Pipeline Documentation

Group 3
lm222qm – dr222ew – ro222fd – aw222zr

Table of Contents

1 – Introduction

This documentation concerns the Gitlab pipeline available at:
https://gitlab.lnu.se/ro222fd/2dv611-grp3

In order to provide new developers with an overview regarding how this pipeline functions and how it is set up, we will in this document analyze each step which might be considered relevant for this understanding to occur. This includes the: foundational network and server structure; app and related aspects; basic pipeline functionality; and finally the extended pipeline functionality.

2 – Server and Network Infrastructure

Server Setup

We have two different environments setups in CS Cloud, one for the development environment and one for the production environment. The two environments are created and structured exactly the same, the main difference is different IP addresses. Each CS Cloud environment consists of one master server and two node servers. The master server on each environment has a floating IP associated with it for us to be able to connect to the master through SSH from our local computers. The master is used not only as the kubernetes master node, but also as a gateway server and through it we can connect to the other servers in the CS Cloud environment.

Load Balancer Setup

Each environment, both the development and the production setups in CS Cloud also have its own load balancer with an associated floating IP for us to be able to access the application through HTTP. The CS Cloud load balancers are configured for using port 80 (HTTP) and are listening on our two node servers and the different 31XXX ports that are used for the services inside the kubernetes clusters. The load balancer algorithm we decided to use is Round Robin since each deployment consists of two replicas that are configured exactly the same.

Network Setup

The network setup in CS Cloud consists of a Router that is connecting the public network and our LAN network to each other. Our LAN network is associated with a sub-network. We are using the 172-series as our network address since kubernetes will be using the 192-series. The network address we are using in both our CS Cloud environments is 172.16.0.0/24 and 172.16.0.2-172.16.0.32 for the IP allocation pools.

3 – Kubernetes

Kubernetes Network Setup

We decided to have two different kubernetes clusters since we are using two different CS Cloud environments for development and production. Both of the clusters are connected to GitLab using GitLab's own built-in functionality for kubernetes. For separating the two clusters, each has a different environment scope, development has "*" (all) and production has only "production" as the environment scope. Each cluster consists of three nodes, one master and two nodes. When initiating the Kubernetes cluster we are using the IP 192.168.0.0/16 and the pod network is installed using "Calico".

Docker Setup

For pushing and pulling are own images in our pipeline file and in the kubernetes manifest files, we had to set up our own private docker registry. We used GitLab's built-in functionality for container registries. We are using two images in our image repository; one client image and one api image. Each image is frequently being pushed and pulled with new changes every time our pipeline is running. Both the client and api image have two tags; stable and latest. The latest tag is used for the pipeline running in development and stable tag is used for the pipeline running in production.

Client Service Setup

When setting up our client service we have two different kubernetes manifest files; dev_client.yml for the development environment and deploy_client.yml. Both of the manifest files are structured the same way the main differences are the tags of docker images that are being pulled from our private docker registry on GitLab. In the kubernetes manifest files, we are first creating a deployment for client-webapp pulling the docker image and building two replicas, meaning it will be two identical pods running the client side of our application. Then, we create a service for the client and open NodePort 31234 instead of just using a Cluster IP.

Api Service Setup

When setting up our api service we are doing the same as for the client service. We have two different kubernetes manifest files; dev_api.yml for the development environment and deploy_api.yml for the production environment. Like the client setup, the files are almost identical except the tags of the docker images that are being pulled from the GitLab container registry. First a deployment is created using the docker image for the api and building two replicas. For the api service, we have to open NodePort 31400 since the client service needs to be able to send requests to the api side when the application is publically running. Using just a local kubernetes address is not possible since the requests are done in the browser and not inside a pod.

MySQL

We deployed the Services in three tasks that are in the *dev_db.yml* and *deploy_db.yml* files. The deploy_db.yml file is the production deployment file for the database. One of the tasks in these files is a ConfigMap that holds a sql file. This file is meant to be used when you deploy MySQL for the first time. Then we have the Deployment and Service task. The only interesting part of the Deployment task is the "sql-init-vol". It is a volume that contains the file of the config-map, init.sql. We must mount it on the directory */dockerentrypoint-initdb.d/*. This is because all files in that folder will always be executed at start.

4 – Application

We chose the open-source application webapp-nodejs developed by Kazumasa Kohtaka (https://github.com/kkohtaka/webapp-nodejs). It is a simple web application, "hello world style".

The origin app is a very simple "hello world"-application. It has a RESTful-API built with Express, a Node.js web application framework. In front of this Jade, a templating engine is used. Jade is primarily used for server-side templating in NodeJS. We think it's more effective and easy to work with a standalone frontend so therefore we decided to rebuild the origin app. Jade was removed and we created a React app. React creates a frontend build pipeline which is perfect for us since we have a RESTapi-solution as backend. The frontend is very simple but that also means that it's very simple to "getting started" as a new developer on the frontend on this application.

