1 Introduction

The aim of this document is to provide an overview of the structure of the *CI/CD* pipeline that is currently deployed on my machine, as well as to explore the technical challenges that arose during the setup, as well as their respective solutions.

2 Docker Components

The pipeline consists of the following components, each one of which is running in a designated docker container:

1. Gitea
2. PostgreSQL *(Part of Gitea)*
3. Jenkins
4. Reposilite

All of these components are instantiated form a single docker compose file. Every component is connected to the same network ('`cicd`').

2.1 Jenkins

Although the setup for the Jenkins container is not particularly sophisticated, is is the most complex one of the bunch. Since the container needs to access the Docker engine of the host machine, it requires both the Docker socket, as well as a docker client to be avilable. The former is solved by mounting the Unix socket as usual:

```
volumes:
    ...
    - /var/run/docker.sock:/var/run/docker.sock
```

To approach the latter, I opted to build a custom Jenkins image that derives from the original `jenkins/jenkins`, whilst including the docker client from the `docker:dind` image. The aforementioned concept maps into the following Dockerfile:

```
FROM jenkins/jenkins
USER root

# Grab only the docker client from the DinD image
COPY --from=docker:dind /usr/local/bin/docker /usr/local/bin/

USER jenkins
```

Said image is then built and used as the container image for the Jenkins service, i.e.:

```
docker build . -t jenkins-with-docker
```

```
...
jenkins:
    container_name: cicd-jenkins
    networks:
        - cicd
    image: jenkins-with-docker
    privileged: true
    ...
```

Also note that the Jenkins container is created as a **privileged** container. This is to allow the container to use the mounted socket for communicating with the host device.

2.2 Reposilite

Contrary to the former, **Reposilite** did not require any amount of problem-solving during its setup. It is, however, worth mentioning, that the following environment variable was set:

```
environment:
    - REPOSILITE_OPTS=--token admin:secret
```

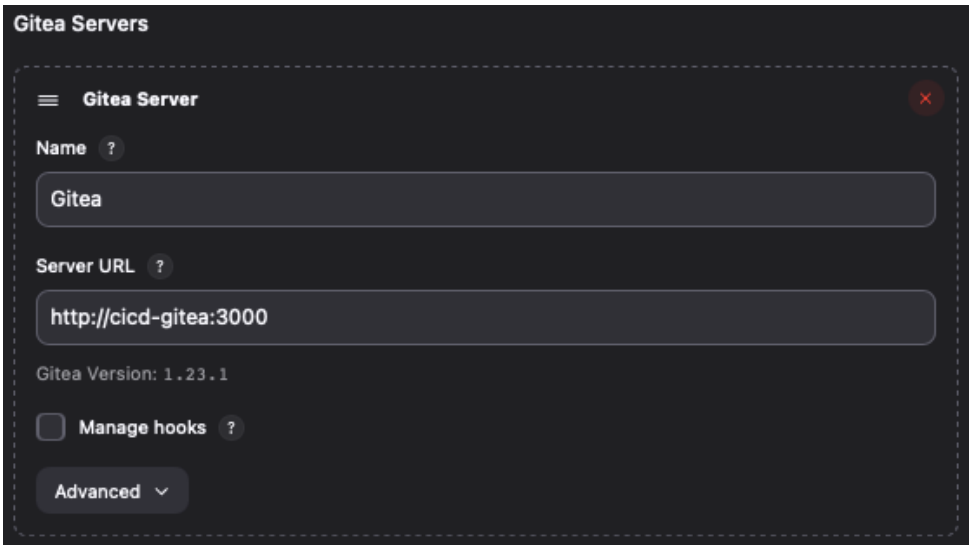
This creates a user `admin`, with the password `secret`.

3 Gitea Config

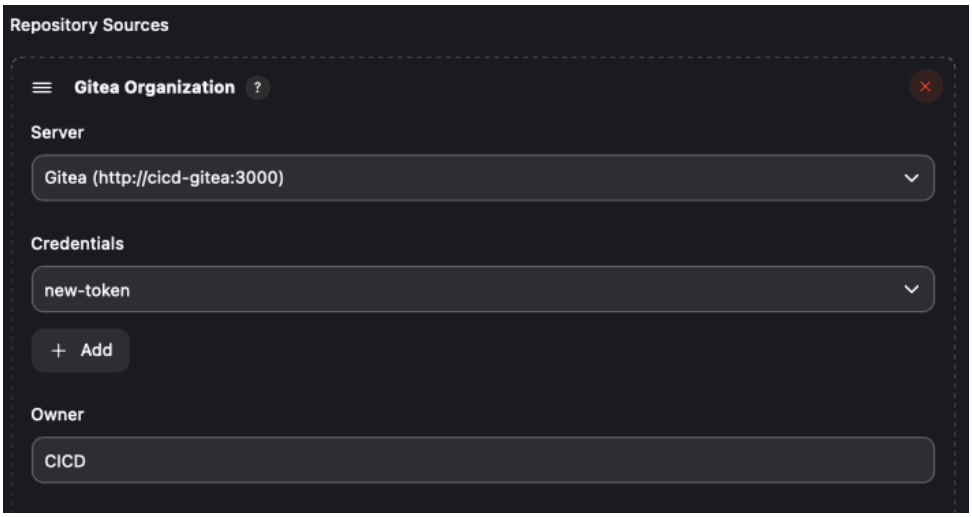
Gitea is configured with a `jenkins` user that is part of a `CICD` organisation. An access token with all read & write permissions was generated for the aforementioned user.

4 Jenkins Config

The `gitea` plugin was installed on Jenkins and was subsequently configured.



Thereafter, a `Gitea` Jenkins item was created. In its configuration section, the repository sources were configured to point at the aforementioned `CICD` organisation. This is also where the access token of the `jenkins` user was supplied.



This configuration allows Jenkins to detect changes on Gitea repositories in the designated organisation. Although this config does the job just fine, and lets the Jenkins instance build codebases through the included `Jenskinsfile`s, it does not allow us to specify a custom (dockerized) environment for each build. To change this, the **Docker plugin** and the **Docker pipeline** plugins were installed. This, together with the previous docker-level configuration of the Jenkins container, allows it to instantiate a new container for each build. This way, the developer may specify a prefered docker image that is shipped with the required build environment, such as the one needed to, among others, build a *Spring Boot* application ...

```
...
pipeline {
    agent {
        docker {
            image 'openjdk:17-slim'
            args '--network=bap_cicd'
        }
    }
    ...
```

5 Reposilite

The artifact repository, **Reposilite**, came with what was by far the least convoluted setup procedure. In fact, it on its own did not require any configuration whatsoever.
Setting up **Gradle** to publish artifacts to reposilite's *releases* repo included setting up a publication as follows:

```
publishing {
    publications {
        create<MavenPublication>("reposilitePublication") {
            from(components["kotlin"])
        }
    }
    repositories {
        maven {
            url = uri("http://reposilite:8080/releases")
            credentials {
                isAllowInsecureProtocol = true
                username = "admin"
                password = "secret"
            }
        }
    }
}
```

Note that we need to explicitly allow the use of HTTP (instead of HTTPS) through `isAllowInsecureProtocol`. This gradle target is then invoked as the last step of the Jenkins pipeline:

```
...
stages {
    stage('Build') {
        ...
    }

    stage('Publish Artifacts') {
        steps {
            script {
                sh "./gradlew publish"
            }
        }
    }
}
```

Upon invoking the pipeline, the artifacts become available in the repository: