Cloud Native Deployments of Cassandra using Kubernetes

Table of Contents

- Prerequisites
- Cassandra Docker
- Quickstart
- Step 1: Create a Cassandra Headless Service
- Step 2: Use a StatefulSet to create Cassandra Ring
- Step 3: Validate and Modify The Cassandra StatefulSet
- Step 4: Delete Cassandra StatefulSet
- Step 5: Use a Replication Controller to create Cassandra node pods
- Step 6: Scale up the Cassandra cluster
- Step 7: Delete the Replication Controller
- Step 8: Use a DaemonSet instead of a Replication Controller
- Step 9: Resource Cleanup
- Seed Provider Source

The following document describes the development of a *cloud native* Cassandra deployment on Kubernetes. When we say *cloud native,* we mean an application which understands that it is running
within a cluster manager, and uses this cluster management infrastructure to

help implement the application. In particular, in this instance, a custom Cassandra `SeedProvider` is used to enable Cassandra to dynamically discover
new Cassandra nodes as they join the cluster.

This example also uses some of the core components of Kubernetes:

- *Pods*
- *Services*
- *Replication Controllers*
- *Stateful Sets*
- *Daemon Sets*

Prerequisites

This example assumes that you have a Kubernetes version >=1.2 cluster installed and running,
and that you have installed the `kubectl`
command line tool somewhere in your path. Please see the
getting started guides
for installation instructions for your platform.

This example also has a few code and configuration files needed. To avoid
typing these out, you can `git clone` the Kubernetes repository to your local
computer.

Cassandra Docker

The pods use the `gcr.io/google-samples/cassandra:v12` image from Google's container registry.
The docker is based on `debian:jessie` and includes OpenJDK 8. This image
includes a standard Cassandra installation from the Apache Debian repo. Through the use of environment variables you are able to change values that are inserted into the `cassandra.yaml`.

ENV VAR	DEFAULT VALUE
CASSANDRA_CLUSTER_NAME	'Test Cluster'
CASSANDRA_NUM_TOKENS	32
CASSANDRA_RPC_ADDRESS	0.0.0.0

Quickstart

If you want to jump straight to the commands we will run,
here are the steps:

```
#
# StatefulSet
#

# create a service to track all cassandra statefulset nod
es
```

```
kubectl create -f examples/storage/cassandra/cassandra-service.yaml

# create a statefulset
kubectl create -f examples/storage/cassandra/cassandra-statefulset.yaml

# validate the Cassandra cluster. Substitute the name of one of your pods.
kubectl exec -ti cassandra-0 -- nodetool status

# cleanup
grace=$(kubectl get po cassandra-0 --template '{{.spec.terminationGracePeriodSeconds}}') \
  && kubectl delete statefulset,po -l app=cassandra \
  && echo "Sleeping $grace" \
  && sleep $grace \
  && kubectl delete pvc -l app=cassandra

#
# Resource Controller Example
#

# create a replication controller to replicate cassandra nodes
kubectl create -f examples/storage/cassandra/cassandra-controller.yaml
```

```
# validate the Cassandra cluster. Substitute the name of
one of your pods.
kubectl exec -ti cassandra-xxxxx -- nodetool status

# scale up the Cassandra cluster
kubectl scale rc cassandra --replicas=4

# delete the replication controller
kubectl delete rc cassandra

#
# Create a DaemonSet to place a cassandra node on each ku
bernetes node
#

kubectl create -f examples/storage/cassandra/cassandra-da
emonset.yaml --validate=false

# resource cleanup
kubectl delete service -l app=cassandra
kubectl delete daemonset cassandra
```

Step 1: Create a Cassandra Headless Service

A Kubernetes *Service* describes a set of

Pods that perform the same task. In

Kubernetes, the atomic unit of an application is a Pod: one or more containers
that *must* be scheduled onto the same host.

The Service is used for DNS lookups between Cassandra Pods, and Cassandra clients
within the Kubernetes Cluster.

