

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**

The screenshot shows a web form for generating a new personal access token. At the top right is a close button (X). Below it is the title "New personal access token". The next section is titled "Token name" with a text input field containing "oxclo" and a green checkmark icon. The following section is titled "Select scopes" with two checkboxes: "Read (default)" (checked) and "Write (optional)". Below this is a link to "personal access token documentation". A large blue button at the bottom is labeled "Generate 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
```



```
oxclo@oxclo: ~/clo-k8s-digital-ocean
+ service_subnet = (known after apply)
+ status         = (known after apply)
+ surge_upgrade = true
+ tags          = [
    + "cassandra-cluster",
]
+ updated_at     = (known after apply)
+ urn           = (known after apply)
+ version       = "1.20.8-do.0"
+ vpc_uuid      = (known after apply)

+ maintenance_policy {
    + day        = (known after apply)
    + duration   = (known after apply)
    + start_time = (known after apply)
}

+ node_pool {
    + actual_node_count = (known after apply)
    + auto_scale        = false
    + id                = (known after apply)
    + labels            = {
        + "cassandra" = "up"
    }
    + name              = "default-pool"
    + node_count        = 3
    + nodes             = (known after apply)
    + size              = "s-2vcpu-4gb"
    + tags              = [
        + "node-pool-tag",
    ]
}
}

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

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
oxclo@oxclo:~/clo-k8s-digital-ocean$
```

12. Bootstrap the k8s cluster in Digital Ocean:

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



13. You should see:

The screenshot shows a terminal window titled "oxclo@oxclo: ~/clo-k8s-digital-ocean". The command entered is "doctl kubernetes cluster kubeconfig save clo-tf-cass-cluster". The output shows the configuration for the cluster, including fields like kube_config, name, region, service_subnet, status, surge_upgrade, tags, updated_at, urn, version, vpc_uuid, maintenance_policy, node_pool, and Plan. The node_pool section details actual_node_count, auto_scale, id, labels, name, node_count, nodes, size, and tags. The Plan section indicates 1 to add, 0 to change, 0 to destroy. The final message shows the cluster being created, with a note that it's still creating after 10 seconds elapsed.

```
+ kube_config      = (sensitive value)
+ name             = "clo-tf-cass-cluster"
+ region           = "lon1"
+ service_subnet   = (known after apply)
+ status            = (known after apply)
+ surge_upgrade    = true
+ tags              = [
  + "cassandra-cluster",
]
+ updated_at       = (known after apply)
+ urn               = (known after apply)
+ version           = "1.20.8-do.0"
+ vpc_uuid          = (known after apply)

+ maintenance_policy {
  + day              = (known after apply)
  + duration         = (known after apply)
  + start_time       = (known after apply)
}

+ node_pool {
  + actual_node_count = (known after apply)
  + auto_scale        = false
  + id                = (known after apply)
  + labels             = {
    + "cassandra" = "up"
  }
  + name              = "default-pool"
  + node_count        = 3
  + nodes              = (known after apply)
  + size               = "s-2vcpu-4gb"
  + tags               = [
    + "node-pool-tag",
  ]
}
}

Plan: 1 to add, 0 to change, 0 to destroy.
digitalocean_kubernetes_cluster.kubernetes_cluster: Creating...
$digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [10s elapsed]
```

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.

The screenshot shows the Digital Ocean console interface. The main area displays a log of the cluster creation process, with messages indicating the cluster is still creating for various amounts of time (e.g., 5m0s, 5m10s, 5m20s, etc.). The final message states that the creation is complete after 6m24s, with a unique ID provided. Below the log, a summary message indicates "Apply complete! Resources: 1 added, 0 changed, 0 destroyed." and the command "oxclo@oxclo:~/clo-k8s-digital-ocean\$".

