Practice 1

Task 1:

We have chosen a project that we have used in our Software-Design course because of its MVC approach.

Here we would be able to implement testing in a meaningful manner.

The Project properties:

SDK: OpenJDK version 17.0.3

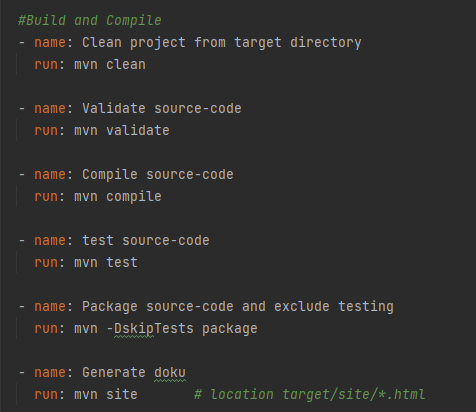
Language level: 17-Sealed types, always-strict floating-point semantics

Build tool: Maven

First request: Integrate 2 - 3 Github actions:

We have devided our Github-workfow into two steps that are executed one after another (serial).

In the first step we are going to:



* Validate
* Compile
* Test (Unit-test)
* Package (without testing)
* Site (generate doku)

The maven commands are executed as in a waterfall procedure, means, that every step is executing all previous steps again, when started.

Why are we doing this?

We wanted to be clear where our application fails and therefore reduce the log output from one step to a minimum.

It is clear that we could do step 1 (validate) till step 4 (packaging) with just one call, but this would defeat the purpose of our approach of trying to divide the steps into small junks for the overview.

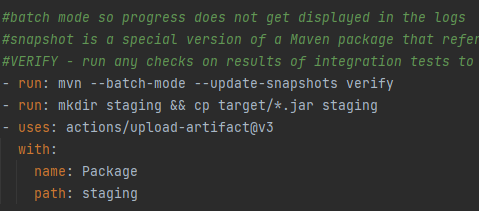
Actions: The actions we uses are upload and download. Picture below shows upload.

With this action we are able to upload our artifacts onto Github and store them.

Package is the name of the artifact.

Staging the path to the arifact.

The download action is almost the same so we are not mentioning it here



Second request: How could CI/CD pipelines help to avoid Anti-Patterns?

1. Infrequent Check in:

After a code change we have to stage our changes and commit them. After doing this we are able to push our changes to the remote repository in order to integrate them to a development branch, as example. After this push the pipeline gets triggered and will provide some feedback about the branch and if our code has broken the attempt of building the branch.

FAIL-SUCCESS

1. IDE-only build:

This is a antipattern which we were also confronted with in our companies due to the developers using different IDEs for development.

With pushing to the remote repo we are establishing an independent environment on which the code has to work and we don’t care if building with the devs IDE works.

As hint: Some IDEs are supporting features that support the developer with building the application like installing maven plugins automatically for example.

1. Works on my machine:

Is similar to the IDE-only example but here we are abstracting further. We want an environment that has only the necessary things installed in order to make the application buildable. Means SDK, Build-tool, Plugins.

Our pipeline image will just have as little as possible configured in order to provide a stable – consistent environment that is supposed to work on every machine.

1. Broken Build:

For me this last point that we want to mention is also the most important of the four points.

Never allow Borken builds in a development environment. This just leads to stacking up the “shit” what you were building. Sorry for the language but after experiencing such approach we never ever want to go back. The easy reason is that you can not build reliable software if the base on which you are building is not in tact!

Third request: How far can CI be executed in with this exercise?

We can cover the whole CI process if we wanted to. Underneath the CI attributes we consider necessary:

Build – Test – Artifact

We build the application with Maven.

We tested the application with a Maven plugin called Junit Jupiter.

We are providing and uploading a artifact to Github.

