Deploying Container-Native Storage

3

In this section you will set up Container-Native Storage (CNS) in your environment. You will use this in "Container-Native Storage Management" lab to dynamically provision storage for containerized applications. It is provided by GlusterFS running in containers. GlusterFS in turn is backed by local storage devices available to the OpenShift nodes. This lab requires that you completed OpenShift installation in the previous module.

Preparation for CNS

All of the following tasks are carried out as the `cloud-user` user on the master node *master.647073518612.aws.testdrive.openshift.com*. Before the deployment a couple of simple prerequisites need to be fulfilled.

Check the Prerequisites

First, on the master, verify that the CNS deployment tool is installed. We will also use Ansible to apply the firewall configuration across all 3 nodes. Though not needed for CNS per se, in this lab it will help us simplify an otherwise tedious manual configuration step.

```
yum list installed cns-deploy
```

The command should return the following:

```
Loaded plugins: product-id, search-disabled-repos, subscription-ma
This system is not registered with an entitlement server. You can
Installed Packages                cns-deploy.x86_64
```

Configure OpenShift Node firewall with Ansible

Just as Ansible was used to install and configure OpenShift, you will configure Ansible to take care of certain CNS prerequisites and installation steps.

There is a host group in `/etc/ansible/hosts` for the three nodes that will initially run CNS. There are three more hosts, currently commented out, for future use.

You should be able to "ping" the first 3 OpenShift App nodes using Ansible:

```
ansible cns -m ping
```

This should result in the following:

```
node01.647073518612.aws.testdrive.openshift.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node02.647073518612.aws.testdrive.openshift.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node03.647073518612.aws.testdrive.openshift.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

This verifies the systems can be reached and configure via Ansible. For the firewall configuration in this lab is a preapred file in the `support` folder called `configure-firewall.yaml`. This is a short Ansible playbook that will configure the firewall to allow GlusterFS-related traffic (namely on ports 24007,24008, 2222 and 49152-49664 - all TCP), required for CNS. Run it with the following command:

```
ansible-playbook /opt/lab/support/configure-firewall.yaml
```

Feel free to examine the playbook file if you are curious. When the Ansible play completes successfully, your output should look like the following:

```
PLAY [nodes] *****************************************************

TASK [setup] *****************************************************
ok: [node01.647073518612.aws.testdrive.openshift.com]
ok: [node02.647073518612.aws.testdrive.openshift.com]
ok: [node03.647073518612.aws.testdrive.openshift.com]

TASK [insert iptables rules required for GlusterFS] **************
changed: [node01.647073518612.aws.testdrive.openshift.com]
changed: [node02.647073518612.aws.testdrive.openshift.com]
```

```
changed: [node03.647073518612.aws.testdrive.openshift.com]

TASK [reload iptables] *******************************************
changed: [node01.647073518612.aws.testdrive.openshift.com]
changed: [node02.647073518612.aws.testdrive.openshift.com]
changed: [node03.647073518612.aws.testdrive.openshift.com]

PLAY RECAP *******************************************************
node01.647073518612.aws.testdrive.openshift.com          : ok=3
node02.647073518612.aws.testdrive.openshift.com          : ok=3
node03.647073518612.aws.testdrive.openshift.com          : ok=3
```

Prepare OpenShift for CNS

It makes sense to create a dedicated **Project** to hold the **Pods** that CNS will use to provide storage inside the OpenShift environment. First, log-in as the cluster administrator:

```
oc login -u system:admin
```

Create a **Project** called `container-native-storage`:

```
oc new-project container-native-storage
```

In CNS, the **Pods** running GlusterFS need access to the physical block devices on the host. OpenShift's security model prevents that behaviour by default. In order to provide the permissions required, you will need to enable a privileged operation mode for the **Pods**. Execute the following command, which will allow the `default` **ServiceAccount** to use the `privileged` **SecurityContextConstraint**:

```
oc adm policy add-scc-to-user privileged -z default
```

A deeper discussion of Security Context Constraints is outside of the scope of these lab modules, but the authorization documentation provides a deeper dive into these concepts.

Describe Container-Native Storage Topology

CNS will virtualize the locally attached block storage on the OpenShift nodes using GlusterFS. In order to deploy CNS, though, you will need to supply the CNS installer with information about where to find these nodes and what network and which block devices to use. This is done using a JSON file that describes the topology of the OpenShift environment.

There has already been a JSON file defined for you:

/opt/lab/support/topology.json

