Docker integration report

Sam Philipsen | 20-1-2022

Table of contents

[How does docker work? 3](#_Toc93592602)

[Integration into gitlab CI/CD 3](#_Toc93592603)

[CI/CD pipeline configuration 3](#_Toc93592604)

[Docker in the CI/CD pipeline 3](#_Toc93592605)

[How to integrate docker 4](#_Toc93592606)

[Cypress 5](#_Toc93592607)

[Problems with implementation 6](#_Toc93592608)

# How does docker work?

Docker is a tool used to deliver software in packages called containers. Containers are packages that contain their own software and dependencies. You basically ‘dockerize’ your application by making an image of it using a dockerfile, and then running it in a container. This means you can run the application on any operating system without installing anything extra, as all the software and dependencies are included in this image.

# Integration into gitlab CI/CD

Since you can run your application anywhere, you can also run them in gitlab. In gitlab you can run your docker container and run your jobs in them. This way you can test if your application runs after each push. You can also You can also use it to run tests on your program in the container rather than testing it on a dedicated server.

This is also very useful for tools like Cypress, which is used to test websites for intended behavior. Because most websites need a running backend to function normally (like getting information, or logging in), you would need to have that running before testing. With docker, you can fire up your back end in a container so your website runs normally, and you can test it with Cypress.

By then putting this in a gitlab pipeline, you can test your entire application any time even when multiple people are working on it.

# CI/CD pipeline configuration

Getting docker to work and run your tests in the project’s pipeline requires some setting up beforehand. Setting up a CI/CD (continuous integration/continuous delivery) pipeline requires you to first set up a configuration file. This file has instructions for performing jobs that a runner is supposed to perform. These jobs are instructions for building the application and testing the application. By performing these in gitlab itself, it ensures the application works properly every time a new push happens even if another member of the repository pushes it.

The runner(s) perform these jobs. Runners are services on your machine or a server that follow the instructions set up in the configuration file. They build and test the application for you.

# Docker in the CI/CD pipeline

By integrating docker into your CI/CD pipeline you can run your tests in the docker containers which run on the images pulled from the latest push. Now you can test if your build runs and if the tests run in the docker container. This is much faster than the normal CI/CD pipeline since it runs in its own container. Running tests in its own container also enables some other options like testing parts of the application that depend on other parts (like Cypress, further explained later).

# How to integrate docker

Putting docker into a repository’s pipeline requires some setting up. Like in a normal pipeline, you need to set up a configuration file and set up a runner to perform the jobs described in the config file. You also need to set up a dockerfile that has steps to build images of your applications. In the configuration file you again describe the jobs the runner has to perform to test and build the application. The difference here, however, is that the configuration file uses docker commands to build the images and run them. Below is an example of what my repository’s docker configuration would look like.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

In the dockerfile these targets are defined. Building an image that points to a target, means a certain stage will be built as defined in the dockerfile. In this case, separating the backend stage and the frontend stage. This shows what the dockerfile would look like.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

The backend is automatically tested when it starts up, while the frontend needs cypress to run in order to test the functionality.

For the backend, first the environment is chosen with the FROM tag, and the backend flag is put next to it for building it in the gitlab pipeline. After that, the file navigates to the folder where the backend is stored. The port 8080 is exposed since that is what the application uses for communication. The built application jar file is added, and then the application runs. This builds the application and tests it.

For the frontend, the environment is chosen with the FROM tag, and the frontend flag is put next to it for building it in the gitlab pipeline. After that, the file navigates to the folder where the backend is stored. It then makes a folder called ‘app’ where it will copy all the dependencies’ tags into and then installs them. It then runs the application, and the cypress tests alongside it.

Once this is all defined, you will need a runner to pick up these jobs and perform them. The way you install this is similar to a normal runner, there are just a few changes you need to make when registering it. As the executor the runner uses, you need to specify that it should use the docker executor, and for the image you need to specify a dind (docker in docker) version. I used version 19.03.

At last, in the configuration file of the docker runner, you need to make sure privileged = true. If you do not, it will not build your docker files.

Once you have this all set up, you should be able to build and test your applications in docker on gitlab.

# Cypress

Cypress is a tool used to test web applications. However, most websites need a running backend to function. For example, to fetch data. This is where docker helps us. You can make docker run the backend and frontend first, and then run the cypress tests. You can see cypress running in the dockerfile shown above at the bottom, where it says “npm bin”, “cypress run”.

# Problems with implementation

After performing the steps above, I ran into a problem. Running the pipeline would not work. In the console it would display the error:

level=error msg="failed to dial gRPC: cannot connect to the Docker daemon. Is 'docker daemon' running on this host?: dial tcp 172.17.0.2:2375: connect: connection refused"

Searching for a fix online lead to a few possible solutions. Most of these involved changing the pipeline configuration to manually connect to an IP that the Docker daemon should be using. None of these seemed to work, however. Another solution would be re-installing the docker runner, which also did not work in my case.

The problem seems to be with using a dind version of 19.03 or up, however apart from reconfiguring the runner I could not find a solution.

Debugging this issue further proves very difficult, and I was unable to do it effectively to get the error I was having.

The result of this is that I am not able to implement Docker into my Gitlab CI/CD pipeline.