The Example in Detail

Let's look at an example of how Kubernetes can be used.

We'll cover the following

Setup

The example is accessible at https://github.com/ewolff/microservice-kubernetes (https://github.com/ewolff/microservice-kubernetes). https://github.com/ewolff/microservice-kubernetes/blob/master/HOW-TO-RUN.md (https://github.com/ewolff/microservice-kubernetes/blob/master/HOW-TO-RUN.md) explains in detail how to install the required software for running the example.

The following steps are necessary for running the example:

- Minikube (https://github.com/kubernetes/minikube) as minimal Kubernetes installation has to be installed. Instructions for this can be found at https://github.com/kubernetes/minikube#installation (https://github.com/kubernetes/minikube#installation).

- kubectl (https://kubernetes.io/docs/user-guide/kubectl-overview/) is a command line tool for handling Kubernetes and also has to be installed. Its installation is described at https://kubernetes.io/docs/tasks/tools/install-kubectl/

(https://kubernetes.io/docs/tasks/tools/install-kubectl/).

- The script `docker-build.sh` generates the Docker images for the microservices and uploads them into the public Docker hub. This step is optional since the images are already available on the Docker hub. It only has to be performed when changes were introduced to the code or to the configuration of the microservices. Before starting the script, the Java code has to be compiled with `./mvnw clean package` (macOS, Linux) or `mvnw.cmd clean package` (Windows) in directory `microservice-kubernetes-demo`. Then the script `docker-build.sh` creates the images with `docker build` and with `docker tag` they receive a globally unique name and `docker push` uploads them into the Docker hub. Using the public Docker hubs spares the installation of a Docker repository and thereby facilitates the handling of the example.

- The script `kubernets-deploy.sh` deploys the images from the public Docker hub in the Kubernetes cluster and thereby generates the pods, the deployments, the replica sets, and the services. For this, the script uses the tool `kubectl`. `kubectl run` serves to start the image which is downloaded at the indicated URL in the Docker hub. In addition, it is defined which port the Docker container should provide. So, `kubectl run` generates the deployment, which creates the replica set and thereby the pods. `kubectl expose` generates the service which accesses the replica set and thus creates the IP address, node port resp. load balancer and DNS entry.

This excerpt from `kubernetes-deploy.sh` shows the use of the tools using the catalog microservice as an example.

```
#!/bin/sh
if [ -z "$DOCKER_ACCOUNT" ]; then
  DOCKER_ACCOUNT=ewolff
fi;
...
kubectl run catalog \\
 --image=docker.io/$DOCKER_ACCOUNT/microservice-kubernetes-demo-catalog:latest
  \\
 --port=80
kubectl expose deployment/catalog --type="LoadBalancer" --port 80
...
```

Kubernetes YAML

An alternative is to use Kubernetes YAML files. They describe the desired state of deployments and services. For example, here is the part of `microservices.yaml`, for the catalog microservices.

```
...

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: catalog
  name: catalog
spec:
  replicas: 1
  selector:
    matchLabels:
      run: catalog
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: catalog
    spec:
      containers:
      - image: docker.io/ewolff/microservice-kubernetes-demo-cat
alog:latest
        name: catalog
        ports:
        - containerPort: 8080
        resources: {}
status: {}

---

...

apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: catalog
```

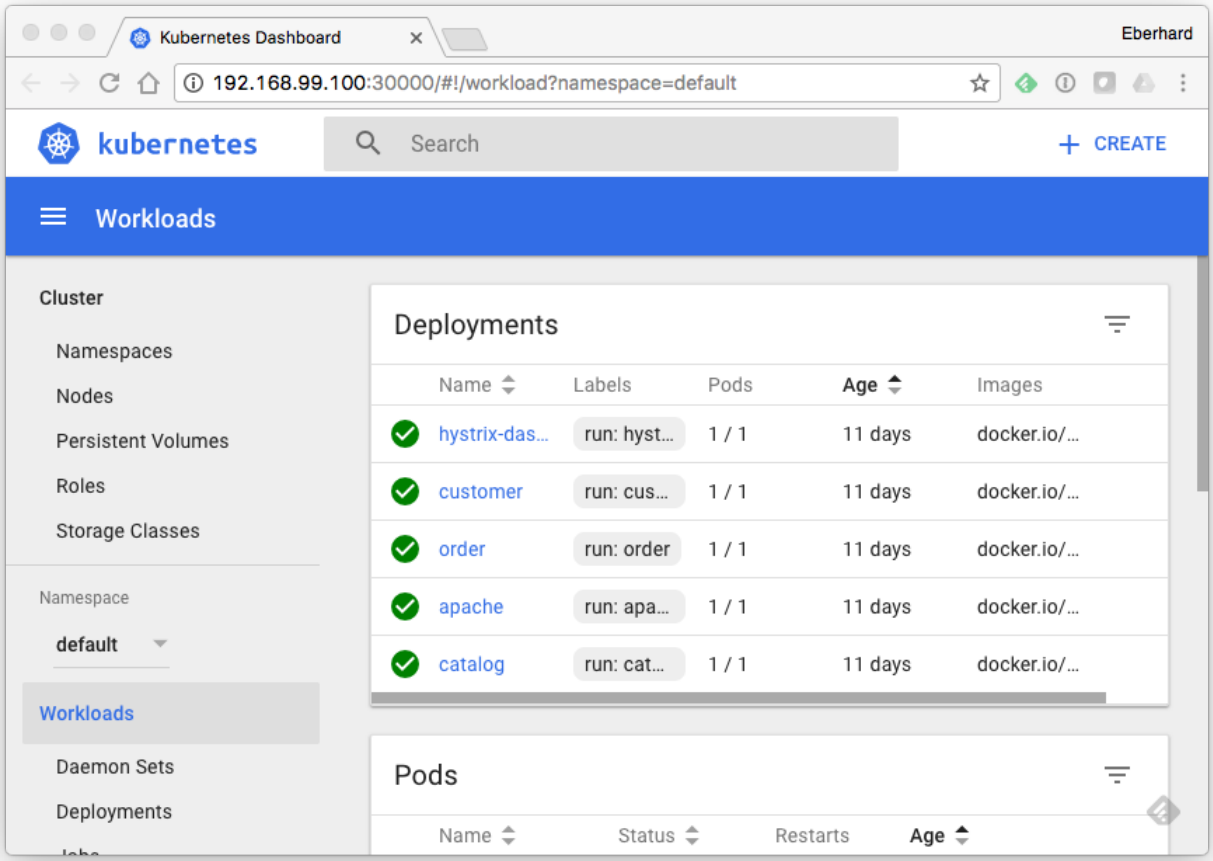
```
  name: catalog
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: catalog
  type: LoadBalancer
status:
  loadBalancer: {}

...
```

The information in the YAML file is very similar to the parameters of the commands above. Using `kubectl apply -f microservices.yaml` all the services and deployment would be created in the Kubernetes cluster. The same command would be used to update the services and deployments after any changes.

Some Minikube commands

`minikube dashboard` displays the dashboard in the web browser, which displays the deployments and additional elements of Kubernetes. This makes it easy to understand the state of the services and deployments. See the screenshot below.



Kubernetes Dashboard

`minikube service apache` opens the Apache service in the web browser and thereby offers access to the microservices in the Kubernetes environment.

The script `kubernetes-remove.sh` can be used to delete the example. It uses `kubectl delete service` for deleting the services, and `kubectl delete deployments` for deleting the deployments.

In the next lesson, we'll look at some additional Kubernetes features.

← Back

Next →

The Example with Kubernetes

Additional Kubernetes Features

Mark as Completed

Report an Issue