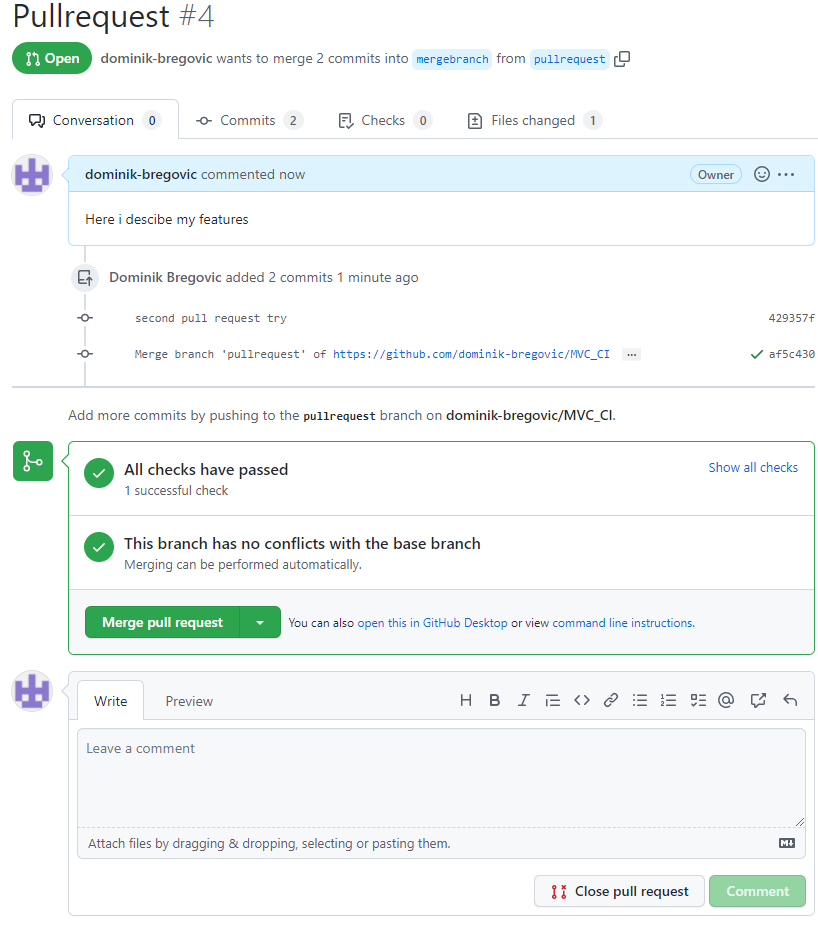
Process.md:

Inside of the Process.md file we are describing every step while we tried to implement it. Means if you want to get an overview of how things went then please go to.

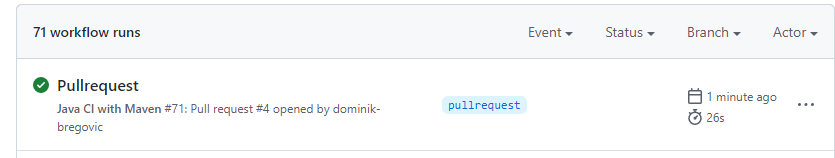
[MVC\_CI/Process.md at main · dominik-bregovic/MVC\_CI (github.com)](https://github.com/dominik-bregovic/MVC_CI/blob/main/Process.md)

Task 2:

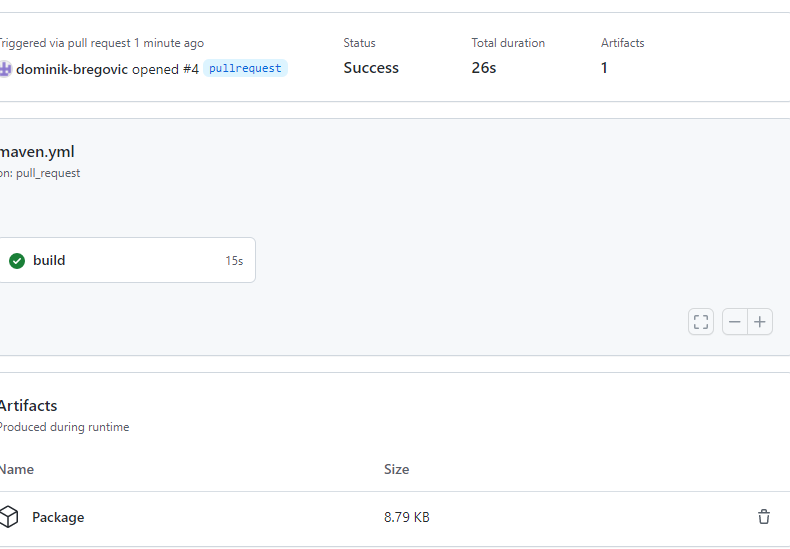
Like task 2 expects form us we made a branch named “pullrequest” in order to provide a separate branch with which we will start a pullrequest and merge it into the main branch but we will use a mergebranch I order to leave production and the yml file untouched.



