Exercise 14

Create a Kubernetes Cluster in DigitalOcean and Deploy Cassandra

Prior Knowledge
Unix Command Line Shell
YAML
Ex 4b Terraform

Learning Objectives
Terraform to instantiate Kubernetes in DigitalOcean
Deploy Cassandra to Kubernetes
See how Cassandra replicates

Software Requirements
Browser
kubectl
k9s
doctl

Overview

In this exercise we are going to instantiate a Kubernetes cluster in DO (using Terraform), then install a Cassandra ring onto the kubernetes cluster. Finally we will do some load-testing.

We will mainly do this from the CLI (so it can be repeated and automated), especially using declarative approaches.

Steps

1. Follow the separate instructions to sign up for the Github education pack and get free DigitalOcean credit. If you already have a DigitalOcean account, it should cost less than $1 to do this exercise if you don't leave resources running.

2. On the Ubuntu VM install some extra software:

   Remove the old version of kubectl

   ```
   sudo apt uninstall kubectl -y
   ```

   And install a newer version:

```
sudo snap install kubectl --classic
```


3. If you haven't done Exercise 4b on Terraform please do that first! (Or at least install terraform following that exercise)

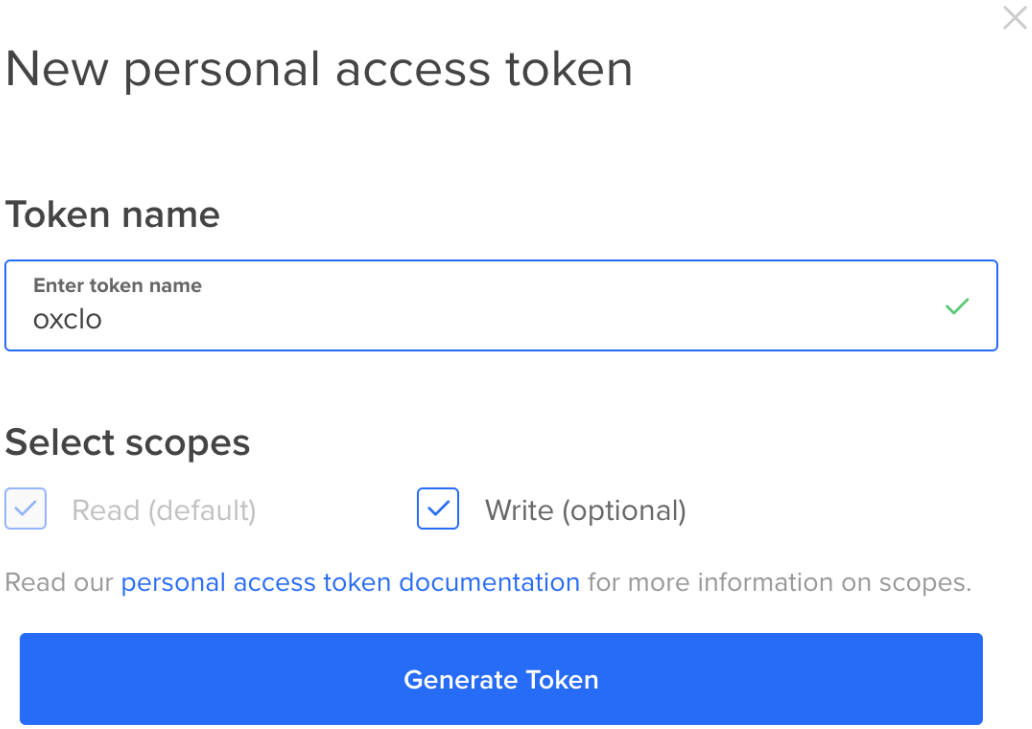
4. Install k9s and doctl

```
wget -O - -q https://freo.me/install-k9s | bash
sudo snap install doctl
sudo snap connect doctl:kube-config
mkdir ~/.kube
```

5. Authenticate to Digital Ocean.

   First browse to https://cloud.digitalocean.com/account/api/tokens

   Click **Generate a new token**



6. Copy the new token into a file so you don't lose it.

7. Now type

```
doctl auth init

Please authenticate doctl for use with your
DigitalOcean account. You can generate a token in
the control panel at
https://cloud.digitalocean.com/account/api/tokens
```

```
Enter your access token:
```

