Exercise 14b

Create a Kubernetes Cluster in DigitalOcean and Deploy Cassandra

Prior Knowledge
Unix Command Line Shell
YAML
Completion of Ex 14a

Learning Objectives
See how Cassandra replicates
Introduction to Kubernetes

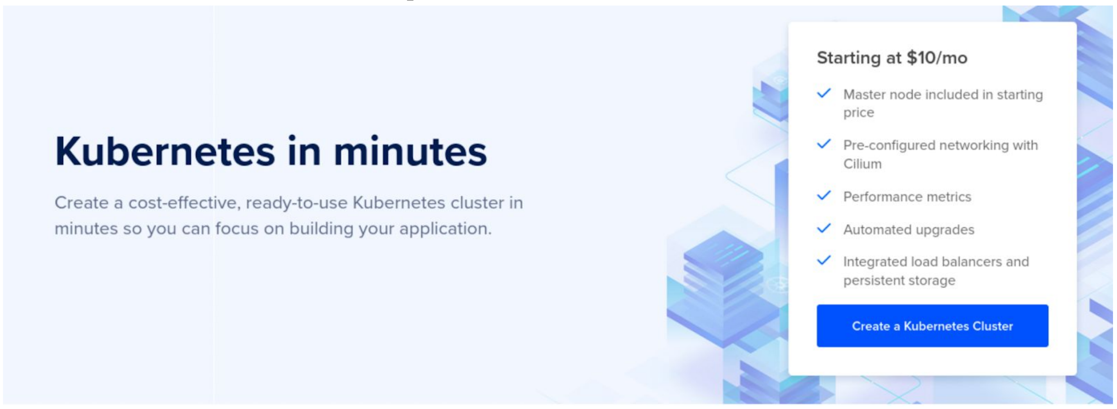
Software Requirements
Browser
kubectl

Overview

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

1. If you have left the cluster running from Ex14a, go straight to **step 2**

   a. Otherwise redo the steps to create a cluster:



Kubernetes in minutes

Create a cost-effective, ready-to-use Kubernetes cluster in minutes so you can focus on building your application.

Starting at $10/mo

- Master node included in starting price
- Pre-configured networking with Cilium
- Performance metrics
- Automated upgrades
- Integrated load balancers and persistent storage

   b. Make sure you install the monitoring 1-click app.

d. Download the config file, then:
```
mv ~/Downloads/k8s-cass-kubeconfig.yaml ~/.kube/
```

In your terminal window:
```
export KUBECONFIG=~/.kube/k8s-cass-kubeconfig.yaml
```

(There are also other things we can do, but this works fine)

e. Check it works:

```
kubectl get all
```

You should see something like:

```
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.245.0.1   <none>        443/TCP   45m
```

2. Now let's make a directory for our Cassandra YAMLs:

```
mkdir ~/cassandra
```

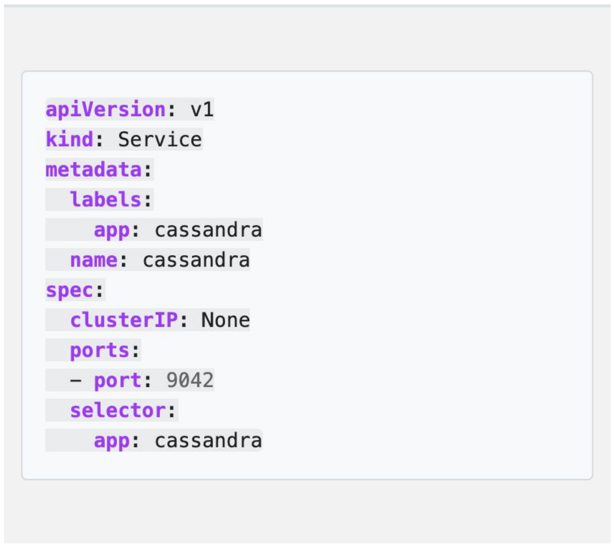
3. There are only two small YAML files required to get Cassandra running. They come from this webpage:
https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

Because we need to modify one of them, let's download them:

```
cd ~/cassandra
wget https://k8s.io/examples/application/cassandra/cassandra-service.yaml
wget https://k8s.io/examples/application/cassandra/cassandra-statefulset.yaml
```

4. 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
```

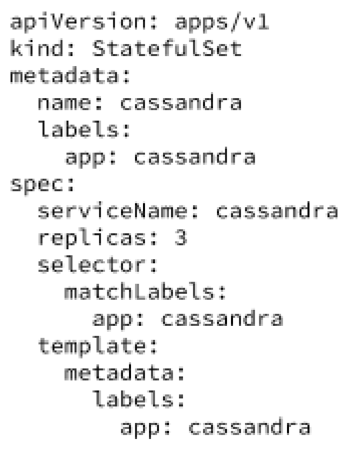
5. This is “kind of” the equivalent of EXPOSE in Docker. You could compare this to the one in the “hello-kubernetes” app if you like.

6. Let’s deploy this:
```
kubectl apply -f cassandra-service.yaml

service/cassandra created
```

7. Check if it is happy:
```
kubectl get svc

NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
cassandra    ClusterIP   None         <none>        9042/TCP   29s
kubernetes   ClusterIP   10.245.0.1   <none>        443/TCP    28m
```

8. You should see the cassandra service running alongside the kubernetes master. We need this to be in place **before** we start the next part because the different pods need to be able to access each other via port 9042 for the cluster to form.

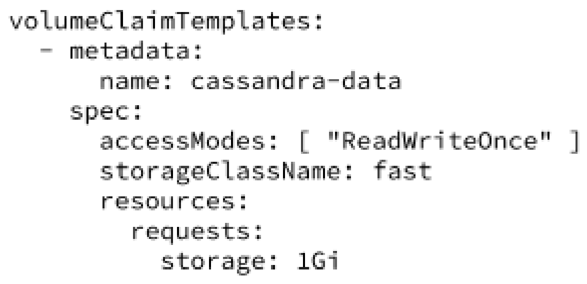
9. The second file is more complex. It basically defines three things:
    a. The cassandra images and config to start a cassandra container



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



```
volumeClaimTemplates:
  - metadata:
      name: cassandra-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: fast
      resources:
        requests:
          storage: 1Gi
```

c. Defines a StorageClass for deploying into Minikube (a kubernetes distro designed to run on developers' machines).