Action executes after merging:



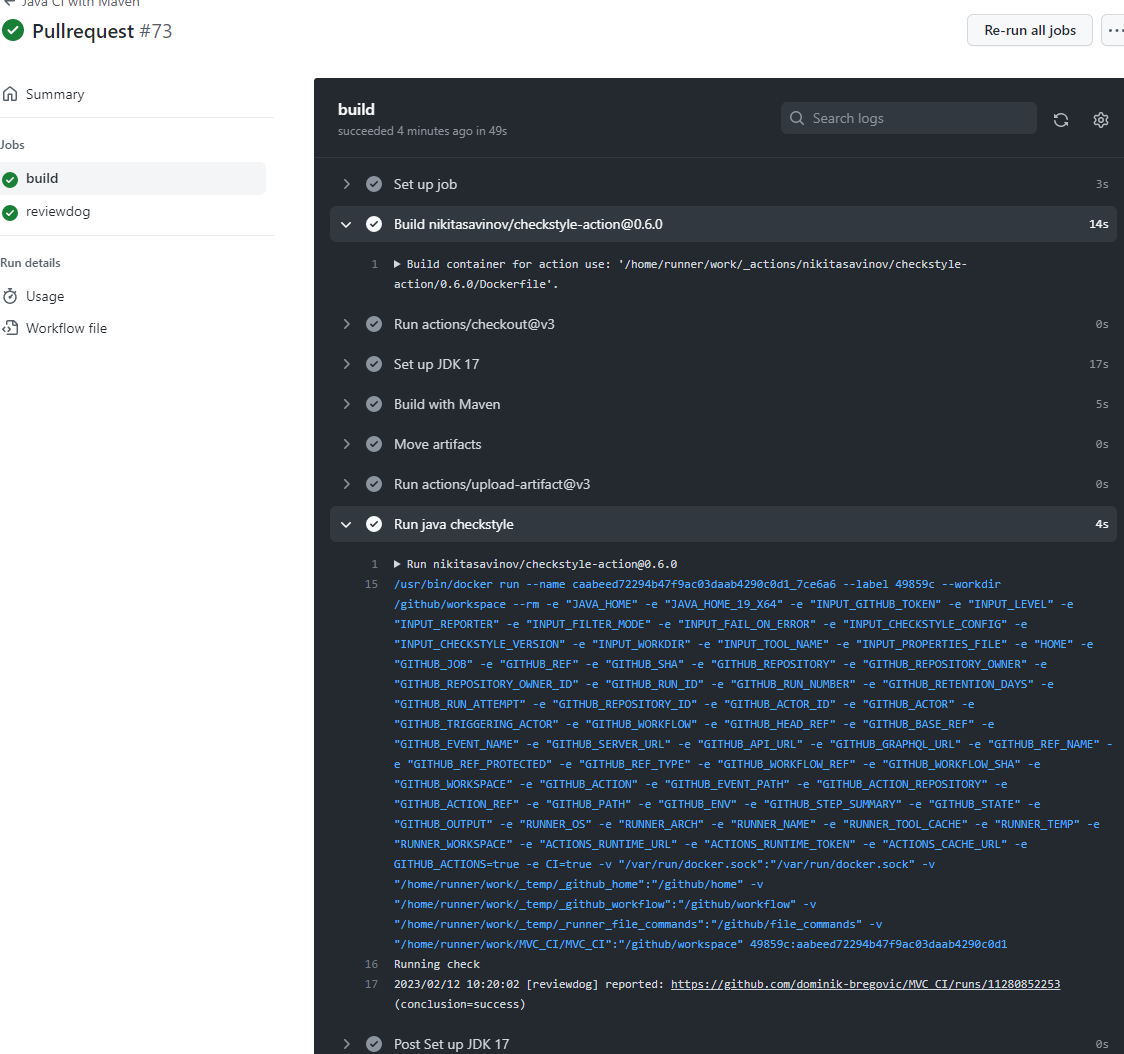
Upload of the package:



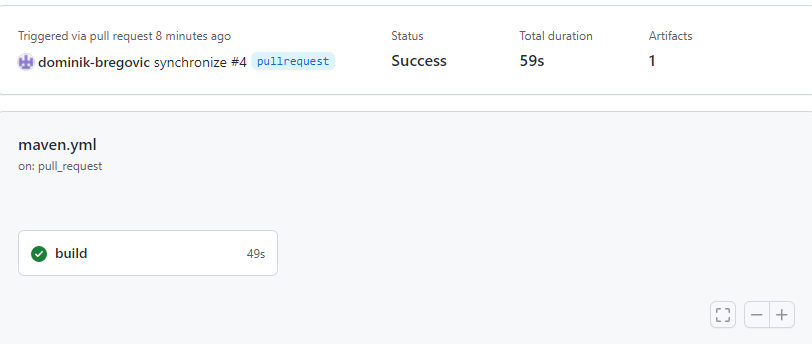
Task 3:

In task 3 we will add the checkstyle action. We are navigating to the marketplace and use the codesnippet nikitasavinov/checkstyle-action@0.6.0 .