Paste in your token.

```
Validating token... OK
```

8. We also need to use that token in the environment so Terraform can find it:

```
echo “export DIGITALOCEAN_TOKEN=<put your token here>” >> ~/.bashrc

source ~/.bashrc
```

9. Check out my git repository:

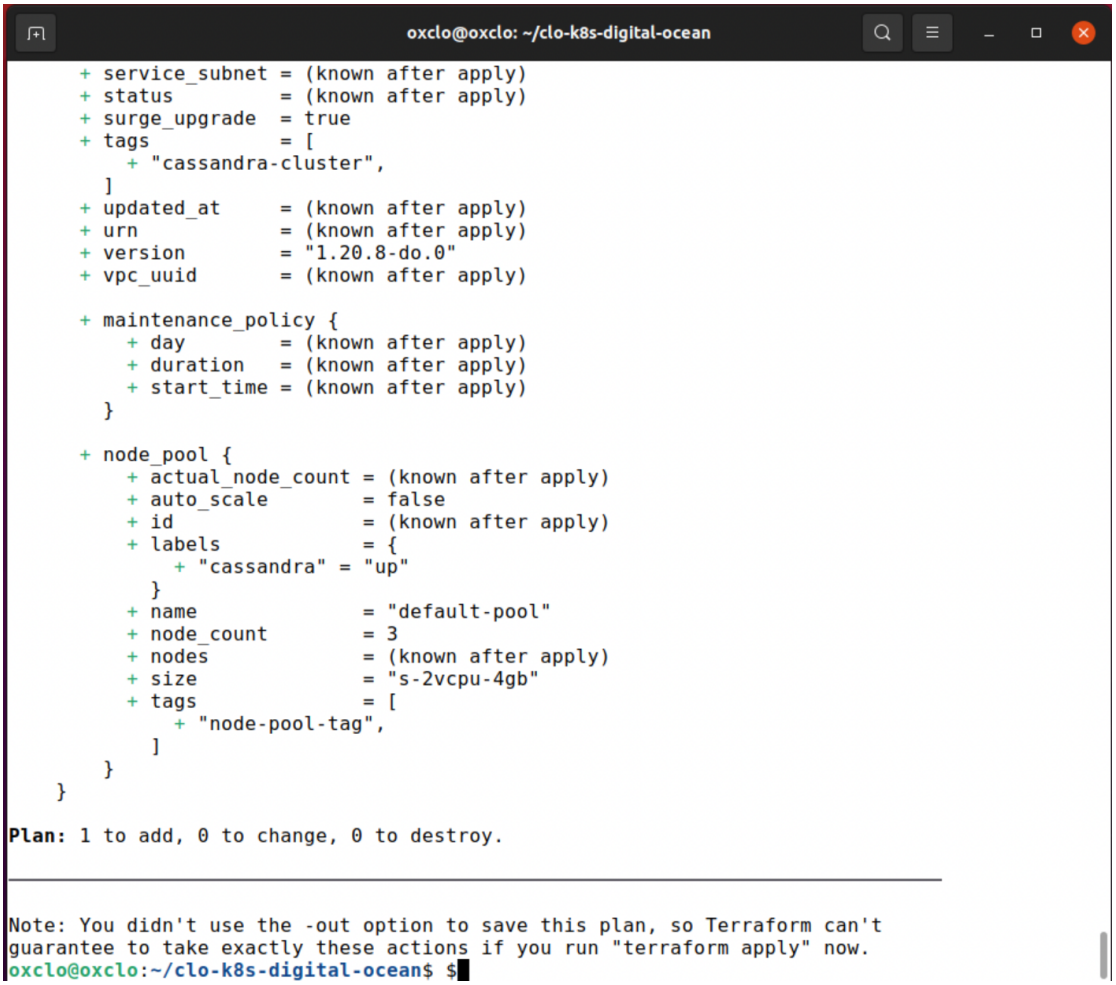
```
git clone https://github.com/pzfreo/clo-k8s-digital-ocean.git

cd clo-k8s-digital-ocean
```

