Patrik Torn, 267185

COURSE PROJECT

01.12.2020
COMP.SE.140

1. CI/CD PIPELINE

A CI / CD pipeline has been created for the project using Gitlab's CI / CD service. Pipeline facilitates the software developer's daily development work with its automated functions. Pipeline has two significant roles. The first task is to perform unit and integration tests of the application whenever changes occur in version control, ie a new commit is introduced to the master branch of the project. In addition to the tests, an important task for the pipeline is to build the project whenever a new version appears in version control. In this case, the release version of the project is constantly updated, and its construction does not require any manual work. If an error occurs in the test times, the build is not imported, but the developer must first correct the errors in the tests.

The course does not offer a paid Gitlab’s CI / CD pipeline, which is why I have taken advantage of the Gitlab in your development environment. Gitlab's local CI / CD tube required installing the Gitlab docker image on your own computer and adjusting the gitlab-runner. Gitlab's runner can therefore also be introduced in the production version, but this feature is subject to a fee. Pipeline functionality is demonstrated in the .gitlab-ci.yml file at the root of the project. The configuration file determines which operations (jobs) the pipeline should perform.

2. INSTRUCTIONS TO TEST THE SYSTEM

2.1 Getting started

There are two ways to use and test the system on your own computer: by utilizing the Docker, or by downloading the required development environment for the system. We recommend using Docker, as the project includes five different subsystems and a lot of dependencies, which can make their interoperability challenging in different environments. I actually developed a system for the Windows operating system, but the system should also act as Linux and Unix operating system as well.

2.1.1 Start the system using Docker

```
1. Install docker to your computer

Run the application with following scripts to see expected results

$ git clone https://github.com/PatrikTorn/rabbitmq.git
$ docker-compose build --no-cache
$ docker-compose up -d
(Wait for at most 30 seconds...)


To close the application, run the following scripts:

$ docker-compose down
```

2.1.2 Start the system without Docker

```

If you want to test the application without Docker, you should in-
stall RabbitMQ server to your computer and start the server on port 5672

Run the following scripts to start each server:
```
cd httpserv && npm install && npm start
cd imed && npm install && npm start
cd orig && npm install && npm start
cd api && cpm install && npm start
```
```

2.2 Set up Gitlab locally

You have to set up Gitlab locally to test the CI/CD pipeline of application (if you don't have bought gitlab runners). The instructions to set up Gitlab is found from

https://gitlab.com/gitlab-org/gitlab-foss/-/issues/50851

2.3 Testing the features

Once you have successfully started the application, you can test its features using either the user interface (Postman, browser) or the command line. Personally, I prefer Postman because it makes it easy to make requests.

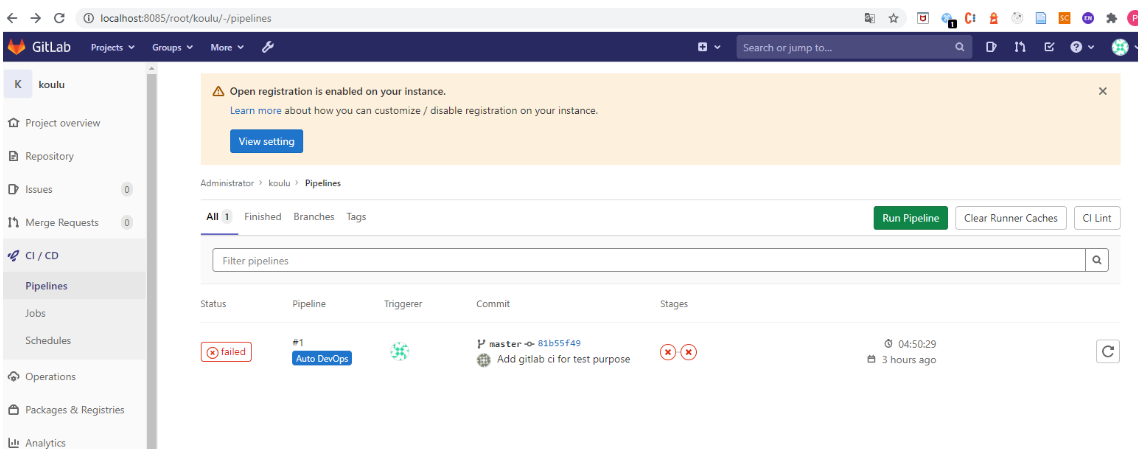
GET localhost:8080/	Gets the newest message from queue
GET localhost:8081/messages	Gets all messages since the startup
GET localhost:8081/state	Gets the state of application
PUT localhost:8081/state (body {payload: RUNNING|PAUSED|INIT|SHUTDOWN})	Sets the state of application
GET localhost:8081/run-log	Gets the logs of states set in interface above

GET commands can be executed directly in the browser to that address, but the PUT command fails in the browser, and the CURL function in Postman or the command line executes the correct query. You can create a PUT command with CURL using the command:

```
curl --location --request PUT 'http://localhost:8081/state' \
--header 'Content-Type: application/json' \
--data-raw '{
    "payload": "RUNNING"
}'
```