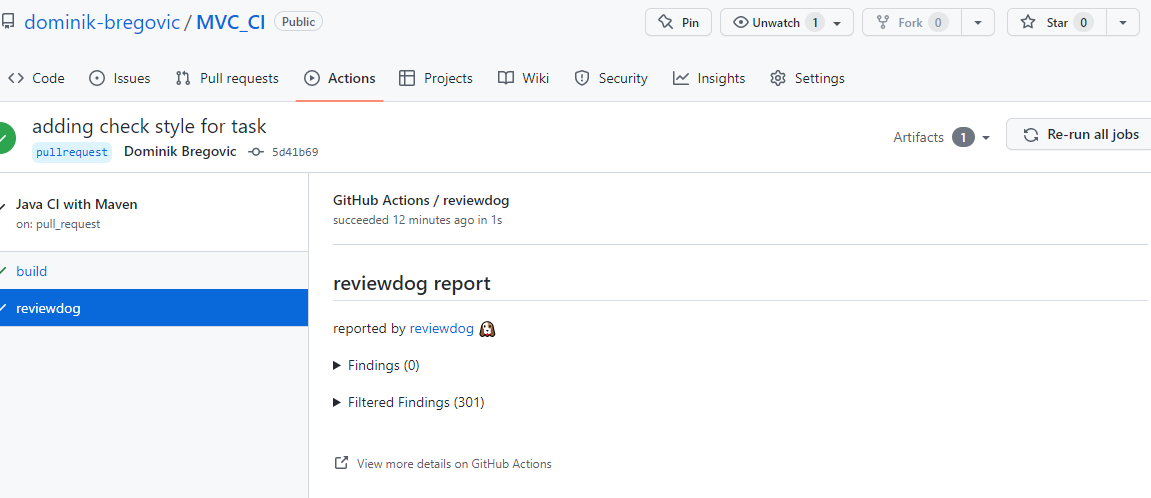
The picture below show the action executed:



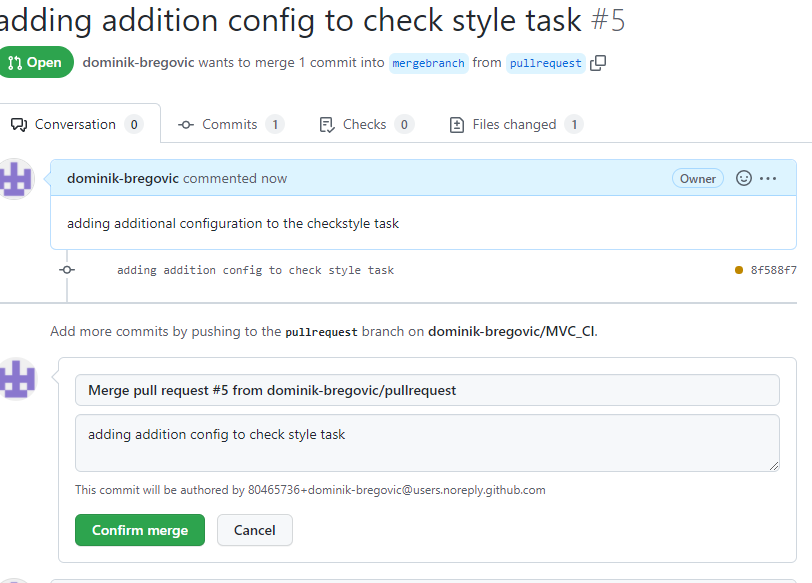
As expected have we implemented this merge with a pull request:



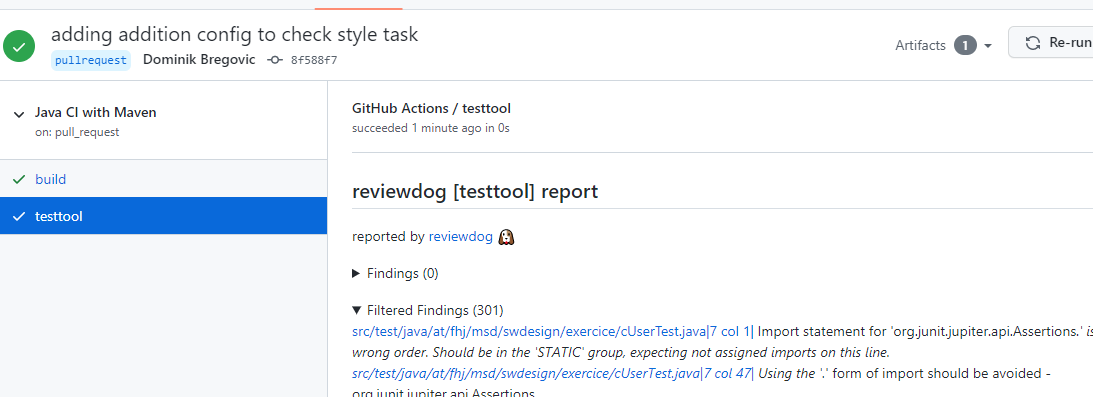
The review dog output which we can find in the build task under the Run java-checkstyle task.



