Kubernetes lab

TLC/ST course

The purpose of this lab is to discover Kubernetes and to try a number of interesting features. For simplicity, we will run Kubernetes on a dummy cluster via minikube.

---

1 Setup a dummy cluster

To install minikube, go to `https://kubernetes.io/docs/tasks/tools/install-minikube/`. You will find basic information on minikube at `https://kubernetes.io/docs/setup/minikube/`. *You might need to run some commands as root/administrator.*

The following commands will help you to easily manage your dummy cluster:

- Start your cluster with `minikube start`.
- Access Kubernetes Dashboard with `minikube dashboard`.
- Access a service with `minikube service <service-name>`.
- You can destroy your cluster with `minikube delete`.

For all other operations, you will need the kubernetes client. Install instructions are provided here: `https://kubernetes.io/docs/tasks/tools/install-kubectl/`

Everything should work if you are able to run `kubectl cluster-info dump` without error (it should output lot of information about your cluster).

2 Discovery of Kubernetes

Try the examples shown in class (files are given in appendix A):

- Create a pod containing two containers and display its configuration. Can you access it from your local browser? Delete your pod when you are done with this question.
- Create a replication controller, display its configuration, and rescale it. Leave it running as you will need it for the next step.
- Create a service acting as a load balancer between the pods managed by your replication controller. Display its port number, and try accessing it from your browser.
- Delete all your remaining pods, replication controllers and services.

3 Deploying WordPress

WordPress[1] is the most famous content management system nowadays. As of October 2016, about 27% of Web servers worldwide were powered by WordPress[2]. The second most famous CMS is Joomla, which powers "only" 3.2% of Web servers worldwide.

Being one of the most famous pieces of software on the Internet, we can expect that somebody has already taken care of developing the required specification files to deploy WordPress on Kubernetes. After a few minutes of search on the Internet, we indeed find a nice implementation of WordPress for Kubernetes[3].

1. Create a "deployment" running wordpress:
   `kubectl run my-wordpress --image=tutum/wordpress --port=80`
   (do not forget to replace "gp" with your own initials)

2. Expose your deployment to the outside world:
   `kubectl expose deployment my-wordpress --type=NodePort`

3. Visit your new site by running `minikube service my-wordpress`. You should now be able to give a name to your site, browse it, edit the pages etc.

4. We are curious to see what happens when scaling the system to two replicas. Let's request two replicas of the same deployment:
   `kubectl scale deployment my-wordpress --replicas=2`
   Verify that a second pod has been created and that both pods are now referenced in the service descrption.

5. Keep on browsing on your web site: edit a couple of pages, view them, add comments, etc. At some point you should see some "strange" behavior. This is the time to stop typing and to start thinking and answering a few questions.

Question 1: can you figure out what is going on? You will need to do a little bit of guesswork, but the following questions should put you on track: how is WordPress organized in terms of web servers, database servers, etc.? How many containers are included in the "wordpress" pod? How many web/database servers? What happens to the site's *data* when I create a second pod?

Question 2: propose a better organization for deploying WordPress in Kubernetes such that we can easily scale it with no difficulty. It is *not* requested to implement your solution (you will not have enough time for that), but please explain in details on paper the way it would work.

Please write your answers to questions 1 and 2 in a small report, and send it back to your instructor within a week after the lab. Do not forget to write your names on the report!

A Kubernetes files

A.1 pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
```

[1] http://www.wordpress.org
[2] https://w3techs.com/
[3] https://cloud.google.com/container-engine/docs/tutorials/hello-wordpress

```
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
```

A.2 rc.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-rc
spec:
  replicas: 4
  selector:
    app: mywebapp
  template:
    metadata:
      name: mywebapp-pod
      labels:
        app: mywebapp
    spec:
      containers:
        - name: frontend
          image: nginx
          ports:
            - containerPort: 80
        - name: keyvaluestore
          image: redis
          ports:
            - containerPort: 6379
```

A.3 service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: mywebapp
  ports:
    - port: 80
      targetPort: 80
  type: NodePort
```

— the end —