Lab 13.1: Deploy a Pod on Kubernetes
IN720 Virtualisation

Introduction

In this lab we will use *microk8s*, a single-host version of Kubernetes to deploy an example pod and see how to work with *kubectl*, the command line interface for Kubernetes. You should do this lab on the single server we used for earlier Docker labs, and **not** on you Docker Swarm machines. Alternatively, you can do this lab on your own computer if you use Ubuntu.

1 Install microk8s

Installing a full Kubernetes cluster is complicated and requires a collection of machines. Luckily, the microk8s version of Kubernetes is designed to be installed on a single machine and gives us all the standard Kubernetes tools. It's primary purpose is to let developers and sysadmins work on container based projects on their workstations, but it's also ideal for learning.

Make no mistake, micro8s is a full version of Kubernetes. It's just a version tailored for special purposes. To avoid clashing with other Kubernetes command line tools, microk8s commands all have to form `microk8s.<subcommand>`. For example, to use `kubectl` we invoke `microk8s.kubectl`. The actual version of kubectl that we are using is a standard version.

We install microk8s on Ubuntu with a snap package rather than a .deb one, so the command to install it is just

```
sudo snap install microk8s --classic
```

Start microk8s and check that it's working properly with the commands

```
sudo microk8s.start
sudo microk8s.status
```

You'll want to be able to work without always needing to invoke sudo, so enable a standard user with the command

```
sudo usermod -a -G microk8s user
```

You'll have to log out and back in for this change to be effective.

(In the case above `user` is just the name of the user we are enabling, so substitute a different user name if necessary.)

2 Prepare a pod manifest

It is possible to deploy a pod to Kubernetes without writing a manifest file, but doing so is somewhat limited and doesn't create a clear record of what our pod is. A better approach is to write a *pod manifest*,

which is a JSON or YAML file declaring the properties of our pods. We'll write a YAML manifest named `hello-pod.yml`. It should contain the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: hello
spec:
  containers:
    - image: tclark/hello
      name: hello-container
```

When we deploy a pod from this manifest, it will unsurprisingly create a pod named `hello` that is comprised of one container and it's associated metadata. Launch this pod with the command

```
microk8s.kubectl apply -f hello-pod.yml
```

Check that the pod is deployed with the command

```
microk8s.kubectl get pods
```

which will return a list of all the pods running on the cluster. If you run it right away, our pod may be shown as `Pending` or `ContainerCreating`, but eventually its status will be `Running`. You can get more details with the command

```
microk8s.kubectl describe pods hello
```

We see that `kubectl` commands generally have the form

```
kubectl <action> <resource-name> [<object-name>]
```

So the commands above apply to the *resource* `pods` and to the *object* `hello`.

We can check the log output from our pod (it's fascinating) with the command

```
microk8s.kubectl logs hello
```

which sort of breaks our pattern, since the resource name, `pods` isn't provided because it's the default resource for this command. We can restrict the output to show the logs for only one container in the pod with

```
microk8s.kubectl logs hello -c hello-container
```

And we can stream the logs live with the command

```
microk8s.kubectl logs hello -f
```

(Hit Ctrl-c to stop this.)

Finally, we can remove our pod from the cluster with the command

```
microk8s.kubectl delete -f hello-pod.yml
```

or

```
microk8s.kubectl delete pods hello
```

3 Next time

One thing about Kubernetes pods is that their containers aren't available on the network without some additional work on our part. This is what we will explore on Friday.