Lab 5.2: CoreOS, etcd, and Fleet
IN720 Virtualisation

Introduction

Last time we started using CoreOS. We say that we could use etcd to share information across our cluster. In this lab we will see how we can communicate with etcd inside running containers on CoreOS hosts. We also used Fleet to launch and then stop a container on our cluster. In this lab we will use some additional features of Fleet to exercise more control over our container deployments.

1 Using etcd inside a container

For this exercise, log into two of your CoreOS hosts. On one of them, start a new Docker container as follows:

```
docker run -t -i ubuntu /bin/bash
```

We are not using Fleet to run this container for two reasons. First, we want to run the container directly on this host. Second, we want to attach to it and interact with it via a shell.

Once you are in the container's shell, install `curl` with the command `apt-get install curl`. When this completes, run the command

```
curl -L http://172.17.42.1:4001/v2/keys/foo-service?wait=true&recursive=true
```

After you hit enter, this command will sit and wait. It is waiting for someone using etcd to change a key under the `foo-service` directory in etcd. The IP address `172.17.42.1` is the virtual `docker0` interface on the CoreOS host. This is how the Docker container can reach the host on which it runs.

Now go to the ssh session on your other CoreOS host. On this one enter the command

```
etcdctl set /foo-service/host1 hello
```

Now look and see what has happened in the container on the other host. We see that containers can read information from etcd, and it turns out that they can write in an analagous way. In this way our containers can share information across the cluster.

2 Running containers across the cluster

One reason for running a cluster is to support high availability. But for this to work properly you need to exercise some control over where your containers run. There is no point in running your standby container on the same host as your primary.

Fleet provides options that give you this sort of control. In this example we will specify that our containers run on different hosts.

Create a service file named `apache@.service` on one of your CoreOS machines with the content below.

file: apache@.service

```
[Unit]
Description=My Apache Frontend
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache
ExecStart=/usr/bin/docker run --rm --name apache1 -p 80:80 coreos/apache /usr/sbin/apache2ctl -D FOREGR
ExecStop=/usr/bin/docker stop apache1

[X-Fleet]
Conflicts=apache@*.service
```

Note the `Conflicts` directive. We can run multiple instances of this service, but only one instance per host. Start three instances of the service like this:

```
fleetctl start apache@1
fleetctl start apache@2
fleetctl start apache@3
```

Then use `fleetctl list-units` to verify that each one runs on a different host.

3 Run a sidekick service

Now, create the following service file:

file: apache-discovery@.service

```
[Unit]
Description=Announce Apache
BindsTo=apache@%i.service
After=apache@%i.service

[Service]
ExecStart=/bin/sh -c "while true; do etcdctl set /services/website/apache@%i '{ \"host\": \"%H\", \"port
ExecStop=/usr/bin/etcdctl rm /services/website/apache@%i

[X-Fleet]
MachineOf=apache@%i.service
```

This service doesn't use a container. It just runs a simple shell script. Its job is to notify etcd that an Apache container is running. Start three of them with the following commands:

```
fleetctl start apache-discovery@1
fleetctl start apache-discovery@2
fleetctl start apache-discovery@3
```

Now, query etcd to get information about your Apache container instances:

```
etcdctl ls /services/ --recursive
```

```
etcdctl get /services/website/apache@1
etcdctl get /services/website/apache@2
etcdctl get /services/website/apache@3
```

You could use this information to configure a load balancer or reverse proxy server.