RED HAT SUMMIT

HANDS-ON LABS

LAB GUIDE:

OPENSHIFT FOR OPERATORS

L100070

PRESENTERS:

N. Harrison Ripps, Manager, Software Engineering
Erik Jacobs, Technical Marketing Manager
Jim Minter, Principal Software Engineer

This series of exercises is designed to reflect some of the most common tasks that an operator would need to perform in order to configure and manage an OpenShift cluster. Each exercise should take about 15 minutes. This will be an instructor-led lab, so feel free to follow along or fly solo and take the exercises on at your own pace.

Red Hat Summit 2017 | Boston, MA | May 2-4, 2017

Exercise 1: Meet the Lab Environment

Your lab environment consists of four hosts:

- ose3-ldap.example.com - LDAP server
- ose3-master.example.com - OpenShift master / infra node host and NFS server
- ose3-node1.example.com - OpenShift node host
- ose3-node2.example.com - OpenShift node host

You can SSH into each host from your lab machine using these credentials:

- username: `root`
- password: `r3dh4t1!`

But it's more convenient for us to set up a passphraseless SSH key, so let's walk through that now:

1. From the menu bar on the lab system desktop, select:
   Applications => Utilities => Terminal

2. In the terminal, generate a new, passwordless SSH key:
   a. `ssh-keygen`
   b. When prompted, accept all defaults and do not provide a passphrase. Just hit `[ENTER]`.

3. Now copy that key to each of the cluster hosts (accepting the new host key for each host):
   a. `ssh-copy-id root@ose3-master.example.com`
   b. `ssh-copy-id root@ose3-node1.example.com`
   c. `ssh-copy-id root@ose3-node2.example.com`

4. You should now be able to log in to each host without a password. Type `exit` once connected:
   a. `ssh root@ose3-master.example.com`
   b. `ssh root@ose3-node1.example.com`
   c. `ssh root@ose3-node2.example.com`

Note that you can open multiple terminals. You may want to open one to use for each host, but that's up to you.

You should also be able to view the OpenShift web console in the web browser, but first you will need to make a security exception for the OpenShift master's self-signed certificate:

1. From the menu bar on the lab system desktop, select:
   Applications => Internet => Firefox Web Browser

2. In the browser search bar, enter: https://ose3-master.example.com:8443/

3. You will be warned about the OpenShift server's self-signed certificate. Below the warning, click the button titled "Advanced".

4. In the section that appears, click the button titled "Add Exception..."

5. An "Add Security Exception" window will appear. In the bottom left corner of that window, click the "Confirm Security Exception" button.

6. You should now see the login page for the OpenShift web console.

You're all set with setting up the environment! Now, here's a quick rundown of concepts that we'll be working with:

Pods:
When Kubernetes launches a container or a set of connected containers on a host in the cluster, this assembly is called a Pod.

Projects:
Individual workspaces in the cluster are called Projects. Users will typically have access to one or more projects and may also have the ability to create new ones. All of the pods associated with a given project have access to each other, but communication from one project to another has to be specifically granted by users with the right permissions.

By default, the cluster will have a number of automatically deployed projects:

- *default* - this is a catch-all project that is visible to users with cluster administrative rights. In our lab environment, this project where we are running the internal registry and router.

- *openshift* - this project contains all of the container images for use in any other projects in the cluster. By default, its assets are readably by every cluster user.

- *kube-system*, *logging*, *management-infra*, *openshift-infra* - depending on how you configure your deployment, these projects may be used for various cluster subsystems. In our lab environment, they are unused.

Exercise 2:
Mapping and Syncing LDAP Groups to OpenShift

The LDAP server has users organized into the following groups:

Group	Description
ocp-users	All users with access to the OpenShift Container Platform cluster
portalapp	Developer users involved with the Portal App project
paymentapp	Developer users involved with the Payment App project
ocp-production	Admin / Ops team with privileges to modify production projects
ocp-platform	Operators with full cluster administration privileges

Your first job is to configure OpenShift to associate certain permissions in the cluster with these user groups. In order to map users into these groups, we'll use the groups-sync function of the OpenShift command line utility.

1. `ssh root@ose3-master.example.com` (if you didn't set up passwordless SSH, the password is `r3dh4t1!`)

2. `cd Exercise_02`

3. `cat groupsync.yaml`
    a. This is the file that describes the relationship between the LDAP groups (above) and the groups we'd like to create in the OpenShift cluster.
    b. Note in particular the `groupsQuery` and `usersQuery` blocks, which will be specific to each LDAP implementation.

4. `cp groupsync.yaml /etc/origin/master`

5. `oadm groups sync --sync-config=/etc/origin/master/groupsync.yaml`
    a. This is a test run, it prints to the terminal the group objects that the command would create in the cluster.

   b. You should see five group definitions, as listed above.

6. `oadm groups sync --sync-config=/etc/origin/master/groupsync.yaml --confirm`
   a. Adding the `--confirm` flag to the previous command causes the system to apply the changes.

7. `oc get groups`
      i. Run this to confirm the mapping by checking the groups that have been created in the cluster.
   b. You should see five groups, each with three or more users:
      i. ocp-platform: david, admin1, admin2
      ii. ocp-production: karla, prod1, prod2
      iii. ocp-users: the entire list of users from the other groups
      iv. paymentapp: marina, payment1, payment2
      v. portalapp: andrew, portal1, portal2

With this mapping in place, you can now assign specific OpenShift privileges to match these groups to their responsibilities in the cluster. Let's start by associating one of LDAP groups with cluster-wide administrative privileges.