```
    {
        "clusters": [
            {
                "nodes": [
                    {
```

```
        "node": {
            "hostnames": {
                "manage": [
                    "node01.internal.aws.testd
                ],
                "storage": [
                    "10.0.1.234" ❷
                ]
            },
            "zone": 1 ❸
        },
        "devices": [
            "/dev/xvdd" ❹
        ]
    },
    {
        "node": {
            "hostnames": {
                "manage": [
                    "node02.internal.aws.testd
                ],
                "storage": [
                    "10.0.3.112" ❷
                ]
            },
            "zone": 2 ❸
        },
        "devices": [
            "/dev/xvdd" ❹
        ]
    },
    {
        "node": {
            "hostnames": {
                "manage": [
                    "node03.internal.aws.testd
```

```
                                ],
                                "storage": [
                                    "10.0.4.184" ❷
                                ]
                            },
                            "zone": 3 ❸
                        },
                        "devices": [
                            "/dev/xvdd" ❹
                        ]
                    }
                ]
            }
        ]
    }
```

❶ heketi uses this FQDN to identify the node in OpenShift (e.g. oc get nodes)

❷ the IP address of the node used for GlusterFS traffic

❸ The failure domain of this node

❹ The host’s storage device(s) to for GlusterFS

> The topology references the nodes by their hostnames as they are known to OpenShift.

This file contains an additional property called `zone` per node. This identifies the failure domain this host resides in. In CNS data is always replicated 3 times. By exposing information about the failure domains we can make sure that two copies are never stored on nodes in the same failure domain. The `zone` definitions are simply arbitrary, but unique integer values.

Install Container-Native Storage

You are now ready to deploy CNS. Alongside CNS **Pods**, the API front-end known as **heketi** is deployed. To protect this API from unauthorized access we will define passwords for the `admin` and `user` role in heketi like below.

Heketi Role	Password
admin	myS3cr3tpassw0rd
user	mys3rs3cr3tpassw0rd

Table 1. CNS passwords

Run the CNS deployer

Next start the deployment routine with the following command:

```
cns-deploy -n container-native-storage -g /opt/lab/support/topolog
```

Answer the interactive prompts with **Y**.

The deployment will take several minutes to complete. Especially the step "Waiting for GlusterFS pods to start" might take some time. On the command line the output should look like this:

```
Welcome to the deployment tool for GlusterFS on Kubernetes and Ope

Before getting started, this script has some requirements of the e
environment and of the container platform that you should verify.

The client machine that will run this script must have:
 * Administrative access to an existing Kubernetes or OpenShift cl
 * Access to a python interpreter 'python'
 * Access to the heketi client 'heketi-cli'

Each of the nodes that will host GlusterFS must also have appropri
rules for the required GlusterFS ports:
 * 2222  - sshd (if running GlusterFS in a pod)
 * 24007 - GlusterFS Daemon
 * 24008 - GlusterFS Management
 * 49152 to 49251 - Each brick for every volume on the host requir
   port. For every new brick, one new port will be used starting a
   recommend a default range of 49152-49251 on each host, though y
   this to fit your needs.

In addition, for an OpenShift deployment you must:
 * Have 'cluster_admin' role on the administrative account doing t
 * Add the 'default' and 'router' Service Accounts to the 'privile
 * Have a router deployed that is configured to allow apps to acce
   running in the cluster

Do you wish to proceed with deployment?