```
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: k8s.io/minikube-hostpath
parameters:
  type: pd-ssd
```

10. This YAML will **not work** as is. That is because we need the Kubernetes cluster to request disk from DigitalOcean specifically, not from minikube.

11. Let's see what StorageClass is available in our DO cluster:

```
kubectl get storageclass
```

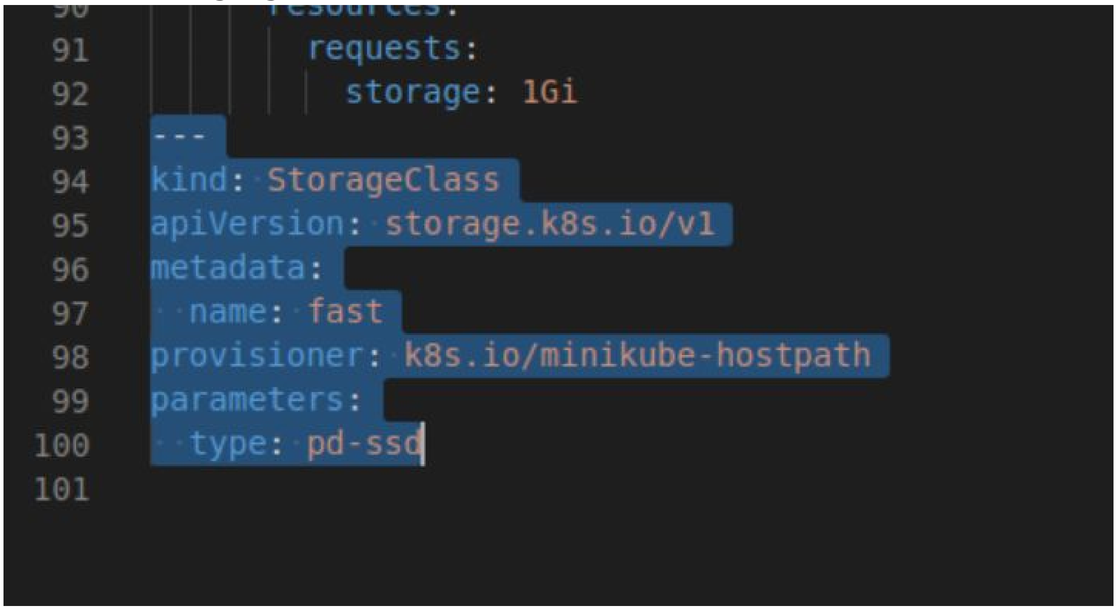
```
NAME                       PROVISIONER                 RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
do-block-storage (default)   dobs.csi.digitalocean.com   Delete          Immediate           true                   5h26m
```

This is a pre-configured storage class for DO called `do-block-storage`

12. We need to edit the YAML file:

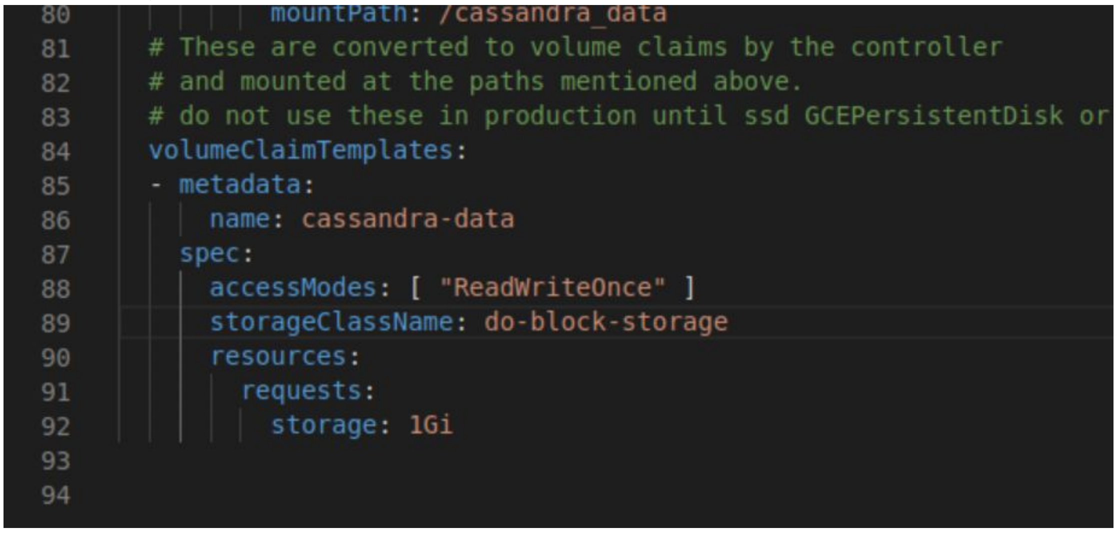
```
code ~/cassandra/cassandra-statefulset.yaml
```

13. Firstly, lets delete the StorageClass section:
Delete all the highlighted lines:



```
        resources:
          requests:
            storage: 1Gi
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: k8s.io/minikube-hostpath
parameters:
  type: pd-ssd
```

14. Secondly, change the storageClassName from **fast** to **do-block-storage** (which we just identified above).



