Lab 5.1: Using CoreOS
IN720 Virtualisation

Introduction

If we can run nearly anything inside a Docker container, then maybe all we need from the host system is the facilities needed to run Docker. CoreOS takes this approach, but it also adds machinery so that we can run distributed clusters of hosts than can share information and seamlessly manage containers across the cluster.

1 Log in to your CoreOS cluster machines

CoreOS systems are generally run in clusters. Get the address of three CoreOS machines that have been set up in a cluster for you by the lecturer. You can only ssh into these machines using a an ssh key that was specified in the `cloud-config` at the time the systems were set up. The ssh in using the key, do the following:

1. ssh into 10.25.1.85 using our standard username/password.
2. In this session, log into your first CoreOS Machine with the command
   `ssh -i virt-student.pem core@<ip-address-of-core-system>`

Repeat this process for the other two CoreOS systems so that you have three ssh sessions open.

2 Experimenting with etcd

When deploying containers across a cluster, it is helpful to have a shared information store for configuration data and message passing. CoreOS uses `etcd`, a distributed key-value store, for this. When we set a key-value pair on one machine in our cluster, the others see that pair almost immediiately.

On one machine in your cluster set a key-value pair with `etcdctl`:

```
etcdctl set /test "Test value"
```

Then, retrieve this value on all three machines with the command

```
etcdctl get /test
```

`etcd` also exposes an interface over HTTP:

```
curl -L http://127.0.0.1:2379/v2/keys/test
```

Later we will see that we can use this to set and get values inside containers running on our cluster.

3 Runing containers with Fleet

We use Fleet to set up, start, and stop containers on our cluster. First we can use Fleet to see the machines in our cluster with the command

```
fleetctl list-machines
```

Services that Fleet is prepared to run are called *units*. We can list them with the commands

```
fleetctl list-unit-files
```

to see what units are defined, and

```
fleetctl list-units
```

to see which are running and where. Right now we only see the core units that are used to operate Fleet.

To define our own unit, we need to create a service file. Place a file with the following on any machine in the cluster:

file: hello.service

```
[Unit]
Description=Hello
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill busybox1
ExecStartPre=-/usr/bin/docker rm busybox1
ExecStartPre=/usr/bin/docker pull busybox
ExecStart=/usr/bin/docker run --name busybox1 busybox /bin/sh -c "while true; do echo Hello World; sleep
ExecStop=/usr/bin/docker stop busybox1
```

Then load the new service with the command

```
fleetctl load hello.service
```

Your service should now show up when you run `fleetctl list-unit-files`.

Start the service with the command

```
fleetctl start hello.service
```

You can check on your running service with the commands

```
fleetctl list-units
```

and

```
fleetctl status hello.service
```

Finally, you can stop your service with the command

```
fleetctl destroy hello.service
```

In our next lab, we will create a more interesting service that demonstrates more of the capabilities of etd and Fleet.