As always adding the configuration via a pull request:



We don’t have any finding so we don’t fail in our case:



Task 4:

When do workflows get triggered?

On push

On pullrequest

When does it make sense to trigger a workflow?

Everytime we want to change code in our repository.

We want to see if changes break the application.

Which steps can be automated?

Building

Testing

Analysing

Artifact upload

What are the different OS systems in runners good for?

With different OS systems we can provide more functionality and are so possible to build our environment similar to the production environment for example. We want to keep everything as easy as possible and runners are additionally giving us the possibility to have more resources like more Ram, CPU speed and cores and more memory while executing a workflow.

How are workflow build?

Workflows do usually have an image of choice at the top of the yaml, then the triggers on which it should execute and then the Jobs followed by the specific steps. With the jobs we can also specify parallel or serial execution.

How can we automate the testing of different java versions inside of a workflow?

For this we can have many different approaches like having multiple branches with specific configuration regarding the version.

We could also have more repositories or just more yaml files.

Though the easiest would be with branches form my point of view.

Practice 2:

Task 1:

Docker installation:

The docker installation was easier for me on Linux Mint. I just had update and upgrade before installing docker with the sudo apt install docker\* and I was ready to go.

My setup:

Windows 10

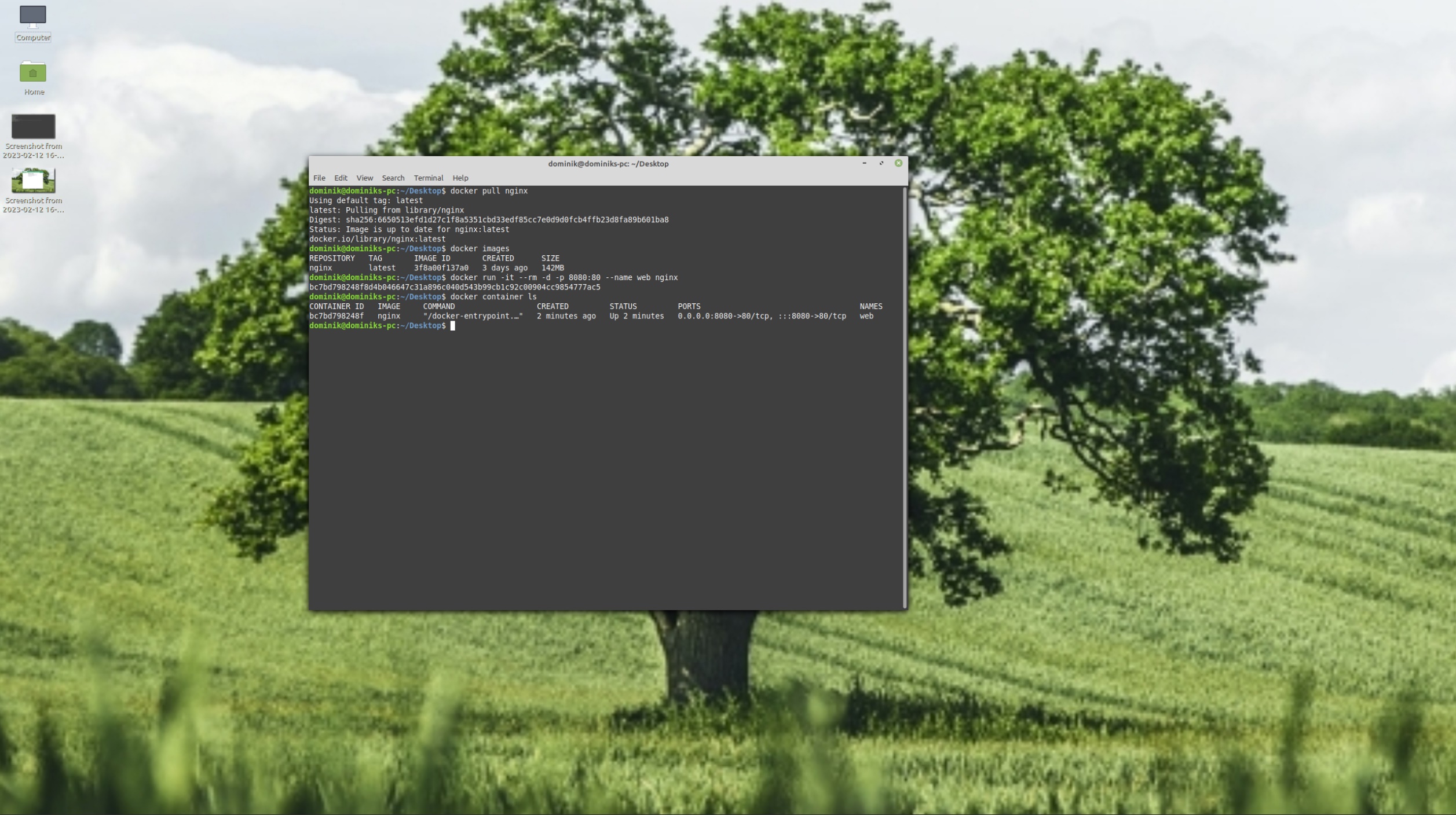
Virtual Box as Hypervizer 2

Installed Linux mint Mate

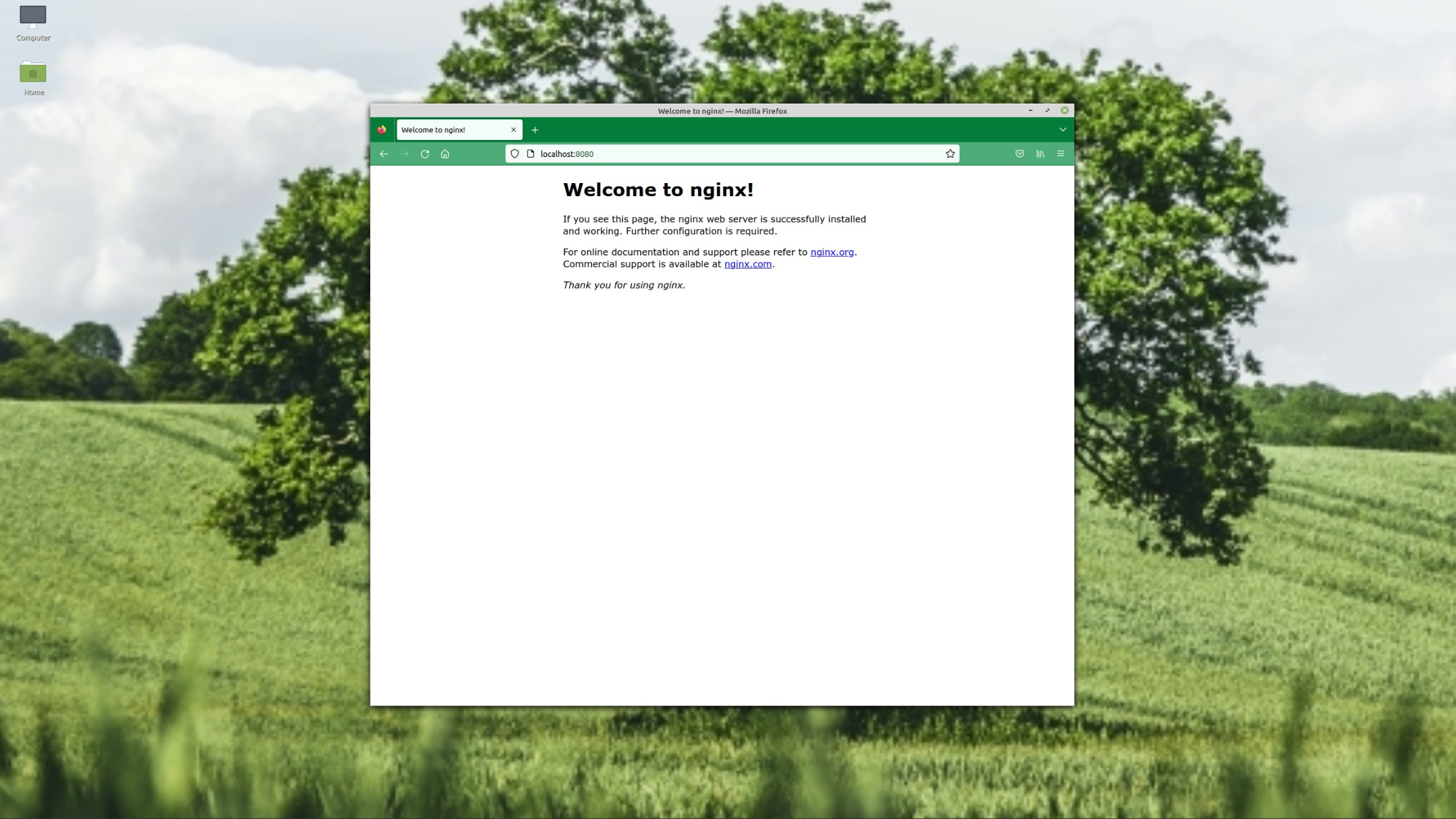
Installed docker sudo apt install docker\*

