration and continous deployment.

Continuous Integration/Continuous Deployment

GitHub Actions Configuration

Build Server Setup

Testing Pipeline For the backend of the project, we're using a GitHub Actions workflow as a testing pipeline. The workflow is triggered on push events as well as pull request events into the dev and main branch. The **build** part of the workflow has steps included to set up the JDK, generating fake JWT keys and building the project using Maven. Following that there is a **test** job which depends on the build job. This means that the test job will not run if the build job fails. If the build job has succeeded, It will download the build artifacts, set the JWT key environment variables, and run the tests for the backend using Maven. This pipeline runs, as said previously, every time a pull request is made, making it super useful, as it prevents any pull requests that do not either build or pass all the tests from being merged into the dev or main branch. For the frontend we have a similar pipeline that first installs the dependencies using **npm run ci**, builds the solution using **npm run build**, and runs the tests using **npm run test**.

Containerization Containerization is a lightweight form of virtualization that packages an application and its dependencies into a single, portable unit called a container. This approach allows our applications to run consistently across various environments, from development to production, without being affected by differences in underlying infrastructure.

Docker Implementation Docker is used to host and build the final application that Quarkus builds, both for local development and cloud deployment. This approach ensures that the production environment closely resembles the development environment, minimizing discrepancies and potential issues.

By using Docker, we can create a consistent and isolated environment for our application, which simplifies the deployment process and enhances reliability across different stages of the softwares lifecycle.

Container Registry For hosting the Docker container in the cloud, we utilize Google Cloud's Artifact Registry. This service provides a centralized location for storing and managing build artifacts and dependencies, which is crucial for maintaining a streamlined and integrated development workflow.

Given that our final deployment is hosted on Google Cloud Run, using Artifact Registry aligns perfectly with our infrastructure, ensuring seamless integration and efficient management of our container images.

Cloud Deployment Cloud deployment involves hosting applications and services on cloud infrastructure, offering scalable, flexible, and efficient resource management. By leveraging cloud platforms, we can deploy our applications across global data centers, ensuring high availability and performance. This approach reduces the need for on-premises hardware, enabling us to focus on development and innovation rather than managing infrastructures.

By deploying applications closer to users through global data centers, we can reduce latency and enhance user experience. This global reach, combined with the reliability and security of cloud services, makes cloud deployment an ideal choice for us when looking to expand our reach in the market.

Google Cloud Backend Deployment As mentioned in the Container Registry chapter, the deployment of the backend is hosted on the Google Cloud Platform (GCP). Why did we chose Google Cloud over Microsoft Azure, especially since DTU provides students with an Azure subscription. Initially, we attempted to set up our cloud infrastructure on Azure but quickly encountered authentication issues when integrating our GitHub Actions workflows.

A runner/account in Azure's IAM is required, but creating one requires access to Microsoft Entra ID, formerly known as Azure Active Directory, which students don't have. Instead of dealing with this complexity and risking losing access after further development, we decided to use Google Cloud, which offers us full control over the entire cloud environment. And it is free to use

In our cloud setup, we have the following elements: ###### Google Cloud Platform (GCP) Components - **SQL Instances**: Host our Quarkus PostgreSQL database instance. - **Artifact Registry**: Store our Docker containers. - **Cloud Run**: Deploy our containerized application and expose it to the web. - **Identity and Access Management (IAM)**: Manage principals and service accounts with the proper access levels, ensuring adherence to best practices for cybersecurity.

Github (GH) Components

- **Security - Secrets and Variables**: Store API keys and credentials securely using Repository Secrets. We have chosen to store our secrets in GitHub to separate our configuration from our deployment.

In summary, choosing Google Cloud Platform for our backend deployment has provided us with greater control and integration capabilities, overcoming the limitations we faced with Azure. This setup ensures a secure and scalable environment, supporting our development and deployment needs effectively. The pay-as-you-go model of GCP services eliminates the need for significant upfront investments, making it a cost-effective solution. Many of the services we utilize are free of charge under the chosen plans, allowing us to optimize our budget while leveraging advanced cloud capabilities.

Netlify Frontend Deployment In our project, we utilize Netlify as the platform for hosting and deploying our frontend application. Netlify offers an automated and seamless deployment workflow by integrating directly with our GitHub repository. Every time changes are pushed to the main branch, Netlify automatically pulls the latest code, runs the specified build command (npm run build), and deploys the application to its globally distributed Content Delivery Network (CDN).

As part of this automated process, Netlify provides real-time feedback on the deployment status. If the npm run build command succeeds, the application is immediately deployed, and the live site is updated. If the build fails, Netlify generates detailed logs, making it easy to identify and resolve any issues. After deployment, Netlify confirms whether the site was successfully deployed, ensuring full transparency in the deployment process.

In addition, for every pull request, Netlify runs a Lighthouse audit to evaluate the performance, accessibility, best practices, and SEO of the site. This feature provides a clear score and actionable insights, helping us continuously optimize the application before merging changes into the main branch. By catching potential issues at the pull request level, we ensure that only high-quality updates make it into production.

We have also implemented GitHub Actions as part of our CI/CD pipeline. These actions complement Netlify by automating additional checks, such as running unit tests, linting, and verifying configurations before deployment. GitHub Actions provide an extra layer of validation, ensuring that the codebase adheres to predefined quality standards and that potential issues are identified early in the development cycle.

Netlify's integration with GitHub and the use of Github Actions significantly streamline our deployment process. Features like automatic builds, deployment status notifications, and Lighthouse audits on pull requests, we are able to maintain a high-quality frontend application. Its global CDN ensures that the deployed site loads quickly for users worldwide, further enhancing the user experience. By leveraging Netlify, we have a reliable, efficient, and developer-friendly solution for managing our frontend deployment pipeline.

Another of the key advantages of using Netlify is that it offers its core features for free, including automated builds, deployments, and a globally distributed CDN. This makes it an excellent choice for small teams or projects, allowing us to deploy a high-quality frontend application without incurring additional costs.

Monitoring and Maintenance A key principle in DevOps is the ability to act based on metrics. To do this we use Micrometer metrics, a common monitoring facade which is vendor neutral, alike SLF4J is for logging. We use the Prometheus format as Quarkus provides a convenient library for combining these, `quarkus-micrometer-registry-prometheus`. To display and view these metrics we use Dashbuilder which allows easy visualization of the metric data via a YML format. Quarkus also has minimal config library for this, named `quarkus-dashbuilder`.

We have also looked into setting up a centralized logging system, but due to time constraints were unable to implement it in time for the writing of this report.

Another clear improvement is monitoring if our frontend server is up or overloaded.