```
            mountPath: /cassandra_data
      # These are converted to volume claims by the controller
      # and mounted at the paths mentioned above.
      # do not use these in production until ssd GCEPersistentDisk or
      volumeClaimTemplates:
      - metadata:
          name: cassandra-data
        spec:
          accessModes: [ "ReadWriteOnce" ]
          storageClassName: do-block-storage
          resources:
            requests:
              storage: 1Gi
```

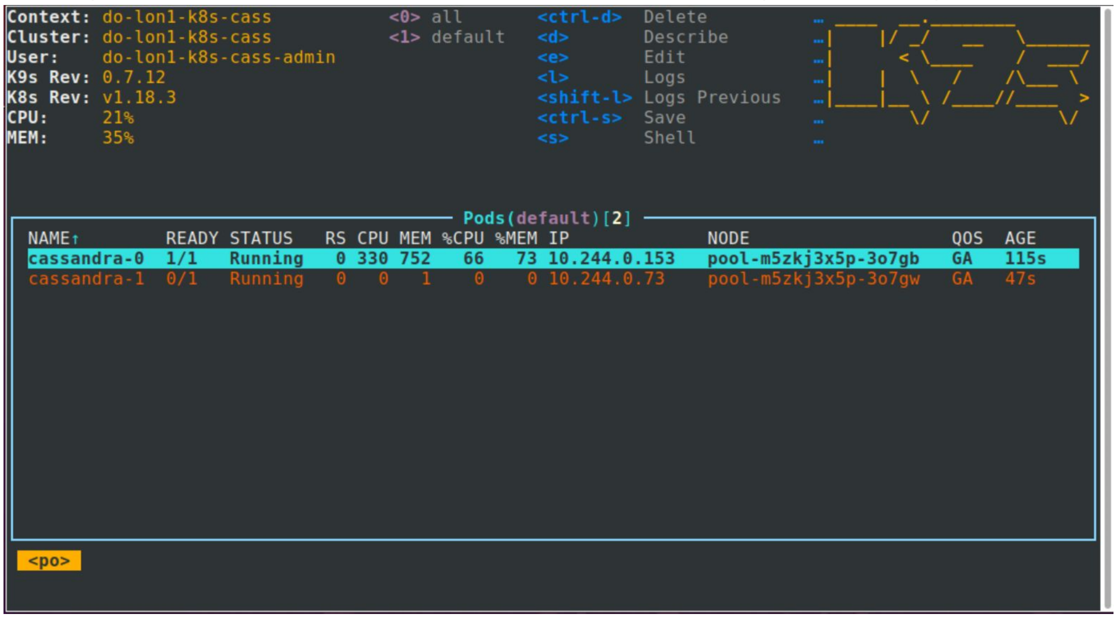
15. Save the file

16. Let's deploy this now:

```
kubectl apply -f cassandra-statefulset.yaml

statefulset.apps/cassandra created
```

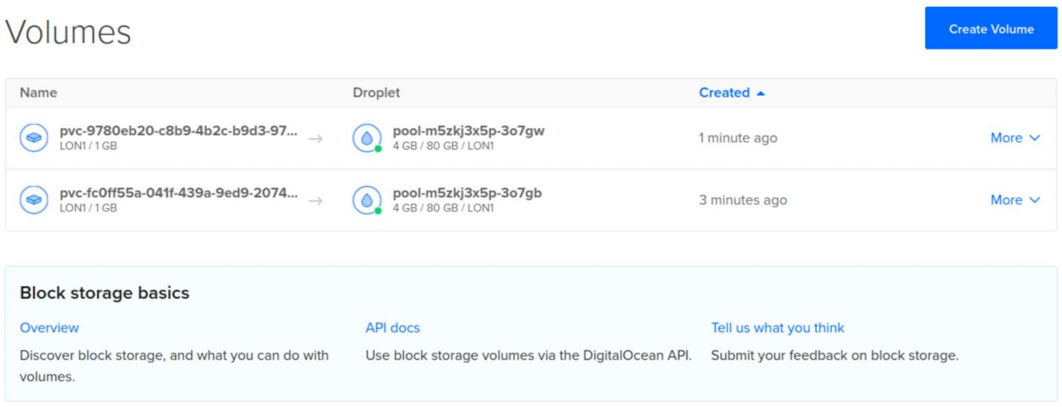
17. We need to wait a bit for this to start. Basically the system is making API requests to DigitalOcean to provision disks.

    If you go to k9s, you should see the containers appearing and starting up:



```
Context:  do-lon1-k8s-cass          <0> all        <ctrl-d>    Delete
Cluster:  do-lon1-k8s-cass          <1> default    <d>         Describe
User:     do-lon1-k8s-cass-admin                   <e>         Edit
K9s Rev:  0.7.12                                   <l>         Logs
K8s Rev:  v1.18.3                                  <shift-l>   Logs Previous
CPU:      21%                                      <ctrl-s>    Save
MEM:      35%                                      <s>         Shell

                          Pods(default)[2]
 NAME↑        READY STATUS   RS CPU MEM %CPU %MEM IP           NODE                   QOS  AGE
 cassandra-0  1/1   Running   0 330 752   66   73 10.244.0.153 pool-m5zkj3x5p-3o7gb   GA   115s
 cassandra-1  0/1   Running   0   0   1    0    0 10.244.0.73  pool-m5zkj3x5p-3o7gw   GA   47s

 <po>
```

19. If you go to the DigitalOcean control panel you will see **Volumes** being created:



Volumes

Name	Droplet	Created ▲	
pvc-9780eb20-c8b9-4b2c-b9d3-97... LON1 / 1 GB	pool-m5zkj3x5p-3o7gw 4 GB / 80 GB / LON1	1 minute ago	More
pvc-fc0ff55a-041f-439a-9ed9-2074... LON1 / 1 GB	pool-m5zkj3x5p-3o7gb 4 GB / 80 GB / LON1	3 minutes ago	More

Block storage basics

Overview
Discover block storage, and what you can do with volumes.

API docs
Use block storage volumes via the DigitalOcean API.

Tell us what you think
Submit your feedback on block storage.

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

```
kubectl get all

NAME              READY   STATUS    RESTARTS   AGE
pod/cassandra-0   1/1     Running   0          10m
pod/cassandra-1   1/1     Running   0          9m21s
pod/cassandra-2   1/1     Running   0          7m37s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/cassandra    ClusterIP   None         <none>        9042/TCP   5h14m
service/kubernetes   ClusterIP   10.245.0.1   <none>        443/TCP    5h42m

NAME                         READY   AGE
statefulset.apps/cassandra   3/3     10m
```

21. 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
```

22. Now let's stress test the server. We can do it over the network between our machine and the DO cluster, but be warned this isn't terribly efficient.