Here is the service description:

<!-- BEGIN MUNGE: EXAMPLE cassandra-service.yaml -->

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

Download example

<!-- END MUNGE: EXAMPLE cassandra-service.yaml -->

Create the service for the StatefulSet:

```
$ kubectl create -f examples/storage/cassandra/cassandra-
service.yaml
```

The following command shows if the service has been created.

```
$ kubectl get svc cassandra
```

The response should be like:

```
NAME        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
cassandra   None         <none>        9042/TCP   45s
```

If an error is returned the service create failed.

Step 2: Use a StatefulSet to create Cassandra Ring

StatefulSets (previously PetSets) are a feature that was upgraded to a Beta component in
Kubernetes 1.5. Deploying stateful distributed applications, like Cassandra, within a clustered
environment can be challenging. We implemented StatefulSet to greatly simplify this
process. Multiple StatefulSet features are used within this example,

but is out of
scope of this documentation. Please refer to the Stateful Set documentation.

The StatefulSet manifest that is included below, creates a Cassandra ring that consists
of three pods.

This example includes using a GCE Storage Class, please update appropriately depending
on the cloud you are working with.

<!-- BEGIN MUNGE: EXAMPLE cassandra-statefulset.yaml -->

```
apiVersion: "apps/v1beta1"
kind: StatefulSet
metadata:
  name: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      containers:
      - name: cassandra
```

```
        image: gcr.io/google-samples/cassandra:v12
        imagePullPolicy: Always
        ports:
        - containerPort: 7000
          name: intra-node
        - containerPort: 7001
          name: tls-intra-node
        - containerPort: 7199
          name: jmx
        - containerPort: 9042
          name: cql
        resources:
          limits:
            cpu: "500m"
            memory: 1Gi
          requests:
           cpu: "500m"
           memory: 1Gi
        securityContext:
          capabilities:
            add:
              - IPC_LOCK
        lifecycle:
          preStop:
            exec:
              command: ["/bin/sh", "-c", "PID=$(pidof jav
a) && kill $PID && while ps -p $PID > /dev/null; do sleep
 1; done"]
```

```
        env:
          - name: MAX_HEAP_SIZE
            value: 512M
          - name: HEAP_NEWSIZE
            value: 100M
          - name: CASSANDRA_SEEDS
            value: "cassandra-0.cassandra.default.svc.clu
ster.local"
          - name: CASSANDRA_CLUSTER_NAME
            value: "K8Demo"
          - name: CASSANDRA_DC
            value: "DC1-K8Demo"
          - name: CASSANDRA_RACK
            value: "Rack1-K8Demo"
          - name: CASSANDRA_AUTO_BOOTSTRAP
            value: "false"
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        readinessProbe:
          exec:
            command:
            - /bin/bash
            - -c
            - /ready-probe.sh
          initialDelaySeconds: 15
          timeoutSeconds: 5
```

```
        # These volume mounts are persistent. They are li
ke inline claims,
        # but not exactly because the names need to match
 exactly one of
        # the stateful pod volumes.
        volumeMounts:
        - name: cassandra-data
          mountPath: /cassandra_data
  # These are converted to volume claims by the controlle
r
  # and mounted at the paths mentioned above.
  # do not use these in production until ssd GCEPersisten
tDisk or other ssd pd
  volumeClaimTemplates:
  - metadata:
      name: cassandra-data
      annotations:
        volume.beta.kubernetes.io/storage-class: fast
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: fast
```

```
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

Download example

<!-- END MUNGE: EXAMPLE cassandra-statefulset.yaml -->

Create the Cassandra StatefulSet as follows:

```
$ kubectl create -f examples/storage/cassandra/cassandra-statefulset.yaml
```

Step 3: Validate and Modify The Cassandra StatefulSet

Deploying this StatefulSet shows off two of the new features that StatefulSets provides.

1. The pod names are known
2. The pods deploy in incremental order

First validate that the StatefulSet has deployed, by running `kubectl` command below.

```
$ kubectl get statefulset cassandra
```

The command should respond like:

```
NAME        DESIRED   CURRENT   AGE
cassandra   3         3         13s
```

Next watch the Cassandra pods deploy, one after another. The StatefulSet resource
deploys pods in a number fashion: 1, 2, 3, etc. If you execute the following
command before the pods deploy you are able to see the ordered creation.

