**DevOps Document**

*S6 Software Engineering*

*4207734*

*Matei-Cristian Mitran*

*Fontys Eindhoven*

*08.03.2023*

**Table of Contents**

1. **Introduction1**
2. **CI/CD Pipeline3**
   1. Git ………………………………………………………………………………………………………………………………………………**3**
   2. Continuous Integration**4**
3. **Tools4**

**1. Introduction**

This document will detail the Continuous Integration and Continuous Delivery pipeline and the technologies used in the software development process. This document provides the approach of implementing CI/CD in the project YouSound. The correct use of this pipeline will deliver high quality software more efficiently and safely. Moreover, this document will provide an in-depth look at the various components of the CI/CD pipeline, including the tools and technologies used to automate the process of software development, deployment, and testing.

**2. CI/CD Pipeline**

# 2.1 Git

Git is a very popular version control system that I use for source code management in my project. Its distribution provides accurate tracking in changes to the codebase over time. It is a valuable tool for ensuring code quality and maintaining an organized codebase.

# 2.2 Continuous Integration

This project has Continuous Integration provided by GitHub Workflows, automating the approach to building, testing, and deploying code changes.

By automating these processes, errors are caught earlier in the development cycle, maximizing efficiency.

This pipeline also runs a security test using OWASP ZAP and a SonarCloud quality assurance test.

# 2.3 Continuous Deployment

In the testing branch, after building and testing the project, a docker image is built for every microservice and the frontend and then pushed to dockerhub. Then, it authenticates with GCP and uses SSH to access a GCP VM running on Ubuntu 20.4 LTS. It runs commands to stop the docker compose containers, then it runs docker compose pull to pull the latest images from dockerhub and then finally, runs docker compose up -d to start the containers with no output in the pipeline.

In the master branch, for the last sprint, Kubernetes deployment will be added to the pipeline.

**3. Tools**

* GitHub
* Docker
* Kubernetes

**Dockerizing and Orchestration**

To make the deployment process more efficient, I decided to dockerize the microservices. This was done by adding a Dockerfile for each microservice and building them into images. I also added a docker-compose file in the root folder to orchestrate the startup process of all the microservices. With this, we can start all the microservices with a single command and specify the order in which they should start.

**Cloud Deployment**

For the database, we chose to deploy a VM instance with an Ubuntu 20.4 OS and MariaDB server installed on it using Google Cloud Platform. We configured the VM instance and moved the databases onto it, removing the need to have it locally. Additionally, we deployed the user database and the community database on Mongo Atlas Cloud, which provides a managed and scalable database service.

**Kubernetes Deployment**

For scalability and resilience, we deployed two microservices, User Service and Music Service, on Google Kubernetes Engine. We created deployments and services for each microservice and configured them to auto scale depending on the needs of the system. We also tested the endpoints to ensure that they were working as expected. To monitor and troubleshoot the microservices, we added Zipkin to the Kubernetes cluster. Zipkin provides distributed tracing, which allows us to trace HTTP requests across multiple microservices. With this, we can identify bottlenecks and diagnose issues that may arise.