    First, let's forward the cluster port 9042 to local port 9040. We are choosing 9040 locally because we might still be running cassandra locally on 9042 and we want to be sure we are talking to the remote cluster:

    ```
    kubectl port-forward pods/cassandra-0  9040:9042
    ```

    You should see:

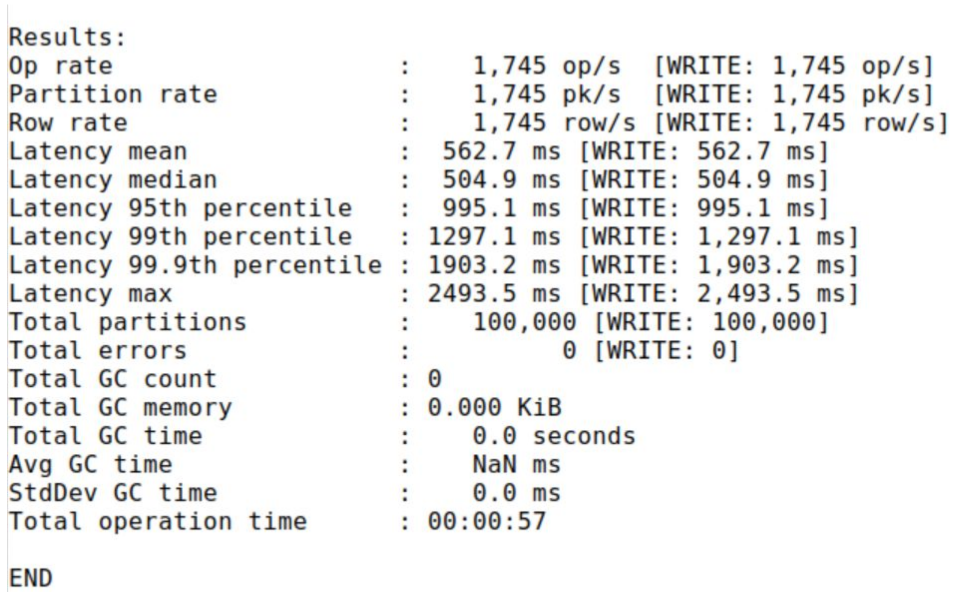
    ```
    Forwarding from 127.0.0.1:9040 -> 9042
    Forwarding from [::1]:9040 -> 9042
    ```

    Leave that window and ***start a new terminal window.***

23. Now let's to a stress test on port 9040:

    ```
    cassandra-stress write n=100000 -port native=9040 -rate threads=1000
    ```

    You may see some Java exceptions in the logs (due to networking challenges). Eventually it should finish with something like:



```
Results:
Op rate                   :    1,745 op/s  [WRITE: 1,745 op/s]
Partition rate            :    1,745 pk/s  [WRITE: 1,745 pk/s]
Row rate                  :    1,745 row/s [WRITE: 1,745 row/s]
Latency mean              :  562.7 ms [WRITE: 562.7 ms]
Latency median            :  504.9 ms [WRITE: 504.9 ms]
Latency 95th percentile   :  995.1 ms [WRITE: 995.1 ms]
Latency 99th percentile   : 1297.1 ms [WRITE: 1,297.1 ms]
Latency 99.9th percentile : 1903.2 ms [WRITE: 1,903.2 ms]
Latency max               : 2493.5 ms [WRITE: 2,493.5 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      : 00:00:57

END
```

24. Kill the port forwarding process (Ctrl-C)

25. Let's now do the same test from within the cluster. To do this, we can start a new pod and get shell access:

    ```
    kubectl apply -f https://raw.githubusercontent.com/pzfreo/ox-clo/master/code/cass-tools/shell.yaml
    ```

27. This is just an Ubuntu container with cassandra tools installed.
    Let's check it started:

```
kubectl get all

NAME                 READY   STATUS    RESTARTS   AGE
pod/cassandra-0      1/1     Running   0          4h27m
pod/cassandra-1      1/1     Running   0          4h26m
pod/cassandra-2      1/1     Running   0          4h24m
pod/casstool         1/1     Running   0          23m

NAME                    TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)    AGE
service/cassandra    ClusterIP   None          <none>
9042/TCP   9h
service/kubernetes   ClusterIP   10.245.0.1    <none>
443/TCP    9h

NAME                          READY   AGE
statefulset.apps/cassandra   3/3     4h27m
```

28. 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)

29. We can now get a command-line in that container:

```
kubectl exec -it casstool -- bash
```

30. Now let's redo that test from within the cluster (with your IP address)

```
cassandra-stress write n=100000 -rate threads=1000 -node 10.244.0.153
```

Unless you have a massively fast connection from your machine to the DO datacentre, you should see much better performance now:

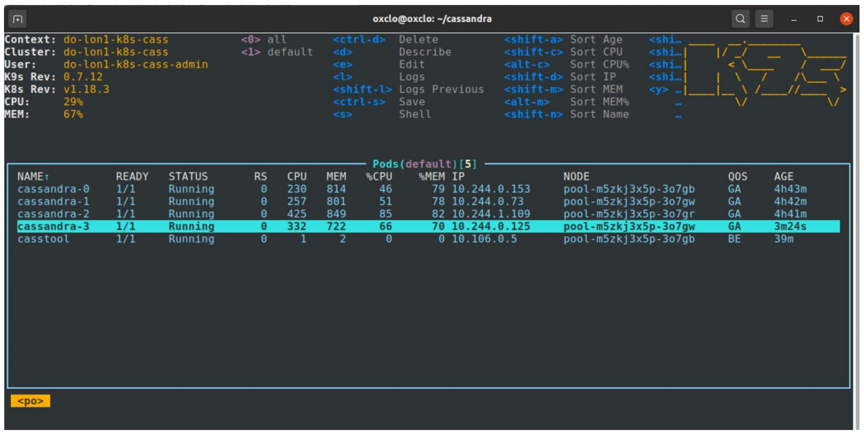
```
Results:
Op rate                   :    7,527 op/s  [WRITE: 7,527 op/s]
Partition rate            :    7,527 pk/s  [WRITE: 7,527 pk/s]
Row rate                  :    7,527 row/s [WRITE: 7,527 row/s]
Latency mean              :  123.7 ms [WRITE: 123.7 ms]
Latency median            :   87.9 ms [WRITE: 87.9 ms]
Latency 95th percentile   :  419.7 ms [WRITE: 419.7 ms]
Latency 99th percentile   :  649.6 ms [WRITE: 649.6 ms]
Latency 99.9th percentile : 1015.0 ms [WRITE: 1,015.0 ms]
Latency max               : 1106.2 ms [WRITE: 1,106.2 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      : 00:00:13

END
```