Task 2:

After the installation I executed all the commands in order and got ngnix running on localhost

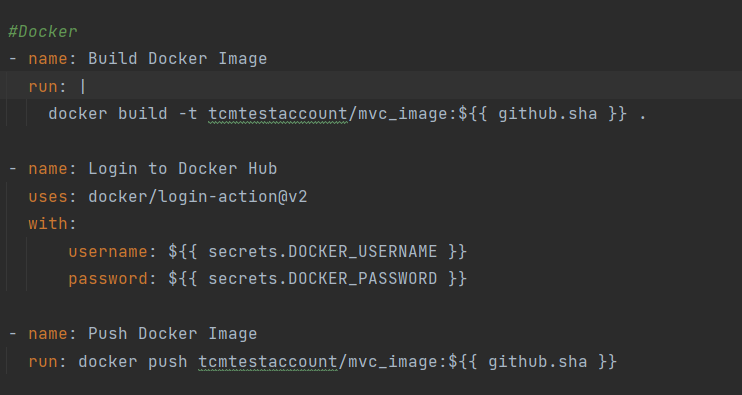


Application running on local host 8080 in browser:

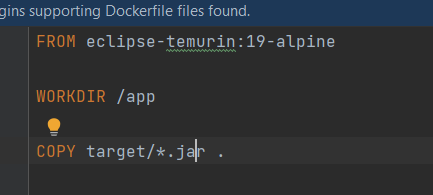


Task3:

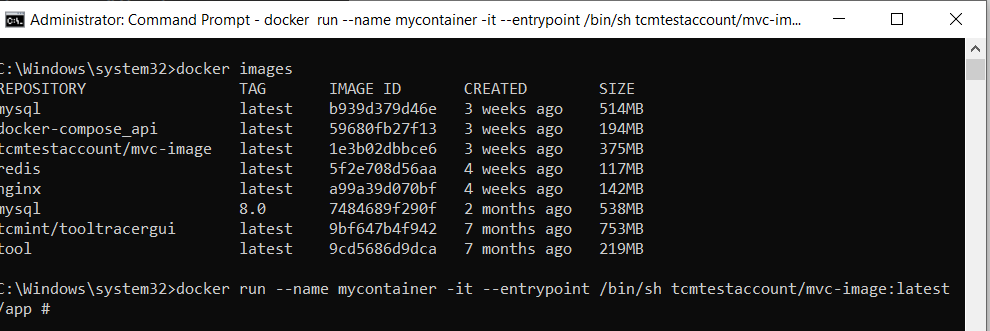
For Practice 2 I continued using my MVC application in which I have just added the wished behavior.



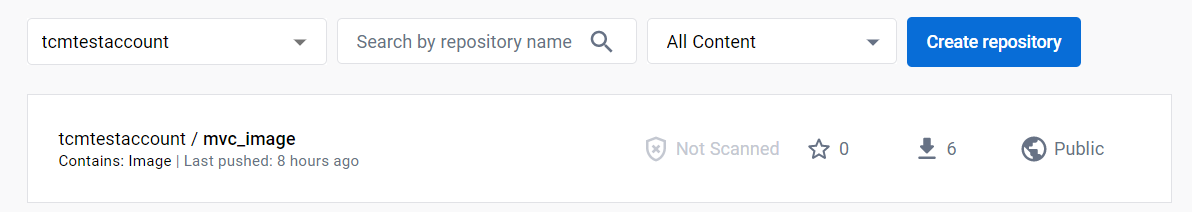
Dockerfile:



Container run and executed shell:

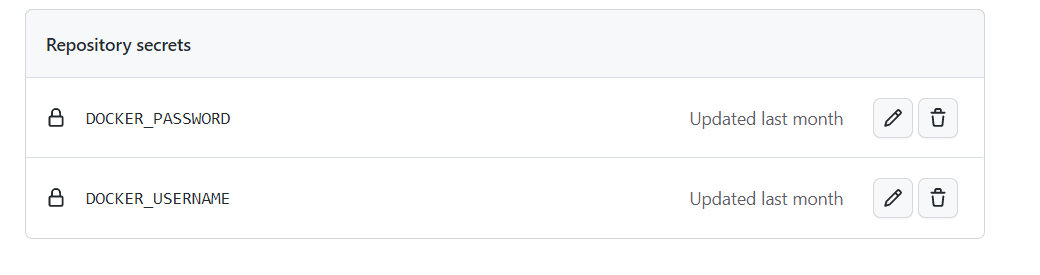


Container uploaded do hub.docker:



Extend pipeline:

As seen in the picture above, already done and login params are saved as secrets.