10. Initialise Terraform

```
terraform init
```

11. Check the plan:

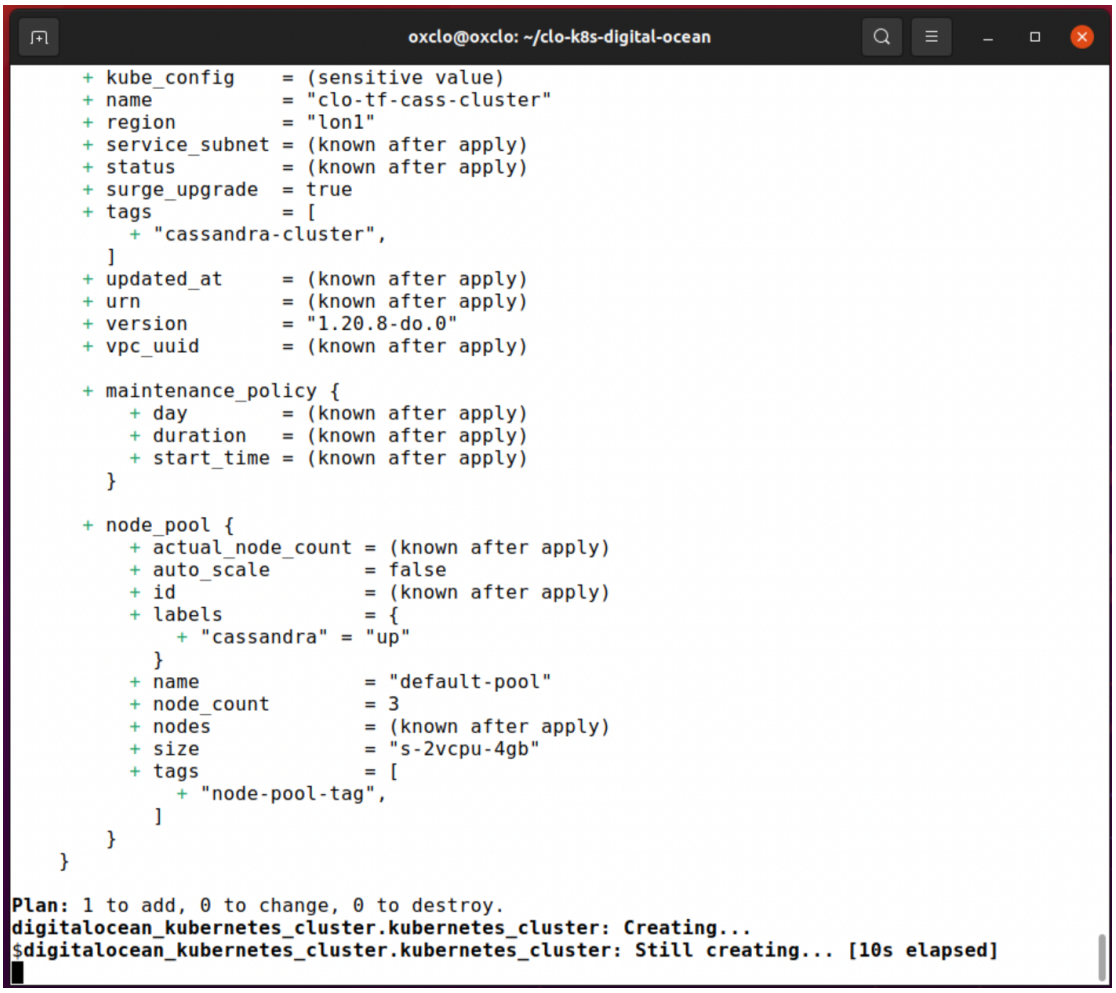
```
terraform plan
```

12. Bootstrap the k8s cluster in Digital Ocean:

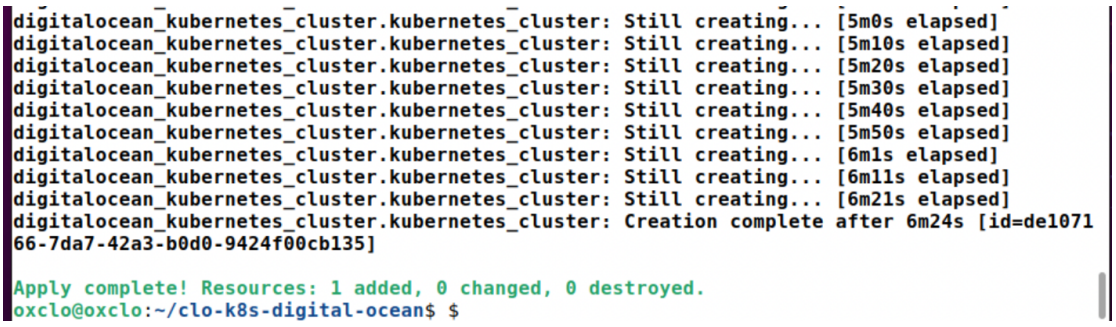
```
terraform apply -auto-approve
```

13. You should see:



14. Go have a coffee… It normally takes ~6 minutes to start up.

15. You can also see what is happening in the Digital Ocean console.



16. Once the cluster is running you need the right credentials for kubectl to connect to it. Luckily there is a cool doctl command to configure that:

```
doctl kubernetes cluster kubeconfig save clo-tf-cass-cluster
```

```
Notice: Adding cluster credentials to kubeconfig file found in
"/home/oxclo/.kube/config"
Notice: Setting current-context to do-lon1-clo-tf-cass-cluster
```

17. Check it works:

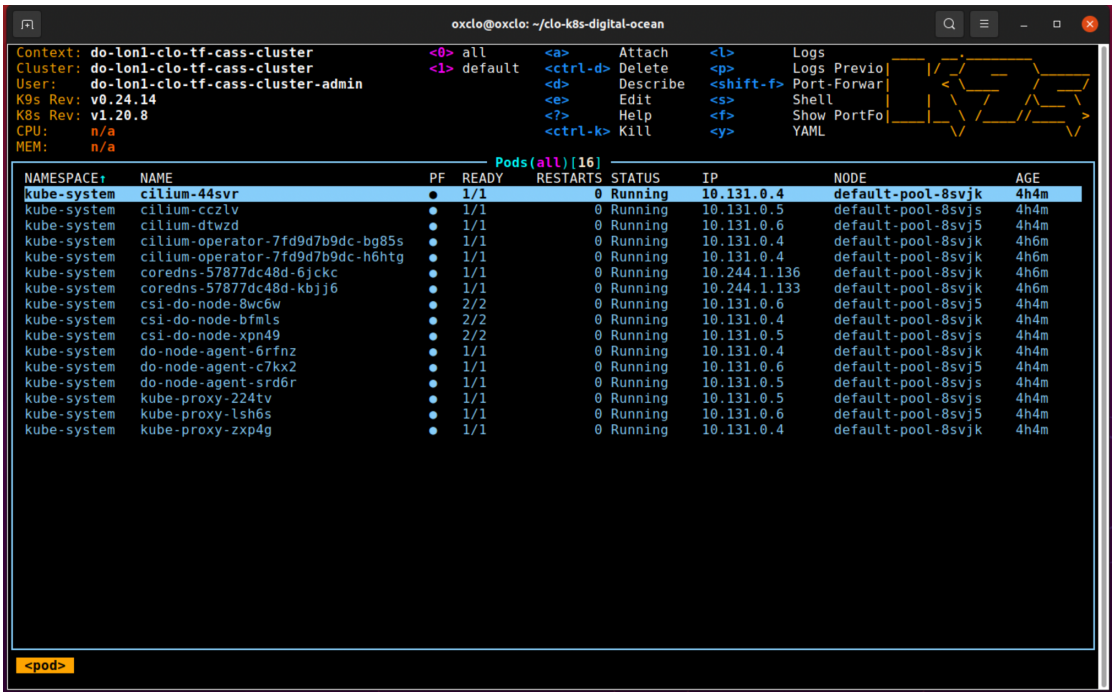
```
kubectl cluster-info
```

You should see something like:

```
Kubernetes control plane is running at
https://de107166-7da7-42a3-b0d0-9424f00cb135.k8s.ondigitalocean.com
CoreDNS is running at
https://de107166-7da7-42a3-b0d0-9424f00cb135.k8s.ondigitalocean.com/api/v1/name
spaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
dump'.
```

18. Try k9s:

```
k9s
```



19. If your screen has no blue lines, it is only showing the default namespace. Hit 0 and it will show all namespaces.

20. Hit '?' to see the help

21. We will come back to this later. For now, hit ':q<ENTER>' or Ctrl-C to quit.

22. There are only two small YAML files required to get Cassandra running. They come from this webpage:

https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

They are in the 'cass-yamls' directory

23. Take a look at the cassandra-service.yaml:

    It is really simple (for YAML!):