The backend in the origin app only had one simple route which only returned a hardcoded string. We have implemented 2 routers and 2 controllers. One controller that keeps track of which environment the application is running in and one controller that establishes a connection to the database and works as a data access layer (DAL). The endpoints in the DAL request information from the database and processes this before returning it to the frontend (the one who sends the request).

Our application does need a database. For this we use mariadb which is a relational database.

The origin app had mocha installed. It's a test framework for Node.js. There was one unit-test implemented but that did not test anything in the application so we have implemented our own test and will add more tests in the future as the functionality and complexity of the application increases.

We wanted to have Integration tests. We use Cypress for this. Cypress is an end-to-end testing framework that is easy to set up and learn how to write tests with.

Every part of the application is "Dockerized". If you have docker installed on your computer you only need one command (docker-compose up --build) to set up the whole environment and start developing.

5 – Basic Pipeline

The basic pipeline consists of building the application and deploying it to the Kubernetes environment, via Gitlab's CI-server. This is done via the general pipeline stages:
# Unit tests are run against the JS application, and if these fail the whole pipeline is considered to have failed and no building of images occurs.
# Images are built and stored in the image repository. (Currently available images can be viewed in "Packages & Registries → Container Registry").
# The images are pulled from the image repo, and the final product artifact is deployed and consequently integration testing occurs. (Pipeline process and result can be viewed in "CI/CD → Pipelines". More detailed info about the current stages is found in "CI/CD → Pipelines → Jobs").

In this basic pipeline, the runner which is used is a choice between the below two:
# Gitlab's shared runners – these however entails a cost for running, after the free 200 minutes per month have been used.
# The runner installed via Gitlab's API for Kubernetes. This runner can however not be customized as easily or to the same degree as a runner which is customized and installed from scratch.
In order to set the desired runner, go to "Settings → CI/CD → Runners". For the API installed runner, click "pause" or "resume". For the shared runner, click "enable" or "disable" shared runners. This manages the type of runner which will be used for handling the pipeline process.

The basic pipeline however depends on a quite limited version of the .gitlab-ci.yml, and is therefore no longer in use. The reason for mentioning the basic pipeline here, is to provide the simplest overview possible over how the pipeline is intended to function. In reality we have extended the pipeline with some additional functionality. This additional functionality will be presented in the following section. A final aspect of the basic pipeline will however be noted here, concerning the running of the pipeline process.

In order to run the pipeline, you can take one of several actions:
# Incur a change to the code-base via a local repository, and push it up to the central repository. This will trigger the pipeline process.
# Incur a change to the .gitlab-ci.yml file by editing it directly via the central repo's online functionality(Project overview →.gitlab-ci.yml → edit → Commit Changes).

# If no change is supposed to be made and one wishes to run the pipeline simply in order to test its functionality, then the most relevant way in which to do so is “CI/CD → Pipelines → Run Pipeline” and then set the configurations for how it should be run.

6 – Extended Pipeline

For the extended pipeline functionality, we have implemented the following:

# The possibility to choose between three different types of runners(custom runner installed via script directly on the Kubernetes cluster(has its own node), Kubernetes API installed runner, Gitlab shared runner).
In order to use the custom runner, remove(in child-pipelines/gitlab-ci-dev.yml) in both locations:
DOCKER_HOST: tcp://docker:2375
To use the Gitlab installed runner again, readd this variable in both locations.
Runners are controlled via “Settings -> CI/CD -> Runners”.

# Splitting the build process – using variables and stages – so that one kind of environment is built for production, and one kind of environment for development.

# Splitting the pipeline into one main pipeline, and two child pipelines(one for production, and one for development). These two child pipelines then runs the integrations test in a final child pipeline.

# Used the “refs::only::changes” so that images will only build if there are changes to the respective codebase. Api will only build if changes have been made to webapp/api/* and kubernetes-manifest/. This is still in an experimental stage, so it might be necessary to add additional folders and/or files to those triggering a rebuild of images.
Production always runs the building of the images, rather than relying on them existing since previous builds. But in development they are only rebuilt in response to changes.
(In the event that one would like to run the whole development pipeline without this limitation, use: “CI/CD -> Pipelines -> Run Pipeline”).

7 – Final Remarks

The pipeline is quite basic. There is not that much advanced functionality. It gets the basic job done, but beyond that only a little bit more. We have tried to document it to the best of our ability, and we have spent time in order to optimize new developers’ understanding and overview of it.

You as a new developer should not have a hard time with getting it up and running, since only a very basic understanding of, and experience with, pipelines would be necessary in order to do so. If you have any questions in regards to a certain aspect of the pipeline - or the pipeline in general - try contacting one of the below:

Anton Wiklund:
aw222zr@student.lnu.se
Linus Mörtzell:
lm222qm@student.lnu.se
Renato Opazo Salgado:
ro222fd@student.lnu.se
Daniela Rondahl:
dr222ew@student.lnu.se