Working with CI and K8s

Contents

1 - Working with Github CI features 1

1.1 - Check the project settings and clone it 1

1.2 - Run the project 2

1.3 - Modify a workflow 2

1.3.1 - Trying a few commands in our own WSL terminal 2

1.3.2 - Adding steps to the existing workflow 2

2 - Run a small K8s Minikube example 2

3 - Deploy the devops-ci-demo example on Minikube 2

3.1 – Start Minikube again 2

3.2 – Deploy devops-ci-demo 3

3.3 – Setup a service to provide access to our application 3

3.4 – See the devops-ci-demo running in Minikube 3

3.5 – Use kubectl to dump current setup as yaml-files 3

3.6 – Use Kompose to generate k8s config files 4

1 - Working with Github CI features

Fork (don’t simply clone) the project https://github.com/DaFessor/devops-ci-demo.

Question(s):

- Look at the Dockerfile – what does it do?
- Do you have to build the project to create a runnable jar?
- If you are in doubt, take a look at https://docs.docker.com/build/building/multi-stage/

1.1 - Check the project settings and clone it

Make sure that your forked project has the same settings for branch protection, actions etc. as the original. Change settings as needed and create a local copy by cloning your forked project to your own PC.

1.2 - Run the project

There's a compose.yaml file in the project, so you should be able to start the project simply by typing the following in a terminal:

```
> docker compose up
```

Check that things are actually working by pointing your browser at http://localhost:8080/

1.3 - Modify a workflow

Look at the file `push_to_main.yaml`.

Question(s):

- Go through the file step by step, what do they do? – try to come up with your best qualified guess at what each step does (otherwise Google and the Github documentation is your friend).

1.3.1 - Trying a few commands in our own WSL terminal

A step in a Github action can do anything, including running an arbitrary command. The jobs themselves (and hence the steps) are executed inside a container running an Ubuntu image. Try these commands in a SWL Terminal:

```
> date                          // 1) Prints current date/time
> echo "Hi Mom!"                // 2) Simply prints a string
> pwd                           // 3) Shows current working directory
```

1.3.2 - Adding steps to the existing workflow

2 - Run a small K8s Minikube example

Click in to https://kubernetes.io/docs/tutorials/hello-minikube/ and follow the instructions to run a small application on your Minikube installation.

Follow all the steps to the end so you can see how to remove and clean up stuff that runs in your (very small) cluster.

3 - Deploy the devops-ci-demo example on Minikube

We'll basically do the same thing as we did in the hello-world example in exercise 2, only this time we will use our own image from (your forked copy of) the devops-ci-demo project on Github.

That image should be fetched like this: `ghcr.io/<your_github_username>/devops-ci-demo/devops-ci-demo-img:latest`
(Note: if have problems getting your forked copy up and running so that you can generate your own images, just use this one from the original project: `ghcr.io/dafessor/devops-ci-demo/devops-ci-demo-img:latest`)

3.1 – Start Minikube again

Start Minikube (if needed) and open the dashboard

```
> minikube start
> minikube dashboard
```

Just let that command run, and start a new Terminal-tab and use that for the remainder of the commands below.

3.2 – Deploy devops-ci-demo

Deploy the your image (use the image from your own forked copy if it works). This is a single long line:

```
> kubectl create deployment devops-ci-demo
  --image=ghcr.io/dafessor/devops-ci-demo/devops-ci-demo-img:latest
```

Look in the dashboard, you should be able to see the deployment.

3.3 – Setup a service to provide access to our application

Create a service to provide network access to the deployment:

```
➢ kubectl expose deployment devops-ci-demo --type=LoadBalancer
  --port=8088
```

Look in the dashboard, you should be able to see the new service.

3.4 – See the devops-ci-demo running in Minikube

To see the application run, we can type:

```
> minikube service devops-ci-demo  // <--Type this command
|-----------|-----------------|-------------|----------------------------|
| NAMESPACE |      NAME       | TARGET PORT |            URL             |
|-----------|-----------------|-------------|----------------------------|
| default   | devops-ci-demo  |        8088 | http://192.168.49.2:31302 |
|-----------|-----------------|-------------|----------------------------|
🏃  Starting tunnel for service devops-ci-demo.
|-----------|-----------------|-------------|------------------------|
| NAMESPACE |      NAME       | TARGET PORT |          URL           |
|-----------|-----------------|-------------|------------------------|
| default   | devops-ci-demo  |             | http://127.0.0.1:37103 |
|-----------|-----------------|-------------|------------------------|
🎉  Opening service default/devops-ci-demo in default browser...
👉  http://127.0.0.1:37103
❗  Because you are using a Docker driver on linux, the terminal needs to
be open to run it.
```

By clicking on the link (shown in blue) we can now see the application running happily inside Minikube.

3.5 – Use kubectl to dump current setup as yaml-files

Once we have a working setup, we can use kubectl to ask the cluster to dump the current configuration as yaml-files:

```
> kubectl get deployment devops-ci-demo -o yaml // Get deployment config
> kubectl get service devops-ci-demo -o yaml    // Get service config
```

If we want to store the config we can copy-paste them from the screen, or more elegantly save the output from the commands into a file:

```
> kubectl get deployment devops-ci-demo -o yaml > deployment.yaml
> kubectl get service devops-ci-demo -o yaml > service.yaml
```

After this we the 2 necessary config files to recreate our setup in the files `deployment.yaml` and `service.yaml`.

3.6 – Use Kompose to generate k8s config files

If we have a docker compose file, we can also use kompose to directly generate the deployment and service files:

```
> kompose convert
INFO Kubernetes file "backend-service.yaml" created
INFO Kubernetes file "backend-deployment.yaml" created
```

Try it out and take a look at the generated files.