Process.md

More information about the project and how the implementation went can be found in the Process.md file in the MVC repository.

Practice 3:

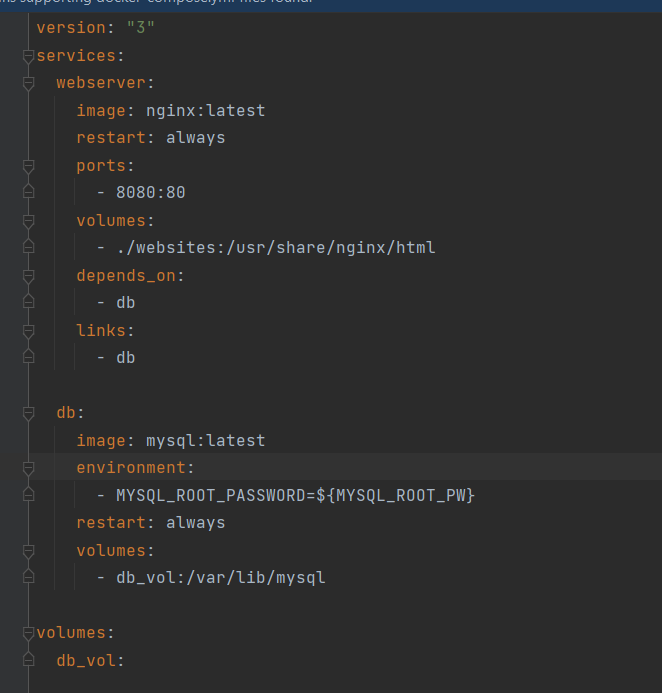
Task 1:

In this practice we continued with the MVC project.

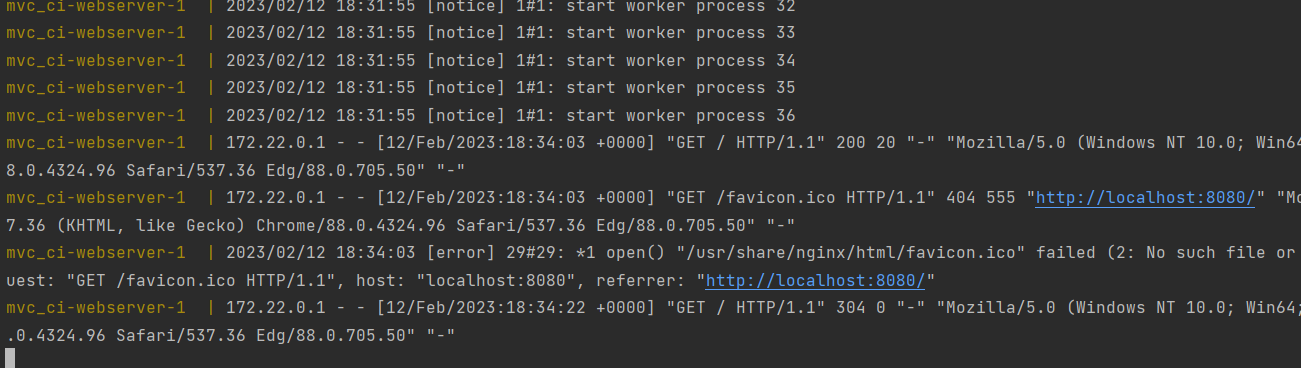
We wanted to run two images at once.

One being the nginx server that serves the application

Second the db that gets started first



Started/docker compose up:



Description:

The docker-compose.yml file is basically like the docker run command but the yml file allows a structured view on the parameters we want to include. Additionally to that is it possible to add networks or link containers to one another.

We are starting the nginx server in order to serve our application.

With the depends\_on keyword we can tell docker-compose to first start the mysql image and after that the ngnix server.

The links attribute is used to link the two containers to each other so that they get listed as known hosts.

In the docker-compose.yml we should avoid entering passphrases.

Better store them into an .env file. Inside of this file it is possible to declare variable which can then be used in the docker-compose.yml files as secrets.