```
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m0s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m10s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m20s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m30s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m40s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [5m50s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [6m1s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [6m11s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Still creating... [6m21s elapsed]
digitalocean_kubernetes_cluster.kubernetes_cluster: Creation complete after 6m24s [id=de107166-7da7-42a3-b0d0-9424f00cb135]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
oxclo@oxclo:~/clo-k8s-digital-ocean$
```

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
```

NAMESPACE	NAME	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
kube-system	cilium-44svr	●	1/1	0	Running	10.131.0.4	default-pool-8svjk	4h4m
kube-system	cilium-cczlv	●	1/1	0	Running	10.131.0.5	default-pool-8svj5	4h4m
kube-system	cilium-dtwzd	●	1/1	0	Running	10.131.0.6	default-pool-8svj5	4h4m
kube-system	cilium-operator-7fd9d7b9dc-bg85s	●	1/1	0	Running	10.131.0.4	default-pool-8svjk	4h6m
kube-system	cilium-operator-7fd9d7b9dc-h6htg	●	1/1	0	Running	10.131.0.4	default-pool-8svjk	4h6m
kube-system	coredns-57877dc48d-6jckc	●	1/1	0	Running	10.244.1.136	default-pool-8svjk	4h6m
kube-system	coredns-57877dc48d-kbjj6	●	1/1	0	Running	10.244.1.133	default-pool-8svjk	4h6m
kube-system	csi-do-node-8wc6w	●	2/2	0	Running	10.131.0.6	default-pool-8svj5	4h4m
kube-system	csi-do-node-bfm1s	●	2/2	0	Running	10.131.0.4	default-pool-8svjk	4h4m
kube-system	csi-do-node-xpn49	●	2/2	0	Running	10.131.0.5	default-pool-8svj5	4h4m
kube-system	do-node-agent-6rfnz	●	1/1	0	Running	10.131.0.4	default-pool-8svjk	4h4m
kube-system	do-node-agent-c7kx2	●	1/1	0	Running	10.131.0.6	default-pool-8svj5	4h4m
kube-system	do-node-agent-srd6r	●	1/1	0	Running	10.131.0.5	default-pool-8svj5	4h4m
kube-system	kube-proxy-224tv	●	1/1	0	Running	10.131.0.5	default-pool-8svj5	4h4m
kube-system	kube-proxy-lsh6s	●	1/1	0	Running	10.131.0.6	default-pool-8svj5	4h4m
kube-system	kube-proxy-zxp4g	●	1/1	0	Running	10.131.0.4	default-pool-8svjk	4h4m

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!):

```
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:

- The `cassandra` images and config to start a `cassandra` container