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cassandra
  name: cassandra
spec:
  clusterIP: None
  ports:
  - port: 9042
  selector:
    app: cassandra
```

24. This is "kind of" the equivalent of EXPOSE in Docker.

25. The second file is more complex. It basically defines two things:
    a. The cassandra images and config to start a cassandra container

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
```

(That is just the start of that bit)

b. The rest defines the storage that will be needed by these servers in a StatefulSet (https://cloud.google.com/kubernetes-engine/docs/concepts/statefulset)

```
84    volumeClaimTemplates:
85    - metadata:
86        name: cassandra-data
87      spec:
88        accessModes: [ "ReadWriteOnce" ]
89        storageClassName: do-block-storage
90        resources:
91          requests:
92            storage: 2Gi
93
```

Note: This is not the general file from the Kubernetes homepage. Instead, this has been specifically modified to use the Digital Ocean storage backend ('do-block-storage')

You can see what storage classes are available on Kubernetes by doing:

```
kubectl get storageclass
```

```
NAME                         PROVISIONER                 RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
do-block-storage (default)   dobs.csi.digitalocean.com   Delete          Immediate           true                   5h26m
```

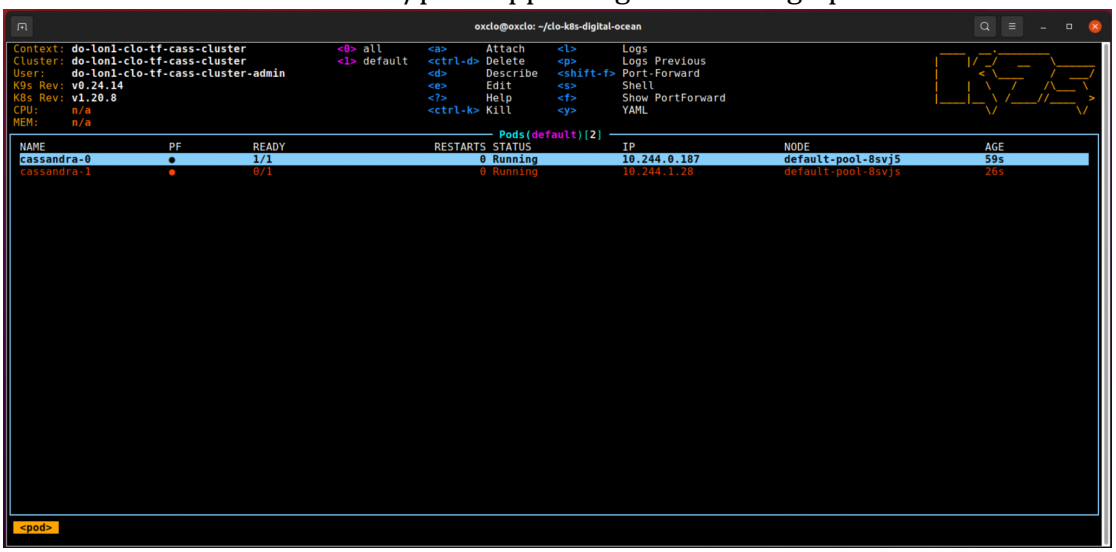

26. Let's deploy Cassandra now:

```
kubectl apply -f cass-yamls/

service/cassandra created
statefulset.apps/cassandra created
```

27. We need to wait a bit for this to start. Basically the system is making API requests to DigitalOcean to provision disks.

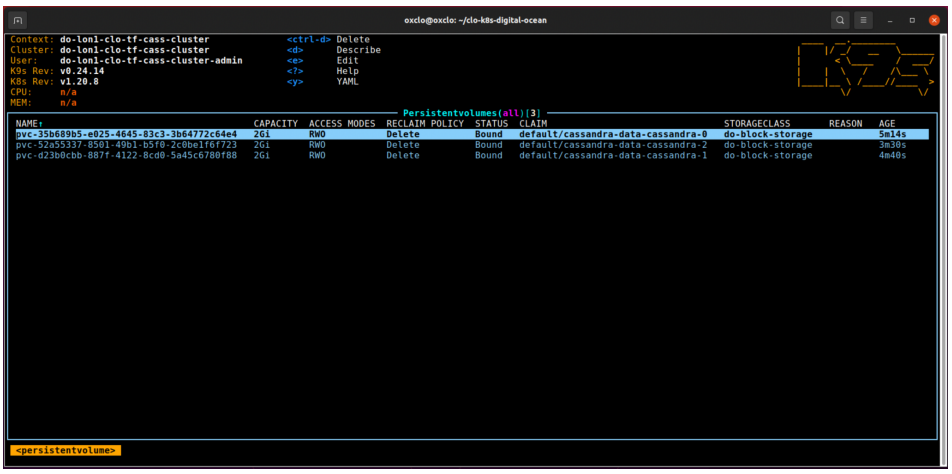
28. Start k9s again. Hit '1' to just show the main default namespace