31. Now let's add another node into the cluster.

Quit that container shell (Ctrl-D) and execute this command:

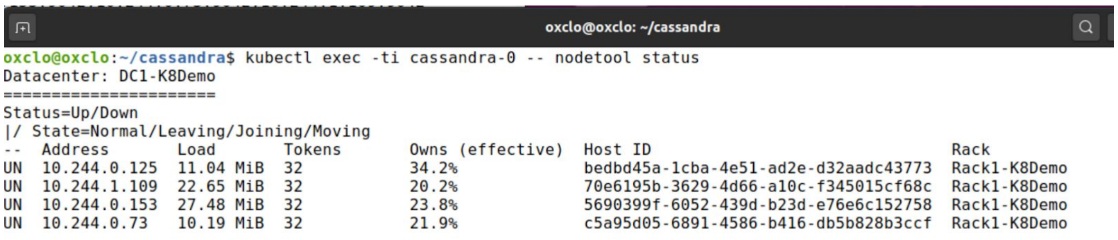
```
kubectl scale --replicas 4 statefulset/cassandra
```

Wait for the new instance to be live:



```
oxclo@oxclo: ~/cassandra
Context: do-lon1-k8s-cass          <0> all       <ctrl-d>   Delete          <shift-a> Sort Age
Cluster: do-lon1-k8s-cass          <1> default   <d>        Describe        <shift-c> Sort CPU
User:    do-lon1-k8s-cass-admin                  <e>        Edit            <alt-c>   Sort CPU%
K9s Rev: 0.7.12                                  <l>        Logs            <shift-d> Sort IP
K8s Rev: v1.18.3                                 <shift-l>  Logs Previous   <shift-m> Sort MEM
CPU:     29%                                     <ctrl-s>   Save            <alt-m>   Sort MEM%
MEM:     67%                                     <s>        Shell           <shift-n> Sort Name

                                   Pods(default)[5]
 NAME↑          READY   STATUS    RS   CPU   MEM  %CPU   %MEM IP             NODE                    QOS  AGE
 cassandra-0    1/1     Running    0   230   814    46     79 10.244.0.153   pool-m5zkj3x5p-3o7gb    GA   4h43m
 cassandra-1    1/1     Running    0   257   801    51     78 10.244.0.73    pool-m5zkj3x5p-3o7gw    GA   4h42m
 cassandra-2    1/1     Running    0   425   849    85     82 10.244.1.109   pool-m5zkj3x5p-3o7gr    GA   4h41m
 cassandra-3    1/1     Running    0   332   722    66     70 10.244.0.125   pool-m5zkj3x5p-3o7gw    GA   3m24s
 casstool       1/1     Running    0     1     2     0      0 10.106.0.5     pool-m5zkj3x5p-3o7gb    BE   39m

<po>
```

33. Now lets ask the status from Cassandra:

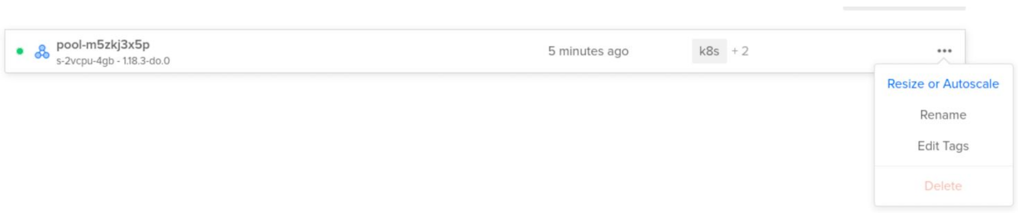


```
oxclo@oxclo:~/cassandra$ kubectl exec -ti cassandra-0 -- nodetool status
Datacenter: DC1-K8Demo
======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.244.0.125  11.04 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
```

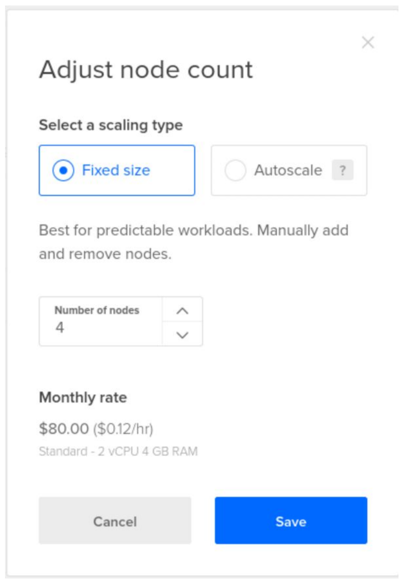
34. Rerun the stress test (Steps 29 and 30).
    Unfortunately, we may not be any faster - because now at least 2 pods are on one node (4 pods, 3 nodes).

```
Results:
Op rate                   :     5,695 op/s  [WRITE: 5,695 op/s]
Partition rate            :     5,695 pk/s  [WRITE: 5,695 pk/s]
Row rate                  :     5,695 row/s [WRITE: 5,695 row/s]
Latency mean              :   164.1 ms [WRITE: 164.1 ms]
Latency median            :    72.7 ms [WRITE: 72.7 ms]
Latency 95th percentile   :   590.3 ms [WRITE: 590.3 ms]
Latency 99th percentile   :   788.0 ms [WRITE: 788.0 ms]
Latency 99.9th percentile :   976.2 ms [WRITE: 976.2 ms]
Latency max               :  1406.1 ms [WRITE: 1,406.1 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      : 00:00:17
```

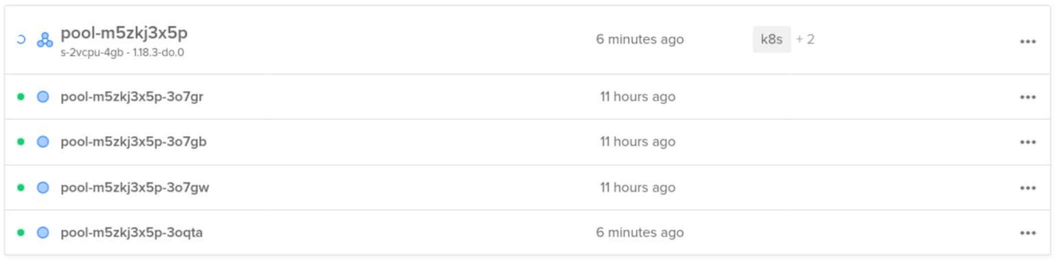
35. Let's fix that:
    Go to the Digital Ocean **Kubernetes**-> **k8s-cass**-> **Nodes**. Click on the Node Pool …