```
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



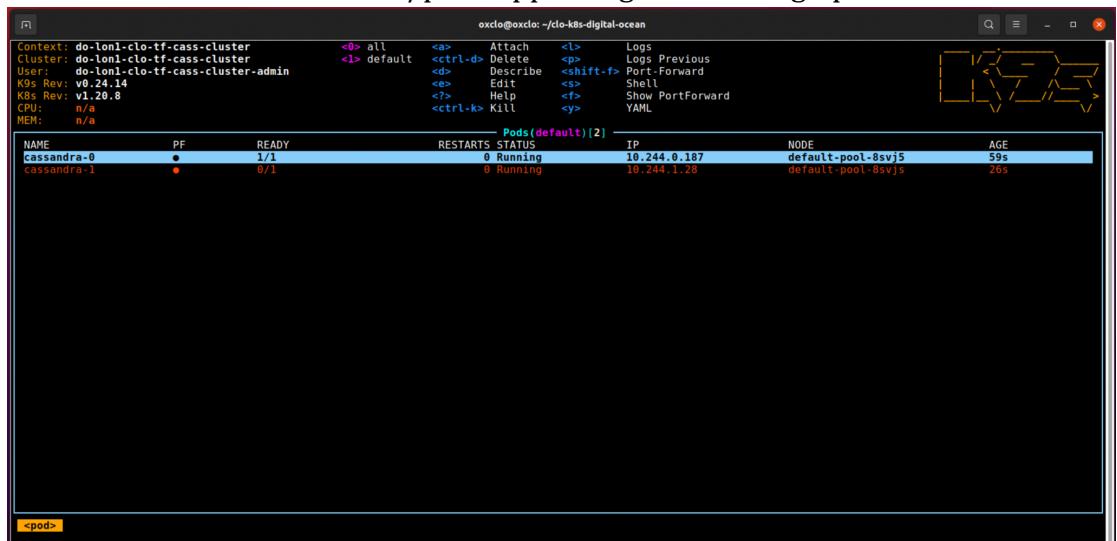
26. Let's deploy Cassandra now:

```
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

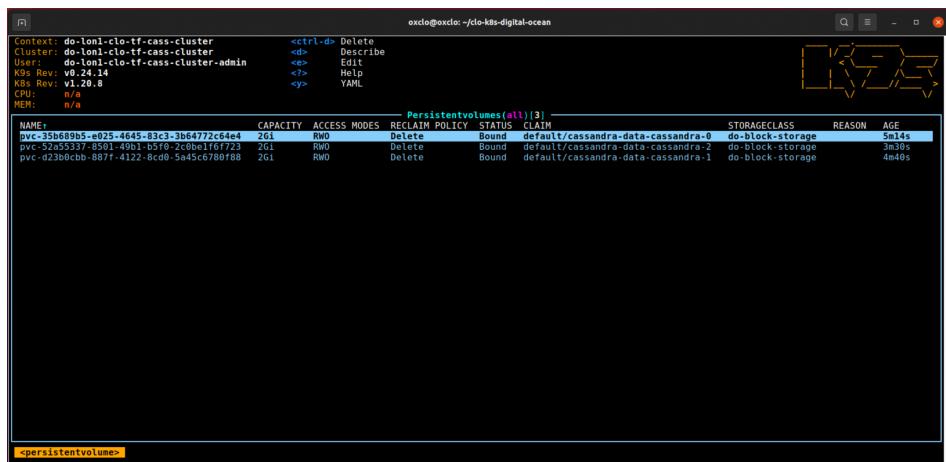
You should see the containers/pods appearing and starting up:



NAME	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
cassandra-0	●	1/1	0	Running	10.244.0.187	default-pool-8svj5	59s
cassandra-1	●	0/1	0	Running	10.244.1.28	default-pool-8svj5	26s

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>'



NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
pvc-d35689b5-e023-4845-83c3-3b64772c54e3	2Gi	RWO	Delete	Bound	default/cassandra-data-cassandra-0	do-block-storage	claim	5m44s
pvc-52555337-8501-49b1-b5f0-2d0be1f6f723	2Gi	RWO	Delete	Bound	default/cassandra-data-cassandra-2	do-block-storage	claim	5m30s
pvc-d23bb0ccb-887f-4122-8cd8-5a45c6780ff88	2Gi	RWO	Delete	Bound	default/cassandra-data-cassandra-1	do-block-storage	claim	4m40s

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

```
oxclo@oxclo: ~/clo-k8s-digital-ocean
Context: do-lon1-clo-tf-cass-cluster      <0> all    <a> Attach  <l> Logs
Cluster: do-lon1-clo-tf-cass-cluster      <1> default <crtl-d> Delete  <p> Logs Previous
User:   do-lon1-clo-tf-cass-cluster-admin  <d> Describe <shift-f> Port-Forward
K9s Rev: v0.24.14                         <e> Edit    <s> Shell
K8s Rev: v1.20.8                          <?> Help    <f> Show PortForward
CPU:   n/a                                <crtl-k> Kill   </> YAML
MEM:   n/a

NAME          PF    READY   RESTARTS STATUS      IP           NODE        AGE
cassandra-0   ●    1/1     0       Running   10.244.0.187 default-pool-8svj5  6m5s
cassandra-1   ●    1/1     0       Running   10.244.1.28  default-pool-8svj5  5m32s
cassandra-2   ●    1/1     0       Running   10.244.1.131 default-pool-8svjk  4m22s