    You should see the containers/pods appearing and starting up:

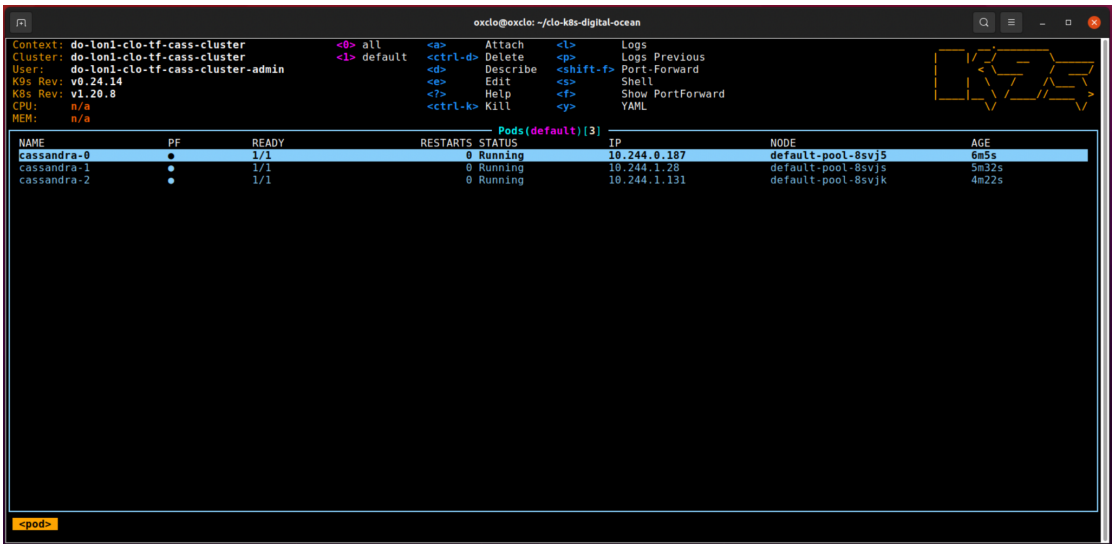


29. You can see different aspects of the Kubernetes cluster (e.g. pods, services, persistent volumes, nodes, etc) by typing ':' and then the type of entity.

    e.g ':pv<ENTER>'



30. After about 5 minutes the cluster should be up and running:



31. Quit k9s.

32. We can now execute commands in the cassandra cluster.
“kubectl exec -ti” is a bit like docker exec. This executes the command that follows -- on the cassandra-0 container instance.

```
kubectl exec -ti cassandra-0 -- nodetool status
```

```
Datacenter: DC1-K8Demo
======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.244.1.109  89.9 KiB   32           52.6%             70e6195b-3629-4d66-a10c-f345015cf68c  Rack1-K8Demo
UN  10.244.0.153  104.55 KiB  32           73.9%             5690399f-6052-439d-b23d-e76e6c152758  Rack1-K8Demo
UN  10.244.0.73   65.81 KiB  32           73.5%             c5a95d05-6891-4586-b416-db5b828b3ccf  Rack1-K8Demo
```

33. Let’s now do a performance test from within the cluster.
First start a pod with some cassandra tools inside it:

```
kubectl apply -f shell.yaml
```

34. Inside kubernetes, the networking is different to the real world. We need to know a host IP to contact the pods on:

```
kubectl describe svc/cassandra

Name:              cassandra
Namespace:         default
Labels:            app=cassandra
Annotations:       Selector:  app=cassandra
Type:              ClusterIP
IP:                None
Port:              <unset>  9042/TCP
TargetPort:        9042/TCP
Endpoints:         10.244.0.153:9042,10.244.0.73:9042,10.244.1.109:9042
Session Affinity:  None
Events:            <none>
```

Choose one of the IP addresses listed as an endpoint. Make a note (in my case 10.244.0.153)