Add an extra node:



36. Wait until the new node is in place:



```
pool-m5zkj3x5p            6 minutes ago    k8s +2
s-2vcpu-4gb - 1.18.3-do.0
pool-m5zkj3x5p-3o7gr      11 hours ago
pool-m5zkj3x5p-3o7gb      11 hours ago
pool-m5zkj3x5p-3o7gw      11 hours ago
pool-m5zkj3x5p-3oqta      6 minutes ago
```

37. Check from kubectl that the new node is active:

```
kubectl get nodes

NAME                    STATUS   ROLES    AGE     VERSION
pool-m5zkj3x5p-3o7gb    Ready    <none>   11h     v1.18.3
pool-m5zkj3x5p-3o7gr    Ready    <none>   11h     v1.18.3
pool-m5zkj3x5p-3o7gw    Ready    <none>   11h     v1.18.3
pool-m5zkj3x5p-3oqta    Ready    <none>   5m57s   v1.18.3
```

38. Use either the Kubernetes Dashboard or k9s or kubectl to see how the pods are assigned to nodes. Unfortunately we are unbalanced, because Kubernetes hasn't had any impetus to move the extra pod to the new node.

39. Rerun the stress test and see what the performance is like

40. We have two options. We could just create a few more Cassandra nodes to more evenly use the nodes. However, there is another option using a cool tool called the Kubernetes Descheduler:

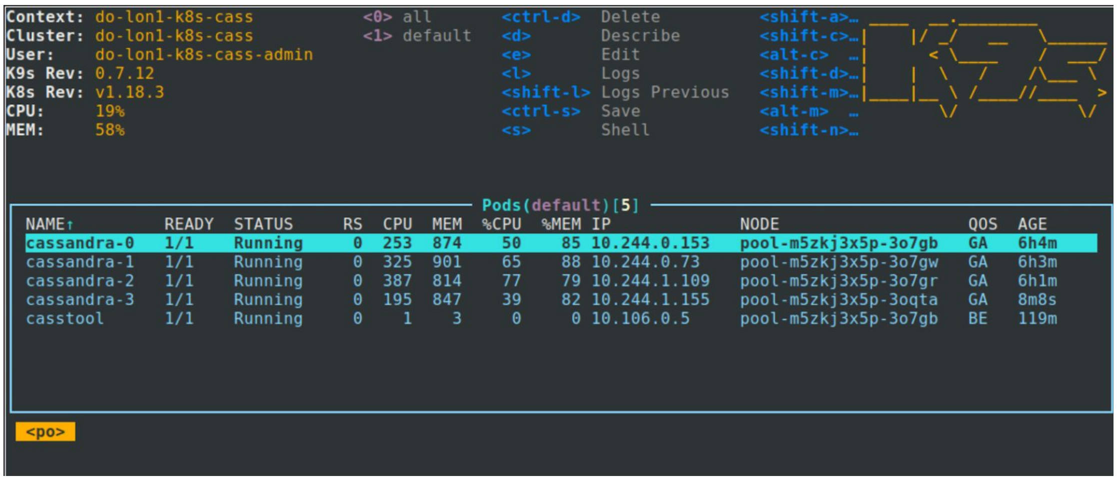
https://github.com/kubernetes-sigs/descheduler

41. In a window do this:

```
git clone
https://github.com/kubernetes-sigs/descheduler.git
cd descheduler
kubectl create -f kubernetes/rbac.yaml
kubectl create -f kubernetes/configmap.yaml
kubectl create -f kubernetes/job.yaml
```