```
$ kubectl get pods -l="app=cassandra"
NAME          READY     STATUS              RESTARTS   AG
E
cassandra-0   1/1       Running             0          1m
cassandra-1   0/1       ContainerCreating   0          8s
```

The above example shows two of the three pods in the Cassandra StatefulSet deployed.
Once all of the pods are deployed the same command will respond with the full
StatefulSet.

```
$ kubectl get pods -l="app=cassandra"
NAME          READY     STATUS    RESTARTS   AGE
```

```
cassandra-0   1/1       Running   0          10m
cassandra-1   1/1       Running   0          9m
cassandra-2   1/1       Running   0          8m
```

Running the Cassandra utility `nodetool` will display the status of the ring.

```
$ kubectl exec cassandra-0 -- nodetool status
Datacenter: DC1-K8Demo
======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address   Load       Tokens       Owns (effective)  H
ost ID                               Rack
UN  10.4.2.4  65.26 KiB  32           63.7%             a
9d27f81-6783-461d-8583-87de2589133e  Rack1-K8Demo
UN  10.4.0.4  102.04 KiB  32           66.7%
5559a58c-8b03-47ad-bc32-c621708dc2e4  Rack1-K8Demo
UN  10.4.1.4  83.06 KiB  32           69.6%             9
dce943c-581d-4c0e-9543-f519969cc805  Rack1-K8Demo
```

You can also run `cqlsh` to describe the keyspaces in the cluster.

```
$ kubectl exec cassandra-0 -- cqlsh -e 'desc keyspaces'

system_traces  system_schema  system_auth  system  system
_distributed
```

In order to increase or decrease the size of the Cassandra StatefulSet, you must use
`kubectl edit`. You can find more information about the edit command in the documentation.

Use the following command to edit the StatefulSet.

```
$ kubectl edit statefulset cassandra
```

This will create an editor in your terminal. The line you are looking to change is
`replicas`. The example does on contain the entire contents of the terminal window, and
the last line of the example below is the replicas line that you want to change.

```
# Please edit the object below. Lines beginning with a '#
' will be ignored,
# and an empty file will abort the edit. If an error occu
rs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  creationTimestamp: 2016-08-13T18:40:58Z
```

```
  generation: 1
  labels:
    app: cassandra
  name: cassandra
  namespace: default
  resourceVersion: "323"
  selfLink: /apis/apps/v1beta1/namespaces/default/statefu
lsets/cassandra
  uid: 7a219483-6185-11e6-a910-42010a8a0fc0
spec:
  replicas: 3
```

Modify the manifest to the following, and save the manifest.

```
spec:
  replicas: 4
```

The StatefulSet will now contain four pods.

```
$ kubectl get statefulset cassandra
```

The command should respond like:

```
NAME        DESIRED   CURRENT   AGE
cassandra   4         4         36m
```

For the Kubernetes 1.5 release, the beta StatefulSet resource does not have `kubectl scale` functionality, like a Deployment, ReplicaSet, Replication Controller, or Job.

Step 4: Delete Cassandra StatefulSet

Deleting and/or scaling a StatefulSet down will not delete the volumes associated with the StatefulSet. This is done to ensure safety first, your data is more valuable than an auto purge of all related StatefulSet resources. Deleting the Persistent Volume Claims may result in a deletion of the associated volumes, depending on the storage class and reclaim policy. You should never assume ability to access a volume after claim deletion.

Use the following commands to delete the StatefulSet.

