# Submission 2

## Tech Report – DevOps

There are many different parts of the DevOps in our project, however not all of them are going to be useful to most people, so I’m only going to go over the main parts, the CI/CD pipeline, and the Virtual Machine.

### CI/CD Pipeline

CI/CD stands for Continuous Integration/Continuous Deployment, and it basically allows the developers of a project to have checks to ensure that when a commit is made, it doesn’t break the application, and that all working commits are immediately deployed so that the most recent working version is the one in deployment.

Our pipeline is all done through the `.gitlab-ci.yml` file, which goes over all the required steps for our app to be deployed and checks that the commit is working. This is done in 5 different stages: check, build, package, release, deploy.

Before any of these stages it first creates a docker image of jhipster:v7.9.4, which runs on top of an ubuntu docker image, in which all the stages are then run on.

#### Check

Check is the first stage in the pipeline, and it is also the shortest. All it does is runs a single command to check if the repository exists in the place it is meant to be in. It does this by running `./mvnw -ntp cheskstyle:check -Dmaven.repo.local=$MAVEN\_USER\_HOME`

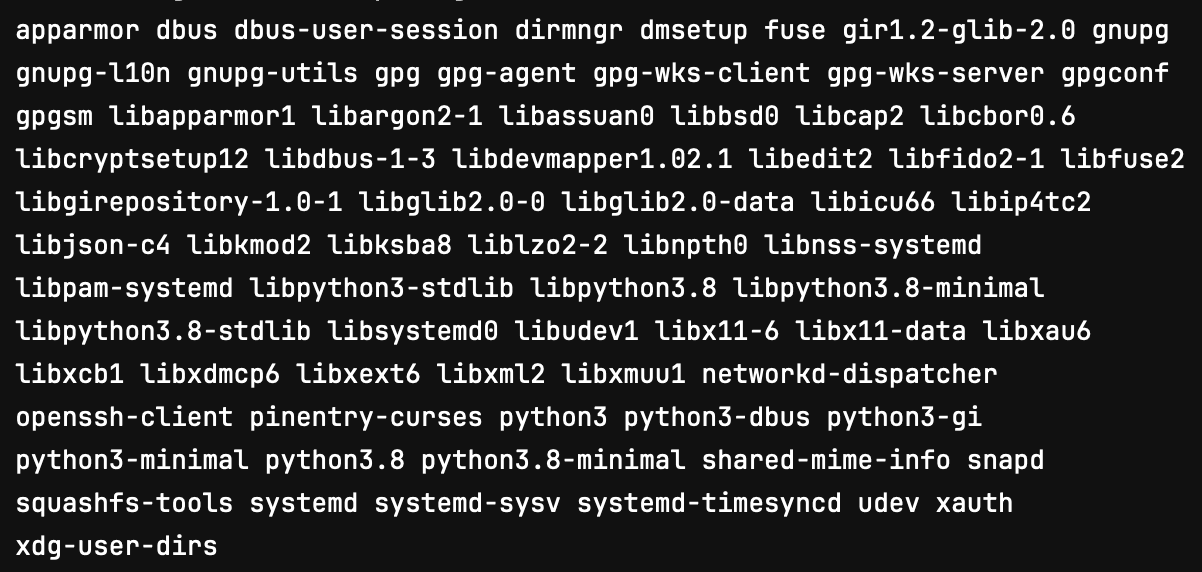
#### Build

After it does its checks to see if the repository exists, it then enters the build stage of the pipeline. In the build stage, the source files are compiled using maven on our Ubuntu VM to prepare us for the next stage. It does this by running `./mvnw -ntp compile -P webapp -Dmaven.repo.local=$MAVEN\_USER\_HOME`

#### Package

After compiling our app, we need to prepare the VM for release. To do this we need to install chromium-browser, which is the basis of most modern browsers. To install chromium, we first need to run `sudo apt-get update`, which checks to see if any installed packages have any updates. Note that this doesn’t update them, it just checks to see if they can be updated. If we tried to just run this, it would fail as you need to input the root password for `sudo`, which is impossible for us to do the normal way. To get around this, we pipe the password into the `sudo apt-get update` command and give it the `-S` flag so that sudo will read the password from the standard input (stdin) instead of from a terminal window. This gives us the final command of `echo “jhipster” | sudo -S apt-get update` where ‘jhipster’ is the root password.