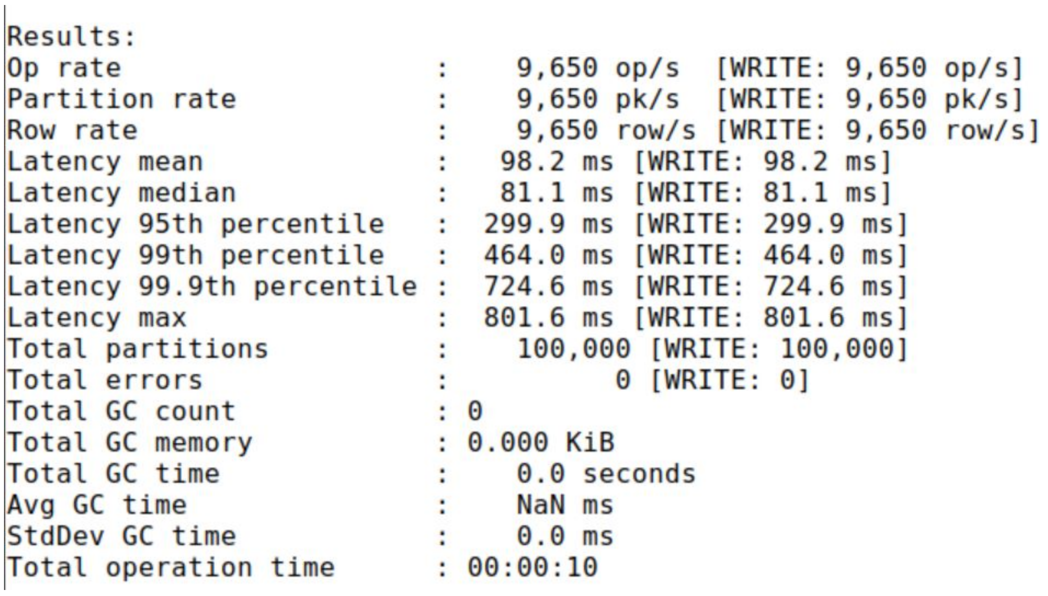
42. Watch what happens using k9s

43. Once the pod has moved, all the cassandra pods should be on different nodes:



```
Context: do-lon1-k8s-cass             <0> all       <ctrl-d>  Delete          <shift-a>…
Cluster: do-lon1-k8s-cass             <1> default   <d>       Describe        <shift-c>…
User:    do-lon1-k8s-cass-admin                     <e>       Edit            <alt-c>  …
K9s Rev: 0.7.12                                     <l>       Logs            <shift-d>…
K8s Rev: v1.18.3                                    <shift-l> Logs Previous   <shift-m>…
CPU:     19%                                        <ctrl-s>  Save            <alt-m>  …
MEM:     58%                                        <s>       Shell           <shift-n>…

                                   Pods(default)[5]
NAME↑         READY  STATUS    RS  CPU  MEM  %CPU  %MEM IP            NODE                    QOS  AGE
cassandra-0   1/1    Running    0  253  874    50    85 10.244.0.153  pool-m5zkj3x5p-3o7gb    GA   6h4m
cassandra-1   1/1    Running    0  325  901    65    88 10.244.0.73   pool-m5zkj3x5p-3o7gw    GA   6h3m
cassandra-2   1/1    Running    0  387  814    77    79 10.244.1.109  pool-m5zkj3x5p-3o7gr    GA   6h1m
cassandra-3   1/1    Running    0  195  847    39    82 10.244.1.155  pool-m5zkj3x5p-3oqta    GA   8m8s
casstool      1/1    Running    0    1    3     0     0 10.106.0.5    pool-m5zkj3x5p-3o7gb    BE   119m

<po>
```

44. Now rerun the stress test one more time:
You can see my results were considerably better:



```
Results:
Op rate                   :    9,650 op/s  [WRITE: 9,650 op/s]
Partition rate            :    9,650 pk/s  [WRITE: 9,650 pk/s]
Row rate                  :    9,650 row/s [WRITE: 9,650 row/s]
Latency mean              :   98.2 ms [WRITE: 98.2 ms]
Latency median            :   81.1 ms [WRITE: 81.1 ms]
Latency 95th percentile   :  299.9 ms [WRITE: 299.9 ms]
Latency 99th percentile   :  464.0 ms [WRITE: 464.0 ms]
Latency 99.9th percentile :  724.6 ms [WRITE: 724.6 ms]
Latency max               :  801.6 ms [WRITE: 801.6 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      : 00:00:10
```

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

46. Let's clean up.

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

```
cd ~/cassandra
kubectl delete -f cassandra-statefulset.yaml
```

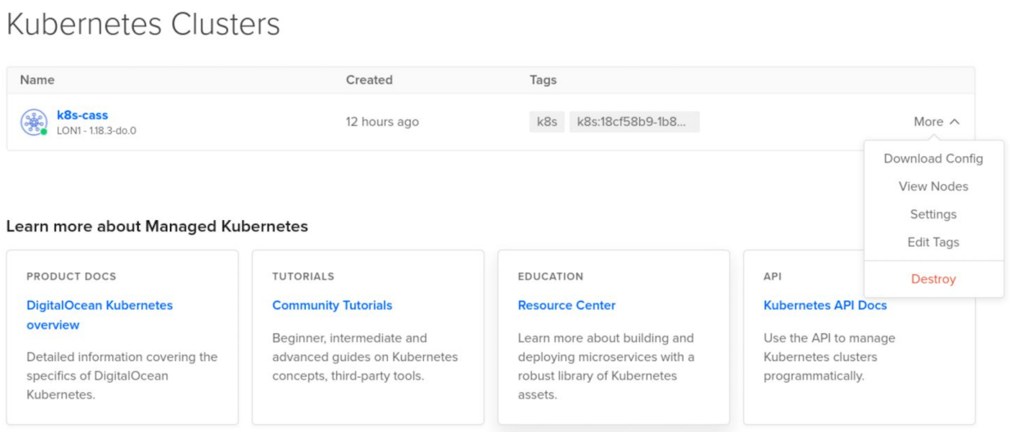
48. Delete our casstool pod:

```
kubectl delete pod/casstool
```

49. Delete the volumes / volume claims:

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

50. Delete the kubernetes cluster (from the DO web panel):



Kubernetes Clusters

Name	Created	Tags	
k8s-cass LON1 - 1.18.3-do.0	12 hours ago	k8s k8s:18cf58b9-1b8...	More

Learn more about Managed Kubernetes

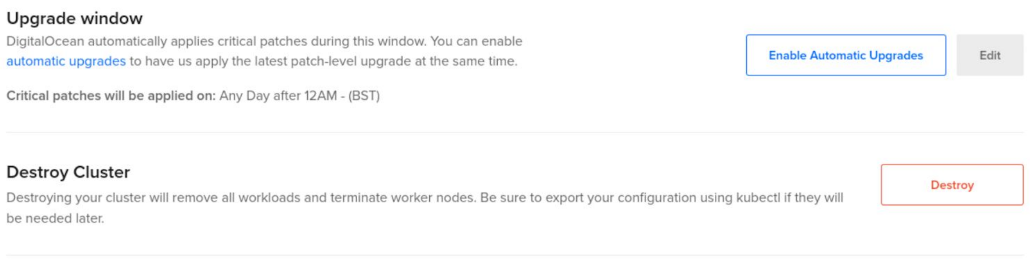
PRODUCT DOCS
DigitalOcean Kubernetes overview
Detailed information covering the specifics of DigitalOcean Kubernetes.

TUTORIALS
Community Tutorials
Beginner, intermediate and advanced guides on Kubernetes concepts, third-party tools.

EDUCATION
Resource Center
Learn more about building and deploying microservices with a robust library of Kubernetes assets.

API
Kubernetes API Docs
Use the API to manage Kubernetes clusters programmatically.

Click **Destroy**



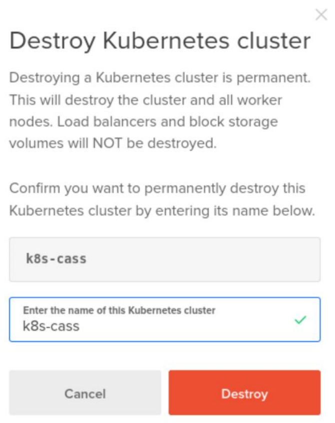
Upgrade window
DigitalOcean automatically applies critical patches during this window. You can enable automatic upgrades to have us apply the latest patch-level upgrade at the same time.