[Y]es, [N]o? [Default: Y]: ❶
Using OpenShift CLI.
```

```
NAME                       STATUS    AGE
container-native-storage   Active    28m
Using namespace "container-native-storage".
Checking that heketi pod is not running ... OK
template "deploy-heketi" created
serviceaccount "heketi-service-account" created
template "heketi" created
template "glusterfs" created
role "edit" added: "system:serviceaccount:container-native-storage
node "node01.internal.aws.testdrive.openshift.com" labeled 2
node "node02.internal.aws.testdrive.openshift.com" labeled 2
node "node03.internal.aws.testdrive.openshift.com" labeled 2
daemonset "glusterfs" created
Waiting for GlusterFS pods to start ... OK 3
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ... OK
Creating cluster ... ID: 307f708621f4e0c9eda962b713272e81
Creating node node01.internal.aws.testdrive.openshift.com ... ID:
Adding device /dev/xvdd ... OK 5
Creating node node02.internal.aws.testdrive.openshift.com ... ID:
Adding device /dev/xvdd ... OK 5
Creating node node03.internal.aws.testdrive.openshift.com ... ID:
Adding device /dev/xvdd ... OK 5
heketi topology loaded.
Saving heketi-storage.json
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-599rc" deleted
```

```
secret "heketi-storage-secret" deleted
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created ❻
Waiting for heketi pod to start ... OK
heketi is now running.
Ready to create and provide GlusterFS volumes.
```

❶ Enter **Y** and press Enter.

❷ OpenShift nodes are labeled. Label is referred to in a DaemonSet.

❸ GlusterFS daemonset is started. DaemonSet means: start exactly **one** pod per node.

❹ All nodes will be referenced in heketi’s database by a UUID.

❺ Node block devices are formatted as physical LVM volumes for later use by GlusterFS.

❻ heketi is deployed in a pod as well.

Some of the output contains auto-generated data, so references to UUIDs and pod names might be slightly different for you.

Verifying and Exploring CNS

Now that CNS is installed, take some time to explore it and verify that all of the components are in place.

Look at the Pods

While still in the `container-native-storage` project on the CLI list all running **Pods**:

```
oc get pods -o wide
```

You will see something like:

```
NAME                READY     STATUS    RESTARTS   AGE       IP
glusterfs-37vn8     1/1       Running   0          3m       10.0.1.2
glusterfs-cq68l     1/1       Running   0          3m       10.0.3.1
glusterfs-m9fvl     1/1       Running   0          3m       10.0.4.1
heketi-1-cd032      1/1       Running   0          1m       10.0.1.7
2
```

1. CNS **Pods**, with each of the designated nodes running exactly one.
2. heketi API frontend pod

The exact **Pod** names will be different in your environment, since they are auto-generated. Also the heketi **Pod** might run on any node.

The CNS **Pods** use the host's network and block devices to run the software-defined storage system. See schematic below for a visualization.

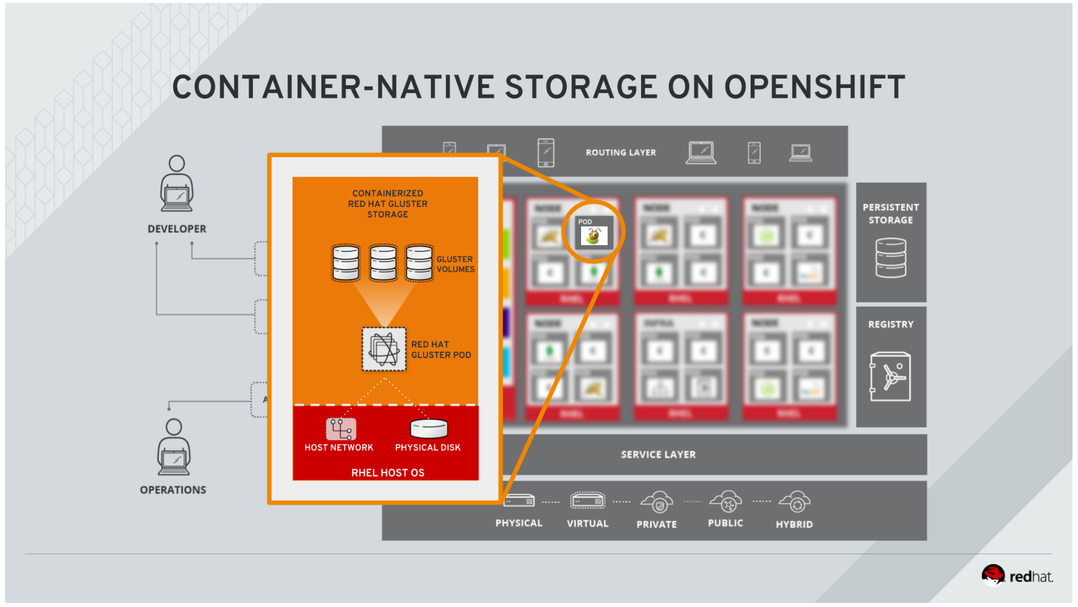

Figure 1. GlusterFS pods in CNS in detail.

heketi is a component that will exposes an API into the storage system for OpenShift. This allows OpenShift to dynamically allocate storage from CNS in a programmatic fashion. See below for a visualization. Note that for simplicity, in our example heketi runs on the OpenShift application nodes, not on the infrastructure node.

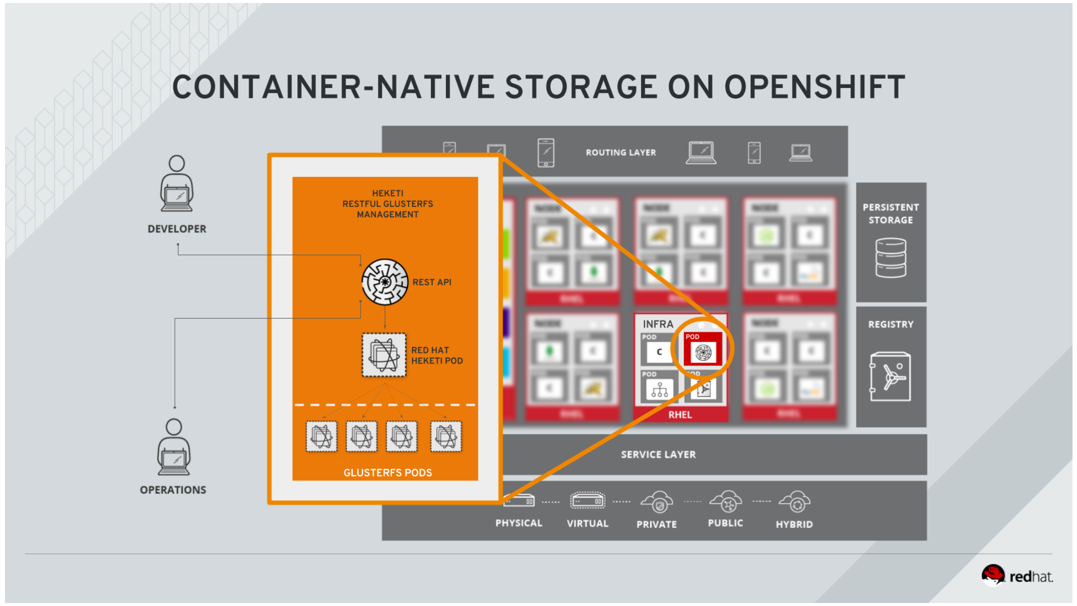

Figure 2. heketi pod running in CNS

Examine heketi

To expose heketi's API a **Service** named *heketi* has been created:

```
oc get service/heketi
```

You will see something like:

```
NAME      CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
heketi    172.30.5.231   <none>        8080/TCP   31m
```

In order to be able to use heketi outside of the OpenShift cluster (for monitoring/data gathering, or other automation), a **Route** has been created. Verify this is the case by running:

```
oc get route/heketi
```

This should output the details, as well as the public URL of the `Route` for the heketi API:

```
NAME      HOST/PORT
heketi    heketi-container-native-storage.apps.647073518612.aws.te
```

You may verify the API is responding and heketi is alive with a trivial health check:

```
curl http://heketi-container-native-storage.apps.647073518612.aws.
```

This should return (likely without a line break)

```
Hello from Heketi
```

That’s it. You have successfully provisioned Container-Native Storage on OpenShift. CNS provides the basis to provide persistent storage to applications, as demonstrated in the "Container-Native Storage Management" module.

CNS is available wherever OpenShift is deployed with no external dependencies.

Exploring Container-Native Storage

Outside of the tight integration with OpenShift you can also work directly with the administrative interface of heketi. This done via the command line client. First, set the following environment variables to configure the client:

```
export HEKETI_CLI_SERVER=http://heketi-container-native-storage.ap
export HEKETI_CLI_USER=admin
export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

You’ll notice how we use the route that OpenShift created for the heketi **Service** to direct the client to the REST API. With username / password from the `cns-deploy` command also set, you can query heketi about the clusters it’s managing:

```
heketi-cli cluster list
```

heketi will list all known clusters with internal UUIDs:

```
Clusters:
ec7a9c8be8327a54839236791bf7ba24 (1)
```

(1) This is the internal UUID of the CNS cluster

The cluster UUID will be different for you since it’s automatically generated.