```
$ grace=$(kubectl get po cassandra-0 --template '{{.spec.
terminationGracePeriodSeconds}}') \
  && kubectl delete statefulset -l app=cassandra \
  && echo "Sleeping $grace" \
  && sleep $grace \
  && kubectl delete pvc -l app=cassandra
```

Step 5: Use a Replication

Controller to create Cassandra node pods

A Kubernetes
Replication Controller
is responsible for replicating sets of identical pods. Like a
Service, it has a selector query which identifies the members of its set.
Unlike a Service, it also has a desired number of replicas, and it will create
or delete Pods to ensure that the number of Pods matches up with its desired state.

The Replication Controller, in conjunction with the Service we just defined,
will let us easily build a replicated, scalable Cassandra cluster.

Let’s create a replication controller with two initial replicas.

<!-- BEGIN MUNGE: EXAMPLE cassandra-controller.yaml -->

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: cassandra
  # The labels will be applied automatically
  # from the labels in the pod template, if not set
  # labels:
```

```
    # app: cassandra
spec:
  replicas: 2
  # The selector will be applied automatically
  # from the labels in the pod template, if not set.
  # selector:
      # app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      containers:
        - command:
            - /run.sh
          resources:
            limits:
              cpu: 0.5
          env:
            - name: MAX_HEAP_SIZE
              value: 512M
            - name: HEAP_NEWSIZE
              value: 100M
            - name: CASSANDRA_SEED_PROVIDER
              value: "io.k8s.cassandra.KubernetesSeedProv
ider"
            - name: POD_NAMESPACE
              valueFrom:
```

```
              fieldRef:
                fieldPath: metadata.namespace
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        image: gcr.io/google-samples/cassandra:v12
        name: cassandra
        ports:
          - containerPort: 7000
            name: intra-node
          - containerPort: 7001
            name: tls-intra-node
          - containerPort: 7199
            name: jmx
          - containerPort: 9042
            name: cql
        volumeMounts:
          - mountPath: /cassandra_data
            name: data
      volumes:
        - name: data
          emptyDir: {}
```

Download example

<!-- END MUNGE: EXAMPLE cassandra-controller.yaml -->

There are a few things to note in this description.

The `selector` attribute contains the controller’s selector query. It can be
explicitly specified, or applied automatically from the labels in the pod template if not set, as is done here.

The pod template’s label, `app:cassandra`, matches the Service selector
from Step 1. This is how pods created by this replication controller are picked up
by the Service."

The `replicas` attribute specifies the desired number of replicas, in this
case 2 initially. We’ll scale up to more shortly.

Create the Replication Controller:

```
$ kubectl create -f examples/storage/cassandra/cassandra-controller.yaml
```

You can list the new controller:

```
$ kubectl get rc -o wide
NAME        DESIRED   CURRENT   AGE       CONTAINER(S)
IMAGE(S)                                SELECTOR
cassandra   2         2         11s       cassandra
gcr.io/google-samples/cassandra:v12   app=cassandra
```

Now if you list the pods in your cluster, and filter to the label `app=cassandra`, you should see two Cassandra pods. (The `wide` argument lets
you see which Kubernetes nodes the pods were scheduled onto.)

```
$ kubectl get pods -l="app=cassandra" -o wide
NAME              READY     STATUS    RESTARTS   AGE
  NODE
cassandra-21qyy   1/1       Running   0          1m
  kubernetes-minion-b286
cassandra-q6sz7   1/1       Running   0          1m
  kubernetes-minion-9ye5
```

Because these pods have the label `app=cassandra`, they map to the service we
defined in Step 1.

You can check that the Pods are visible to the Service using the

following service endpoints query:

```
$ kubectl get endpoints cassandra -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  creationTimestamp: 2015-06-21T22:34:12Z
  labels:
    app: cassandra
  name: cassandra
  namespace: default
  resourceVersion: "944373"
  selfLink: /api/v1/namespaces/default/endpoints/cassandr
a
  uid: a3d6c25f-1865-11e5-a34e-42010af01bcc
subsets:
- addresses:
  - ip: 10.244.3.15
    targetRef:
      kind: Pod
      name: cassandra
      namespace: default
      resourceVersion: "944372"
      uid: 9ef9895d-1865-11e5-a34e-42010af01bcc
  ports:
  - port: 9042
    protocol: TCP
```

To show that the `SeedProvider` logic is working as intended, you can use the
`nodetool` command to examine the status of the Cassandra cluster. To do this,
use the `kubectl exec` command, which lets you run `nodetool` in one of your
Cassandra pods. Again, substitute `cassandra-xxxxx` with the actual name of one
of your pods.

```
$ kubectl exec -ti cassandra-xxxxx -- nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host
 ID                               Rack
UN  10.244.0.5  74.09 KB   256     100.0%            86fe
da0f-f070-4a5b-bda1-2eeb0ad08b77  rack1
UN  10.244.3.3  51.28 KB   256     100.0%            dafe
3154-1d67-42e1-ac1d-78e7e80dce2b  rack1
```