35. Now let's redo that test from within the cluster

```
kubectl exec -ti casstool -- /bin/bash
```

You should see something like:
`root@pool-gdbxlmdop-3oyjy:/#`

From this terminal (with your IP address)

```
cassandra-stress write n=100000 -rate threads=1000 -node 10.244.0.41
```

```
Results:
Op rate                   :   10,242 op/s  [WRITE: 10,242 op/s]
Partition rate            :   10,242 pk/s  [WRITE: 10,242 pk/s]
Row rate                  :   10,242 row/s [WRITE: 10,242 row/s]
Latency mean              :   93.1 ms [WRITE: 93.1 ms]
Latency median            :   71.9 ms [WRITE: 71.9 ms]
Latency 95th percentile   :  303.0 ms [WRITE: 303.0 ms]
Latency 99th percentile   :  405.3 ms [WRITE: 405.3 ms]
Latency 99.9th percentile :  481.3 ms [WRITE: 481.3 ms]
Latency max               :  656.9 ms [WRITE: 656.9 ms]
Total partitions          :    100,000 [WRITE: 100,000]
Total errors              :          0 [WRITE: 0]
Total GC count            : 0
Total GC memory           : 0.000 KiB
Total GC time             :    0.0 seconds
Avg GC time               :    NaN ms
StdDev GC time            :    0.0 ms
Total operation time      : 00:00:09

END
```

36. Do it again just to make sure its warmed up.

37. Now let's add another node into the Kubernetes cluster.

    Quit that container shell (Ctrl-D).

    Edit the `main.tf`

    Make
    `node_count = 4`

38. Check the Terraform plan:

```
terraform plan
digitalocean_kubernetes_cluster.kubernetes_cluster: Refreshing state... [id=de107166-7da7-42a3-b0d0-9424f00cb135]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # digitalocean_kubernetes_cluster.kubernetes_cluster will be updated in-place
  ~ resource "digitalocean_kubernetes_cluster" "kubernetes_cluster" {
        id            = "de107166-7da7-42a3-b0d0-9424f00cb135"
        name          = "clo-tf-cass-cluster"
        tags          = [
            "cassandra-cluster",
        ]
        # (14 unchanged attributes hidden)

      ~ node_pool {
            id              = "0b40908d-1ca3-4643-ad9f-72041d90adab"
            name            = "default-pool"
          ~ node_count      = 3 -> 4
            tags            = [
                "node-pool-tag",
            ]
            # (7 unchanged attributes hidden)
        }
        # (1 unchanged block hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
```

39. Now apply the change:

```
terraform apply -auto-approve
```

40. It will take a minute or so:

```
digitalocean_kubernetes_cluster.kubernetes_cluster:
Modifications complete after 1m42s
[id=de107166-7da7-42a3-b0d0-9424f00cb135]

Apply complete! Resources: 0 added, 1 changed, 0
destroyed.
```

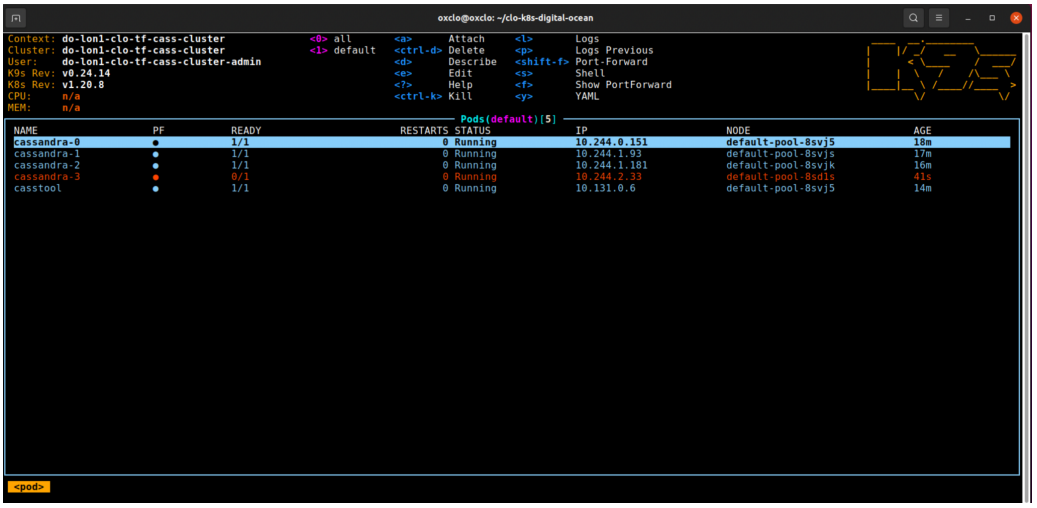
41. You can see the node using:

```
kubectl get nodes
NAME                  STATUS   ROLES    AGE     VERSION
default-pool-8sd1s    Ready    <none>   63s     v1.20.8
default-pool-8svj5    Ready    <none>   4h48m   v1.20.8
default-pool-8svjk    Ready    <none>   4h48m   v1.20.8
default-pool-8svjs    Ready    <none>   4h48m   v1.20.8
```