This tells you that as part of running `cns-deploy` not only the heketi and CNS **Pods** were started, but the 3 CNS **Pods** have also formed a cluster which is now ready to serve replicated, software-defined storage.

Please note the UUID for later reference in the next chapter.

To get more detailed information about the topology of your CNS cluster (i.e. nodes, devices and volumes heketi has discovered) run the following command (output abbreviated):

```
heketi-cli topology info
```

You will get a lengthy output that describes the GlusterFS cluster topology as it is known by heketi:

```
Cluster Id: ec7a9c8be8327a54839236791bf7ba24

    Volumes

        Name: heketidbstorage (1)
        Size: 2
        Id: 272c8d37828c62c4002a19027abd2feb
        Cluster Id: ec7a9c8be8327a54839236791bf7ba24
        Mount: 10.0.1.234:heketidbstorage
        Mount Options: backup-volfile-servers=10.0.3.112,10.0.3.11
        Durability Type: replicate
        Replica: 3
        Snapshot: Disabled

    Nodes:

        Node Id: 099b016da11a623bef37de9b85aaa2d7
        State: online
        Cluster Id: ec7a9c8be8327a54839236791bf7ba24
        Zone: 3
        Management Hostname: node03.internal.aws.testdrive.openshi
        Storage Hostname: node03.internal.aws.testdrive.openshift.
        Devices:
                Id:e64fac664861c14bd75e3116f805b8fc   Name:/dev/xv
                        Bricks:
                            [...]

        Node Id: 43336d05323e6003be6740dbb7477bd6
        State: online
        Cluster Id: ec7a9c8be8327a54839236791bf7ba24
        Zone: 1
        Management Hostname: node01.internal.aws.testdrive.openshi
        Storage Hostname: 10.0.1.234
        Devices:
                Id:11a148d8065f6a6220f89c2912d00d13   Name:/dev/xv
                        Bricks:
                            [...]

        Node Id: 6c738028f642e37b2368eca88f8c626c
        State: online
        Cluster Id: ec7a9c8be8327a54839236791bf7ba24
        Zone: 2
        Management Hostname: node02.internal.aws.testdrive.openshi
        Storage Hostname: 10.0.3.112
        Devices:
                Id:cf7c0dfb258f07be25ac9cd4c4d2e6ae   Name:/dev/xv
                        Bricks:
                            [...]
```

❶ An internal GlusterFS volume that is automatically generated by the setup routine to hold the heketi database.

Dynamic Storage Provisioning

OpenShift supports dynamic storage provisioning for storage providers that have APIs and where a kubernetes plug-in exists. This is a fairly simple framework in which only 3 components exists: the storage provider, the storage volume and the request for a storage volume.