Step 6: Scale up the Cassandra cluster

Now let’s scale our Cassandra cluster to 4 pods. We do this by telling the
Replication Controller that we now want 4 replicas.

```
$ kubectl scale rc cassandra --replicas=4
```

You can see the new pods listed:

```
$ kubectl get pods -l="app=cassandra" -o wide
NAME              READY     STATUS    RESTARTS   AGE
  NODE
cassandra-21qyy   1/1       Running   0          6m
  kubernetes-minion-b286
cassandra-81m2l   1/1       Running   0          47s
  kubernetes-minion-b286
cassandra-8qoyp   1/1       Running   0          47s
  kubernetes-minion-9ye5
cassandra-q6sz7   1/1       Running   0          6m
  kubernetes-minion-9ye5
```

In a few moments, you can examine the Cassandra cluster status again, and see
that the new pods have been detected by the custom `SeedProvider`:

```
$ kubectl exec -ti cassandra-xxxxx -- nodetool status
Datacenter: datacenter1
======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host
 ID                               Rack
UN  10.244.0.6  51.67 KB   256     48.9%             d07b
23a5-56a1-4b0b-952d-68ab95869163  rack1
UN  10.244.1.5  84.71 KB   256     50.7%             e060
df1f-faa2-470c-923d-ca049b0f3f38  rack1
UN  10.244.1.6  84.71 KB   256     47.0%             83ca
1580-4f3c-4ec5-9b38-75036b7a297f  rack1
UN  10.244.0.5  68.2 KB    256     53.4%             72ca
27e2-c72c-402a-9313-1e4b61c2f839  rack1
```

Step 7: Delete the Replication Controller

Before you start Step 5, **delete the replication controller** you

created above:

```
$ kubectl delete rc cassandra
```

Step 8: Use a DaemonSet instead of a Replication Controller

In Kubernetes, a *Daemon Set* can distribute pods
onto Kubernetes nodes, one-to-one. Like a *ReplicationController*, it has a
selector query which identifies the members of its set. Unlike a *ReplicationController*, it has a node selector to limit which nodes are scheduled with the templated pods, and replicates not based on a set target
number of pods, but rather assigns a single pod to each targeted node.

An example use case: when deploying to the cloud, the expectation is that
instances are ephemeral and might die at any time. Cassandra is built to
replicate data across the cluster to facilitate data redundancy, so that in the
case that an instance dies, the data stored on the instance does not,

and the
cluster can react by re-replicating the data to other running nodes.

`DaemonSet` is designed to place a single pod on each node in the Kubernetes
cluster. That will give us data redundancy. Let’s create a
DaemonSet to start our storage cluster:

<!-- BEGIN MUNGE: EXAMPLE cassandra-daemonset.yaml -->

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  labels:
    name: cassandra
  name: cassandra
spec:
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      # Filter to specific nodes:
      # nodeSelector:
      #  app: cassandra
      containers:
        - command:
            - /run.sh
```

```
            env:
              - name: MAX_HEAP_SIZE
                value: 512M
              - name: HEAP_NEWSIZE
                value: 100M
              - name: CASSANDRA_SEED_PROVIDER
                value: "io.k8s.cassandra.KubernetesSeedProv
ider"
              - name: POD_NAMESPACE
                valueFrom:
                  fieldRef:
                    fieldPath: metadata.namespace
              - name: POD_IP
                valueFrom:
                  fieldRef:
                    fieldPath: status.podIP
            image: gcr.io/google-samples/cassandra:v12
            name: cassandra
            ports:
              - containerPort: 7000
                name: intra-node
              - containerPort: 7001
                name: tls-intra-node
              - containerPort: 7199
                name: jmx
              - containerPort: 9042
                name: cql
                # If you need it it is going away in C* 4.0
```