3. EXAMPLE RUNS

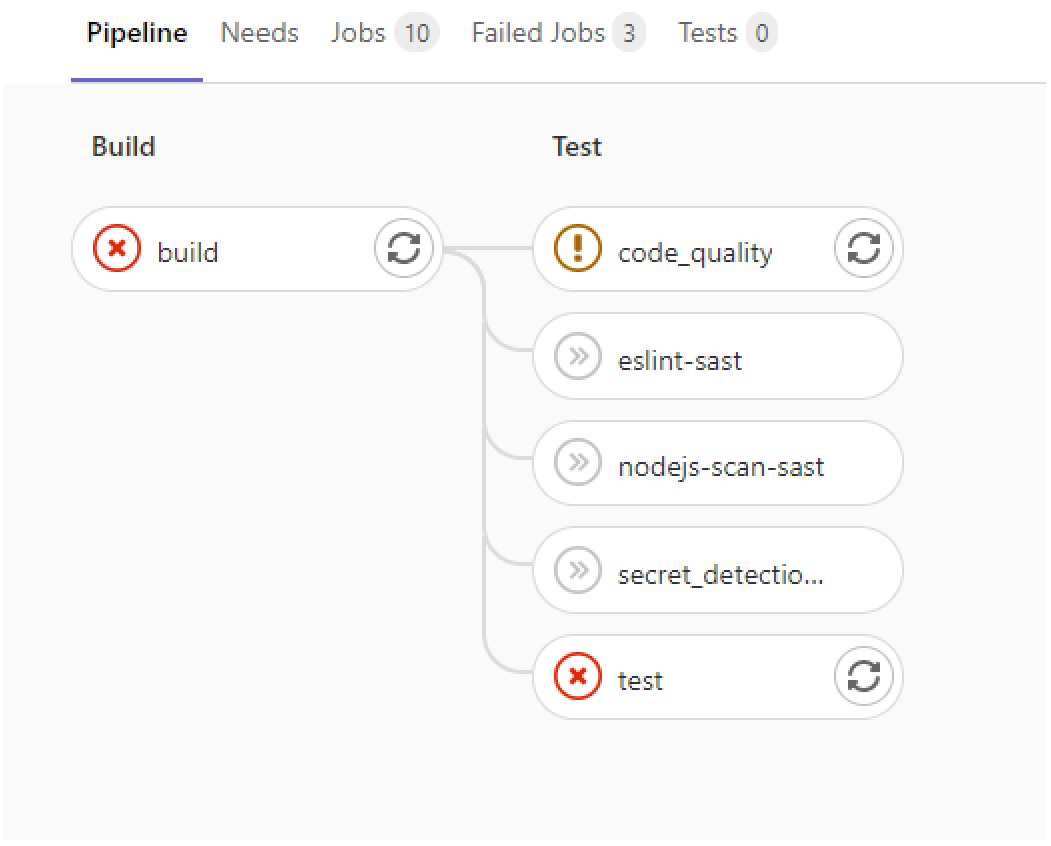
The Gitlab CI / CD tube provides logs of tests and builds performed. Since my working environment was Windows, I didn't manage to get Gitlab runner work properly, so I have only failed case of runs in Gitlab. I spent tens of hours thinking what might be problem, but I didn't find the solution, why runner didn't work.

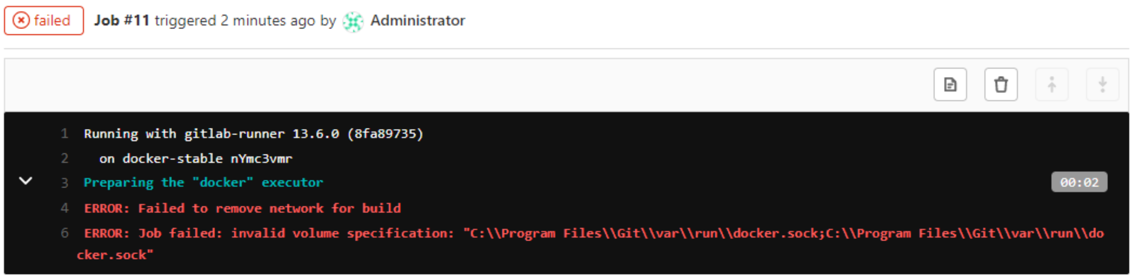

localhost:8085/root/koulu/-/pipelines

Open registration is enabled on your instance.
Learn more about how you can customize / disable registration on your instance.

Administrator > koulu > Pipelines

Status	Pipeline	Triggerer	Commit	Stages		
failed	#1 Auto DevOps		master 81b55f49 Add gitlab ci for test purpose		04:50:29 3 hours ago	





failed Job #11 triggered 2 minutes ago by Administrator

```
Running with gitlab-runner 13.6.0 (8fa89735)
  on docker-stable nYmc3vmr
Preparing the "docker" executor                                                    00:02
ERROR: Failed to remove network for build
ERROR: Job failed: invalid volume specification: "C:\\Program Files\\Git\\var\\run\\docker.sock;C:\\Program Files\\Git\\var\\run\\do
cker.sock"
```

4. MAIN LEARNINGS

I have worked in software development positions for more than half a decade, and I have not yet set up so far, a single pipelineä. This course assignment was very rewarding for me, even though the work was complex. The level of difficulty of the task in the amount of code was not so much a problem, but mainly learning something new and understanding and applying it. I learned the new **AMQP protocol** at work because I used the RabbitMQ message breaker to communicate between different servers. In addition, I learned how to use the **Docker** and **Docker-compose** tools. I see that in many cases a project is good to implement with a virtual machine if its complexity and package dependencies start to escape. I learned to use **Test-driven development** for the first time, where tests were performed before the code itself. I did not see this approach very useful for myself, because I'm used to carry out the tests always in arrears. In addition, perhaps most importantly, I learned **CI / CD pipeline** construction for a version control service. This is definitely the most important contribution of this exercise work, which I also spent considerably the most time on. I believe that these lessons brought by the course will give me many more skills in my future working life as well.

I have listed below the number of hours the project different stages:

Phase 1 coding	15h
Tests	6h
Phase 2 coding	12h
Gitlab CI/CD	15h
Total	**48h**