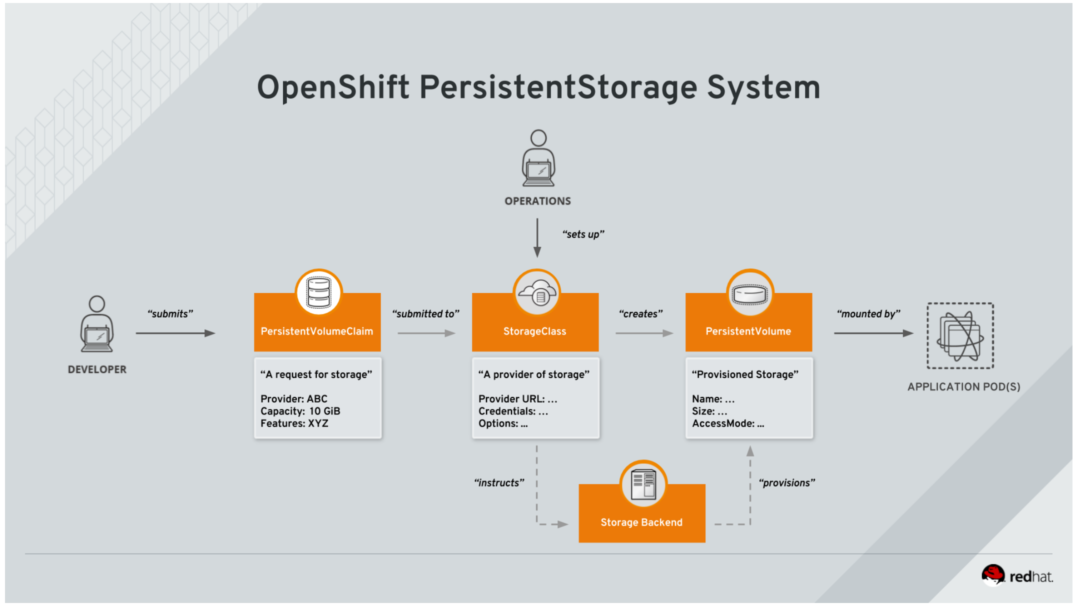

Figure 3. OpenShift Storage Lifecycle

OpenShift knows non-ephemeral storage as "persistent" volumes. This is storage that is decoupled from **Pod** lifecycles. Users can request such storage by submitting a **PersistentVolumeClaim** to the system, which carries aspects like desired capacity or access mode (shared, single, read-only). A storage provider in the system is represented by a **StorageClass** and is referenced in the claim. Upon receiving the claim it talks to the API of the actual storage system to provision the storage. The storage is represented in OpenShift as a **PersistentVolume** which can directly be used by **Pods** to mount it.

With these basics defined we can configure our system for CNS. First we will set up the credentials for CNS in OpenShift.

Configure the Secret

Secrets are a way to store private information, like passwords, certificates, and more. You can learn more about **Secrets** in the secrets documentation

In our case, we want to provide the administrative password for CNS (the heketi API, specifically) so that OpenShift can use it when provisioning or deleting storage volumes.

The definition of this **Secret** is in a file `/opt/lab/support/cns-secret.yaml`. Feel free you use `cat` the display review this file. It contains a base64-encoded version of the heketi admin password that you used earlier when deploying CNS.

/opt/lab/support/cns-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: cns-secret
  namespace: default
data:
  key: bXlTM2NyM3RwYXNzdzByZA==
type: kubernetes.io/glusterfs
```

To create the **Secret** from this file, make sure you are still in the `container-native-storage` **Project**:

```
oc project container-native-storage
```

Then, you can create (instantiate) the **Secret**:

```
oc create -f /opt/lab/support/cns-secret.yaml
```

The **Secret** now stores the credentials for the heketi service in a hashed form. The **StorageClass** will use this to authenticate against the heketi API.

Create the Storage Class

To tell OpenShift about the existence of a dynamic storage provider, you use a **StorageClass**. There is a definition file `/opt/lab/support/cns-storageclass.yaml` that describes the **StorageClass** as well as references the **Secret** you just created. You will need to edit it appropriately. Open the file with your favorite editor, and substitute `INSERT-CLUSTER-ID-HERE` with the ID of your cluster.

Remmeber: you can get the CNS cluster ID from `heketi-cli cluster list`.

cns-storageclass.yaml

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: cns-gold
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "tru
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-container-native-storage.apps.64
```

```
    restauthenabled: "true"
    restuser: "admin"
    volumetype: "replicate:3" (2)
    clusterid: "INSERT-CLUSTER-ID-HERE" (3)
    secretNamespace: "default" (4)
    secretName: "cns-secret" (5)
```

1. The HTTP route OpenShift uses to expose the heketi API
2. The only currently supported GlusterFS volume type: 3-way replication.
3. **Use your cluster UUID here!**
4. The namespace in which we created the secret file before
5. The name of the secret which stores the heketi admin password

Create the **StorageClass**:

```
oc create -f /opt/lab/support/cns-storageclass.yaml
```

With these components in place, OpenShift can now dynamically provision storage capacity from Container-Native Storage. The **StorageClass** has also been setup as the system-wide default so that requests that don’t request a particular **StorageClass** will use this one and end up being served by CNS.

More information about dynamic storage provisioning in OpenShift can be found in the dynamic storage provisioning documentation.

Go to previous module Go to next module