```
            #- containerPort: 9160
            #  name: thrift
          resources:
            requests:
              cpu: 0.5
          volumeMounts:
            - mountPath: /cassandra_data
              name: data
      volumes:
        - name: data
          emptyDir: {}
```

Download example

<!-- END MUNGE: EXAMPLE cassandra-daemonset.yaml -->

Most of this DaemonSet definition is identical to the ReplicationController
definition above; it simply gives the daemon set a recipe to use when it creates
new Cassandra pods, and targets all Cassandra nodes in the cluster.

Differentiating aspects are the `nodeSelector` attribute, which allows the
DaemonSet to target a specific subset of nodes (you can label nodes just like
other resources), and the lack of a `replicas` attribute due to the 1-to-1 node-

pod relationship.

Create this DaemonSet:

```
$ kubectl create -f examples/storage/cassandra/cassandra-
daemonset.yaml
```

You may need to disable config file validation, like so:

```
$ kubectl create -f examples/storage/cassandra/cassandra-
daemonset.yaml --validate=false
```

You can see the DaemonSet running:

```
$ kubectl get daemonset
NAME        DESIRED   CURRENT   NODE-SELECTOR
cassandra   3         3         <none>
```

Now, if you list the pods in your cluster, and filter to the label `app=cassandra`, you should see one (and only one) new cassandra pod

for each

node in your network.

```
$ kubectl get pods -l="app=cassandra" -o wide
NAME                READY     STATUS    RESTARTS   AGE
  NODE
cassandra-ico4r   1/1       Running   0          4s
  kubernetes-minion-rpo1
cassandra-kitfh   1/1       Running   0          1s
  kubernetes-minion-9ye5
cassandra-tzw89   1/1       Running   0          2s
  kubernetes-minion-b286
```

To prove that this all worked as intended, you can again use the `nodetool` command to examine the status of the cluster. To do this, use the `kubectl exec` command to run `nodetool` in one of your newly-launched cassandra pods.

```
$ kubectl exec -ti cassandra-xxxxx -- nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
```

```
--  Address     Load       Tokens  Owns (effective)  Host
 ID                                   Rack
UN  10.244.0.5  74.09 KB   256     100.0%            86fe
da0f-f070-4a5b-bda1-2eeb0ad08b77  rack1
UN  10.244.4.2  32.45 KB   256     100.0%            0b1b
e71a-6ffb-4895-ac3e-b9791299c141  rack1
UN  10.244.3.3  51.28 KB   256     100.0%            dafe
3154-1d67-42e1-ac1d-78e7e80dce2b  rack1
```

Note: This example had you delete the cassandra Replication Controller before
you created the DaemonSet. This is because – to keep this example simple – the
RC and the DaemonSet are using the same `app=cassandra` label (so that their pods map to the
service we created, and so that the SeedProvider can identify them).

If we didn't delete the RC first, the two resources would conflict with respect to how many pods they wanted to have running. If we wanted, we could support running
both together by using additional labels and selectors.

Step 9: Resource Cleanup

When you are ready to take down your resources, do the following:

```
$ kubectl delete service -l app=cassandra
$ kubectl delete daemonset cassandra
```

Custom Seed Provider

A custom `SeedProvider`
is included for running Cassandra on top of Kubernetes. Only when you deploy Cassandra
via a replication control or a daemonset, you will need to use the custom seed provider.
In Cassandra, a `SeedProvider` bootstraps the gossip protocol that Cassandra uses to find other
Cassandra nodes. Seed addresses are hosts deemed as contact points. Cassandra
instances use the seed list to find each other and learn the topology of the
ring. The `KubernetesSeedProvider`
discovers Cassandra seeds IP addresses via the Kubernetes API, those Cassandra
instances are defined within the Cassandra Service.

Refer to the custom seed provider README for further
`KubernetesSeedProvider` configurations. For this example you should not need
to customize the Seed Provider configurations.

See the image directory of this example for specifics on how the container docker image was built and what it contains.

You may also note that we are setting some Cassandra parameters (`MAX_HEAP_SIZE` and `HEAP_NEWSIZE`), and adding information about the namespace.
We also tell Kubernetes that the container exposes both the `CQL` and `Thrift` API ports. Finally, we tell the cluster manager that we need 0.1 cpu (0.1 core).

<!-- BEGIN MUNGE: GENERATED_ANALYTICS -->

<!-- END MUNGE: GENERATED_ANALYTICS -->