Critical patches will be applied on: Any Day after 12AM - (BST)

Destroy Cluster
Destroying your cluster will remove all workloads and terminate worker nodes. Be sure to export your configuration using kubectl if they will be needed later.

Then **Destroy** again.

Then enter the cluster name (k8s-cass) and actually finally **Destroy**


Destroy Kubernetes cluster

Destroying a Kubernetes cluster is permanent. This will destroy the cluster and all worker nodes. Load balancers and block storage volumes will NOT be destroyed.

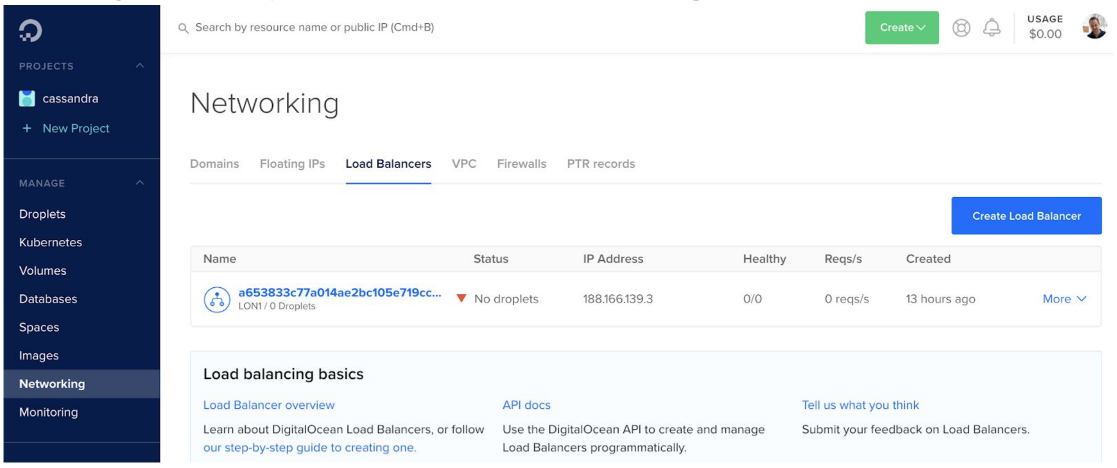
Confirm you want to permanently destroy this Kubernetes cluster by entering its name below.

k8s-cass

Enter the name of this Kubernetes cluster: k8s-cass

51. If you came straight to this lab from the last one, you will also have one last remaining load balancer running.

DigitalOcean will also have created a load-balancer to handle the incoming traffic for your service. Go to **Networking -> Load Balancers**



Networking

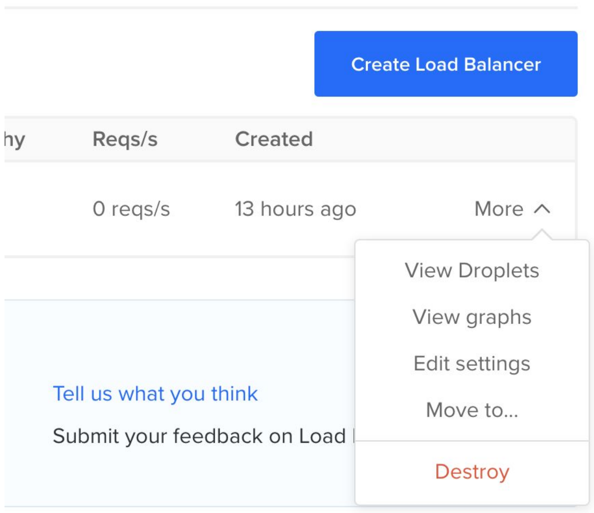
Name	Status	IP Address	Healthy	Reqs/s	Created	
a653833c77a014ae2bc105e719cc... LON1 / 0 Droplets	No droplets	188.166.139.3	0/0	0 reqs/s	13 hours ago	More

Load balancing basics

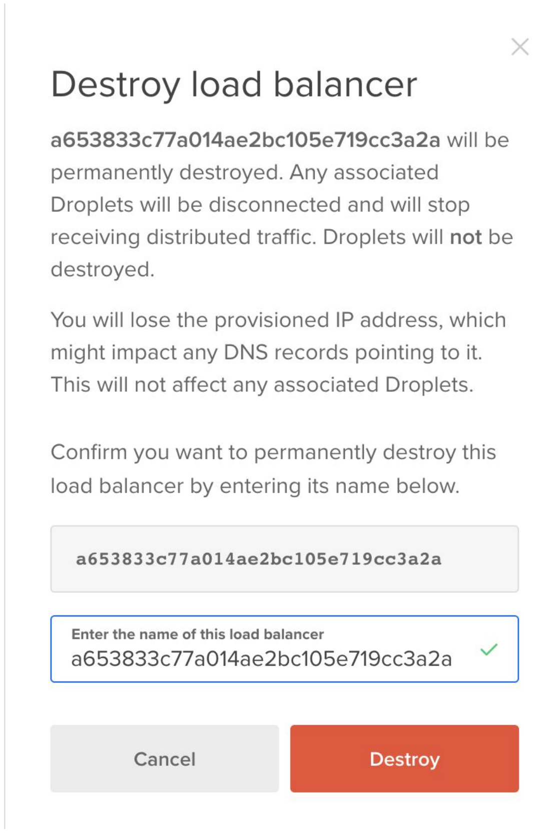
Load Balancer overview — Learn about DigitalOcean Load Balancers, or follow our step-by-step guide to creating one.

API docs — Use the DigitalOcean API to create and manage Load Balancers programmatically.

Tell us what you think — Submit your feedback on Load Balancers.



51. Click on **Destroy** and once again enter the name (copy and paste!)



Destroy load balancer

a653833c77a014ae2bc105e719cc3a2a will be permanently destroyed. Any associated Droplets will be disconnected and will stop receiving distributed traffic. Droplets will **not** be destroyed.

You will lose the provisioned IP address, which might impact any DNS records pointing to it. This will not affect any associated Droplets.

Confirm you want to permanently destroy this load balancer by entering its name below.

`a653833c77a014ae2bc105e719cc3a2a`

Enter the name of this load balancer
a653833c77a014ae2bc105e719cc3a2a ✓

52. This lab is done! Congratulations.