After this, we then need to actually install chromium, which is done by running `sudo apt-get install chromium-browser`, however this encounters the same issues as update so we use the same method to get it to work, but with an added `-y` flag which simply says ‘yes’ to any prompts that occur in the installation process, since we wouldn’t be able to do this manually. Finally, this gives us our final command of `echo “jhipster” | sudo -S apt-get install -y chromium-browser`.

On installation of chromium-browser, these additional packages also get installed.

After this, we verify the results of the integrity tests to ensure that the quality criteria is met. This is done by running `./mvnw -ntp verify -Pprod -DskipTests -Dmaven.repo.local =$MAVEN\_USER\_HOME`

#### Release

The release stage only has one command, but it does a lot, which is building the docker image, so it is ready for release and deployment. It does this by first installing all prerequisites for the project, then it reads through the dockerfile to add the rest of the required dependencies.

#### Deploy

The deploy stage of the pipeline has 2 different branches, one for development and one for production. The pipeline defaults to dev and only pushes to prod when it is a tagged commit on the main branch. To do a tagged commit, run `epolgit tag -a <version> -m <message>` (example: `git tag -a v0.1.1 -m “version 0.1.1”` before you commit and run `git push -u origin main`.

##### deploy-prod

Deploy-prod is the deployment for when we want the commit to be publicly available and accessible, it has all the extra hidden parts removed so we don’t accidentally expose secrets such as the RSA private key. It has deploy-dev as a dependency since it must be able to be deployed to dev first before we can deploy to prod.

To deploy, before we run the deploy script we need to install and configure SSH on the gitlab runner. /to do this we first change the permissions of the base of the app by using `chmod og=$RSA`. This changes the ‘owner’ and ‘group’ permission to be the same as the private RSA key file. After this we need to install the openssh-client so that we can ssh into the vm. After this we can deploy the app by running the docker image.

##### deploy-dev

To deploy to dev, it is the same as prod but with some added steps to setup the deployment server, so it is ready for deployment. This mostly consists of transferring the docker folder onto the deployment server and running `install-app.sh`, which removes the previous version of the app, loads environmental variables, and pulls and configures the docker image.

## Subset of UML Diagrams

### UML Diagram generated from JDL code

A diagram of a student

Description automatically generated

### JDL code

entity Student {  
 studentID Integer  
 firstName String  
 lastName String  
 yearGroup Integer  
 formID Integer  
 profilePicture ImageBlob  
}

entity Teacher {  
 teacherID Integer  
 firstName String  
 lastName String  
 accessRights String  
}  
  
entity Class {  
 classID Integer  
 teacherID Integer  
 classroomID Integer  
 subject String  
 capacity Integer  
}  
  
entity Classroom {  
 classroomID Integer  
 classroomName String  
}  
  
entity Form {  
 formID Integer  
 formName String  
 classRoomID  
 teacherID Integer  
}

entity StudentClass {  
 studentID Integer  
 classID Integer  
}  
  
entity StudentForm {  
 studentID Integer  
 formID Integer  
}  
  
relationship *OneToMany* {  
 Student to StudentClass  
 Teacher to Class  
 Class to StudentClass  
 Classroom to Class  
 Form to StudentForm  
}  
  
relationship *OneToOne* {  
 Student to StudentForm  
 Form to Teacher  
 Form to Classroom  
}

## Kanban Feature Cards

I have 4 kanban feature cards here, all of which are labelled, commented, timestamped with milestones

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## Timesheets

A screenshot of a data

Description automatically generatedHere is my timesheet for the dates 14/02 until 27/02, which encompasses all work done for this submission.