Lab 9.2: Kubernetes Deployments and Services
ID720 Virtualisation

Introduction

In the previous lab we saw how we could deploy pods to a Kubernetes cluster that runs containers and creates volumes on the cluster's nodes. But until we do something to set it up, those containers are not networked. Two containers on the cluster cannot communicate with each other and it is not possible to connect to a container over the network from outside the cluster. To do either of these, we need to create and configure Kubernetes *service objects*. In this lab we will see how to do this.

Also, last time we launched our containers using Pods, which are the basic building blocks of many Kubernetes resources. Usually however, we prefer to work with higher-level constructs, like the *deployments* we will create today

1 Namespaces

We are all working on one shared instance of Kubernetes, so there will be problems when we want to launch resources identified by a particular name. Those names need to be unique across the Kubernetes instance. We get around this by using *namespaces*. This way, you can create a deployment named `redis`, for example, without it conflicting with another redis deployment in a different workspace.

Create your workspace with the command

```
microk8s.kubectl create namespace <namespace-name>
```

Use your username for your namespace name.

2 Deploy redis

We will deploy two containers to our cluster. One will use a standard `redis` image and the other will use the same `flaskapp` image we have used previously. Recall that the flask application needs to connect to redis over the network in order to function, and the flask application itself is meant to be accessed over the network.

First, launch a redis container with the following command

```
microk8s.kubectl -n <namespace-name> create deployment redis --image=redis:alpine
```

This creates a *deployment*, which is a higher level construct than a simple pod.

Redis listens for requests on port 6379, but our container hasn't been created with that port open. It's easy to modify its configuration with the command

```
microk8s.kubectl  -n <namespace-name> edit deployment redis
```

This will open our deployment's manifest in a text editor. We can make changes which will be applied once we save and close the editor. Look for the section of the manifest where the container properties are defined

and add the following, right after the `name` property. (That property is included below to illustrate the correct indenting level. You don't need to add it again.)

```
name: redis
ports:
  - containerPort: 6379
    name: redis
    protocol: TCP
```

Even though we've exposed the container port, it isn't accessible to other containers in the cluster yet. We need to create the service object with the command

```
microk8s.kubectl  -n <namespace-name> expose deployment redis
```

and then inspect its properties with the command

```
microk8s.kubectl  -n <namespace-name> describe service redis
```

We see from the output that this service provides a `ClusterIP` to which we can connect on port `6379` (because that is the port exposed by the container). Other containers in the cluster that connect to the ClusterIP will have their requests routed to one of the `Endpoints`, of which there is only one in our case. It's possible to connect directly to the endpoint, but Kubernetes will keep the `ClusterIP` stable. The endpoint addresses, on the other hand, may change. Also, since we have enabled the Kubernetes DNS service, the `ClusterIP` is reachable using the service's name, `redis`.

Services exposed through a `ClusterIP` are not accessible outside the cluster.

3 Deploy the flask application

We're going to deploy a Flask application in a different way, using a manifest file. Start by copying the manifest file `/home/ubuntu/flask-depolyment.yml` to your home directory. Edit the file to change the `namespace` property to match your own namespace.

Create the flaskapp deployment with the command:

```
microk8s.kubectl apply -f flask-deployment.yml
```

The flask application container needs to be able to connect to the redis container using the hostname `redis`, and it can. We configured that above. We cannot connect to the flask application, however. We expose it using a *service*.

Copy the manifest file `/home/ubuntu/flask-svc.yml` to your home directory and again edit it to set the correct namespace. Then create your service with the command

```
microk8s.kubectl apply -f flask-svc.yml
```

and see what external port has been assigned by entering the command

```
microk8s.kubectl -n <namespace-name> describe service flask-svc
```

and looking for the `NodePort` value. It should be a number over 30000.

Now, using a browser or curl, you should be able to access the flask application at the address `http://k8s.foo.org.nz:<NodePort>`.

4 Examine your deployed items

You can see the objects you have placed in Kubernetes with commands like

```
microk8s.kubectl -n <namespace-name> get pods
```

and see more detail with commands like

```
microk8s.kubectl -n <namespace-name> describe pod <podname>
```

You can use similar commands to inspect your deployments and services.

When you're done, you can delete your resources with commands like

```
microk8s.kubectl -n <namespace-name> delete deployment redis
microk8s.kubectl -n <namespace-name> delete service redis
microk8s.kubectl delete -f flask-deployment.yml
microk8s.kubectl delete -f flask-svc.yml
```