Lab 13.2: Kubernetes Services
IN720 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 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.

N.B.: Begin this lab by enabling the Kubernetes DNS feature with the command `microk8s.enable dns`.

1 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 create deployment redis --image=redis:alpine
```

This creates a *deployment*, which is a higher level construct than a simple pod. In this case it's simply a matter of convenience. `kubectl create` can't be used to create a 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 edit deployment redis
```

This will open our deployemnt'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 expose deployment redis
```

and then inspect its properties with the command

```
microk8s.kubectl 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.

2 Deploy the flask application

The first steps in this process are basically the same as the ones we carried our for the redis container.

Create the flaskapp deployment:

```
microk8s.kubectl create deployment flaskapp --image=tclark/flaskapp
```

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.

Edit the deployment's properties, this time to expose port 5000 on the container:

```
microk8s.kubectl edit deployment flaskapp
```

Edit the manifest as we did above, just with the different port.

Now we will use an `expose` command to create the service object, but we do not want a `ClusterIP` in this case since it's not accessible outside the cluster. Instead, create a service with a `NodePort` using the command

```
microk8s.kubectl expose deployment flaskapp --type=NodePort
```

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

```
microk8s.kubectl describe service flaskapp
```

and looking for the `NodePort` value

Now, using a browser or curl, you should be able to access the flask application at the address `http://<ip address of server>:<NodePort>`.