<pod>
```

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:

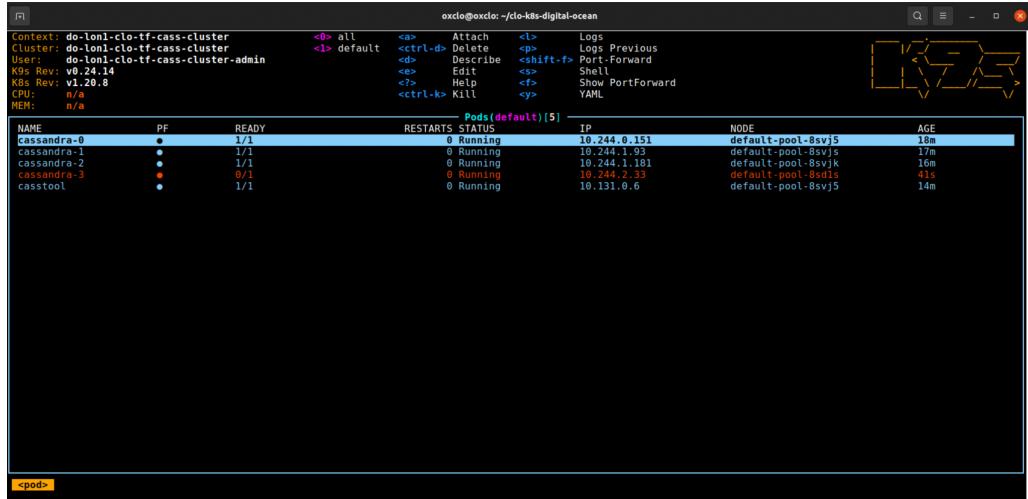
```
kubectl get nodes
NAME           STATUS   ROLES      AGE     VERSION
default-pool-8sd1s Ready    <none>    63s    v1.20.8
default-pool-8svj5 Ready    <none>    4h48m   v1.20.8
default-pool-8svjk Ready   <none>    4h48m   v1.20.8
default-pool-8svjs Ready   <none>    4h48m   v1.20.8
```



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

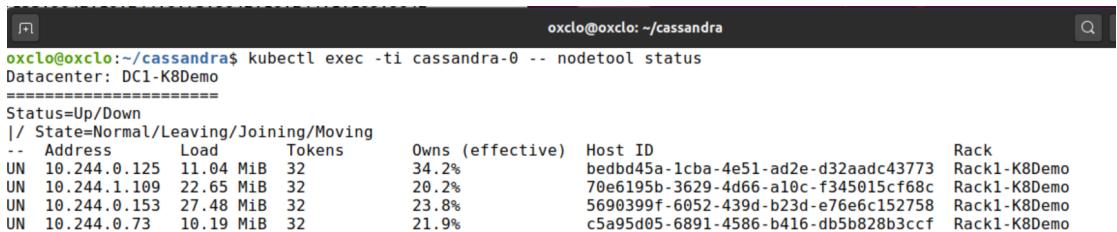


The screenshot shows the k9s application interface. At the top, there is a command-line interface with various flags and options. Below it is a table titled "Pods(default)(5)" showing the status of five pods. The columns are NAME, PF, READY, RESTARTS, STATUS, IP, NODE, and AGE. The pods listed are cassandra-0, cassandra-1, cassandra-2, cassandra-3, and casstool. The casstool pod is currently running. The cassandra pods have 0 restarts and are in a "Running" state. The IP column lists the external IP addresses of the pods. The NODE column shows they are running on default-pool-8svj5 nodes. The AGE column indicates the time since the pods were created.

NAME	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
cassandra-0	●	1/1	0	Running	10.244.0.151	default-pool-8svj5	17m
cassandra-1	●	1/1	0	Running	10.244.1.93	default-pool-8svj5	17m
cassandra-2	●	1/1	0	Running	10.244.1.101	default-pool-8svj5	16m
cassandra-3	●	0/1	0	Running	10.244.2.33	default-pool-8svj5	15s
casstool	●	1/1	0	Running	10.131.0.6	default-pool-8svj5	14m

43. Now lets ask the status from Cassandra:

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



The screenshot shows a terminal window with the command "kubectl exec -ti cassandra-0 -- nodetool status" run. The output shows the status of the Cassandra cluster across four nodes. The output includes information about the rack each node is in, its host ID, and token ranges. The data is rebalanced, with each node holding approximately 25% of the tokens.

```
oxclo@oxclo:~/cassandra$ 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 10.244.0.125  11.04 MiB  32        34.2%       bedbd45a-1cba-4e51-ad2e-d32aadc43773  Rack1-K8Demo
UN 10.244.1.109  22.65 MiB  32        20.2%       70e6195b-3629-4d66-a10c-f345015cf68c  Rack1-K8Demo
UN 10.244.0.153  27.48 MiB  32        23.8%       5690399f-6052-439d-b23d-e76e6c152758  Rack1-K8Demo
UN 10.244.0.73   10.19 MiB  32        21.9%       c5a95d05-6891-4586-b416-db5b828b3ccf  Rack1-K8Demo
```

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.