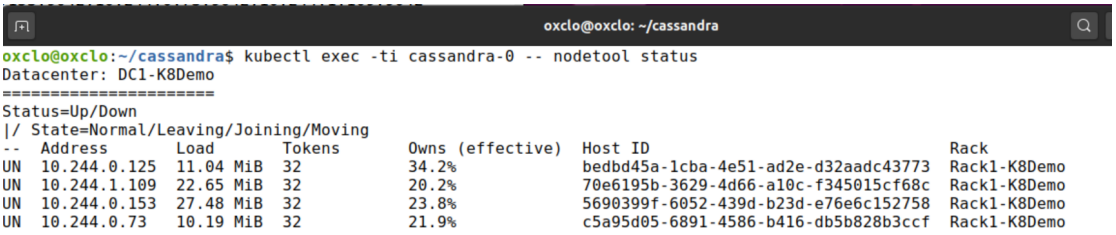

42. Now let's scale cassandra on the cluster:

```
kubectl scale --replicas 4 statefulset/cassandra
```

Wait for the new instance to be live. You can watch in k9s



43. Now lets ask the status from Cassandra:
kubectl exec -ti cassandra-0 -- nodetool status



Notice how the data has already been rebalanced.

44. Rerun the stress test (Steps 29 and 30).

```
Results:
Op rate                   :   11,270 op/s  [WRITE: 11,270 op/s]
Partition rate            :   11,270 pk/s  [WRITE: 11,270 pk/s]
Row rate                  :   11,270 row/s [WRITE: 11,270 row/s]
Latency mean              :   80.9 ms [WRITE: 80.9 ms]
Latency median            :   62.9 ms [WRITE: 62.9 ms]
Latency 95th percentile   :  272.9 ms [WRITE: 272.9 ms]
Latency 99th percentile   :  409.5 ms [WRITE: 409.5 ms]
Latency 99.9th percentile :  514.6 ms [WRITE: 514.6 ms]
Latency max               :  602.9 ms [WRITE: 602.9 ms]
Total partitions          :    100,000 [WRITE: 100,000]
Total errors              :          0 [WRITE: 0]
Total GC count            : 0
Total GC memory           : 0.000 KiB
Total GC time             :    0.0 seconds
Avg GC time               :    NaN ms
StdDev GC time            :    0.0 ms
Total operation time      : 00:00:08

END
```

I saw a reasonable improvement in performance.

45. Sometimes the pods get bunched up on one node and not evenly spread. However, there is a cool tool called the Kubernetes Descheduler:

    https://github.com/kubernetes-sigs/descheduler

46. Congratulations - we have deployed cassandra, scaled it, tested it and increased performance all in a kubernetes cluster.

47. Finally let's clean up.

48. Firstly, let's delete our cassandra cluster.

```
kubectl delete -f cass-yamls/
```

49. Delete our casstool pod:

```
kubectl delete pod/casstool
```

50. Delete the volumes / volume claims:

```
kubectl delete persistentvolumeclaim -l app=cassandra
```


51. Delete the kubernetes cluster:

```
terraform destroy -auto-approve

Plan: 0 to add, 0 to change, 1 to destroy.
digitalocean_kubernetes_cluster.kubernetes_cluster:
Destroying... [id=de107166-7da7-42a3-b0d0-9424f00cb135]
digitalocean_kubernetes_cluster.kubernetes_cluster:
Destruction complete after 0s

Destroy complete! Resources: 1 destroyed.
```

52. Check in the DigitalOcean Control panel for any remaining resources and just check you haven't left anything running!

53. This lab is done! Congratulations.