8. `oadm policy add-cluster-role-to-group cluster-admin ocp-platform`
   a. This applies the 'cluster-admin' role to the 'ocp-platform' group. Anyone in this group now has the same capabilities as the default 'system:admin' account.
   b. **Be careful here!** This mapping is convenient for our lab, but the 'cluster-admin' role should be tightly controlled in a real OpenShift environment.

9. `oc login -u admin1 -p r3dh4t1!`
   a. The admin1 user belongs to the ose-platform group
   b. On login you should see a list of projects that are visible to this user

NOTE: Our lab doesn't cover the configuration of the LDAP server for authentication in the first place. You can see how we have connected our OpenShift system to the LDAP server by looking at `/etc/origin/master/master-config.yaml` under the `oauthConfig` section.

There are various pre-defined roles with certain permissions that come with OpenShift. You can see more about roles in the OpenShift documentation.

Now we're ready to set up the development, test and production environments for our app components and our users.

Here's the overall mapping plan:

Project	Group with Access
Portal App Development	portalapp (marina, payment1, payment2)
Portal App Test	portalapp (marina, payment1, payment2)
Portal App Production	ocp-production (karla, prod1, prod2)
Payment App Development	paymentapp (andrew, portal1, portal2)
Payment App Test	paymentapp (andrew, portal1, portal2)
Payment App Production	ocp-production (karla, prod1, prod2)

And here's what we need to do to set this up:

1. `cat create_projects.sh`
    a. This script calls `oadm new-project` several times to set up projects that we'll be using.
    b. Access rights can be set on a project-by-project basis.

2. `./create_projects.sh`
    a. You should see several lines of confirmation like "Created project <project_name>"

3. `cat add_group_roles.sh`
    a. This script provides the `oadm policy` commands that will add 'admin' privileges to the right user groups.

4. `./add_group_roles.sh`

a. You should see output like “Mapping group <group> to <projects>”

5. `oc login -u payment1 -p r3dh4t1!`
   a. You should see that the payment1 user has access to two projects: paymentapp-dev and paymentapp-test

Exercise 3: Setting up Storage Classes

Storage Classes are a way of setting up persistent storage that offers more flexibility and control than the simple PV / PVC options that have always been a part of the OpenShift platform. Support for Storage Classes was introduced with OpenShift 3.4.

Dynamic Provisioning and Lab Limitations
One of the primary features of Storage Classes is that they can be used to provide storage that is dynamically provisioned. This is explained in detail in our docs, and dynamic provisioning is supported for:

- OpenStack Cinder
- AWS EBS
- GCE PersistentDisk
- GlusterFS
- Ceph RBD

Unfortunately, we can't demonstrate dynamic provisioning against these storage platforms in our lab environment. Instead we'll walk through a workflow that exposes NFS-backed storage as Storage Classes, and highlight the differences between our configuration and a fully dynamic configuration.

Storage Scenario
We've defined two NFS shares to simulate two different types of storage:

- "nfs-fast" will be our placeholder for a high I/O speed storage option (SSD-based, for instance)

- "nfs-slow" will be our placeholder for a slower I/O speed storage option (like rotational-media old school hard drive storage)

We are going to deploy two applications into a new namespace. We will provision the fast storage to one of the apps and slow storage to the other.

So, let's get down to business:

1. `ssh root@ose3-master.example.com`

   a. You can skip this if you are already connected to the master host.

2. `cd ~/Exercise_03`

3. `showmount -e`
   a. This command will display all of the NFS shares that are available on ose3-master.
   b. You should see two shares in the list with the names "/exports/nfs_fast" and "/exports/nfs_slow". For our purposes, we're using these to represent our different storage classes.

4. `cat fast-nfs-storageclass.yaml`
   a. This file is pretty minimal. Note that the 'provisioner:' key is set to 'no-provisioning', and the 'parameters:' list is empty.

The lack of provisioning capabilities is one of the main ways that an NFS-backed storage class differs from other storage classes. For comparison, here is a Storage Class definition for a GlusterFS-backed store:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: gluster-fast
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://127.0.0.1:8081"
  restuser: "admin"
  secretName: "heketi-secret"
  secretNamespace: "default"
  gidMin: "40000"
  gidMax: "50000"
```

The 'provisioner:' here specifies the GlusterFS provisioner, and the 'parameters:' map provides all of the info necessary for OpenShift to communicate with the Gluster service. With these in place, any requests against this "gluster-fast" Storage Class can cause new storage to be automatically provisioned.

Getting back to our exercise, let's walk through the process of linking a Storage Class to a Persistent Volume:

1. `oc login -u admin1 -p r3dh4t1!`

2. `oc create -f fast-nfs-storageclass.yaml`

3. `cat fast-nfs-pv.yaml`
    a. Notice the 'annotations:' map. The 'volume.beta.kubernetes.io/storage-class:' key specifies 'nfs-fast', which is the name that we assigned to our fast NFS Storage Class.

4. `oc create -f fast-nfs-pv.yaml`

5. `oc describe pv/pv-nfs-fast`
    a. The StorageClass should match 'nfs-fast', and the 'Source:' map should reflect the details of the NFS mount on ose3-master.example.com

6. Repeat this for the 'nfs-slow' Storage Class:
    a. `oc create -f slow-nfs-storageclass.yaml`
    b. `oc create -f slow-nfs-pv.yaml`
    c. `oc describe pv/pv-nfs-slow`

Now that these Persistent Volumes have been defined, let's create a new project and see how developers can map them into their app containers:

1. `cat deploy_storage_project.sh`
    a. This script creates a new project, launches an application in the project, and creates some PersistentVolumeClaims against our StorageClass-back PersistentVolumes.

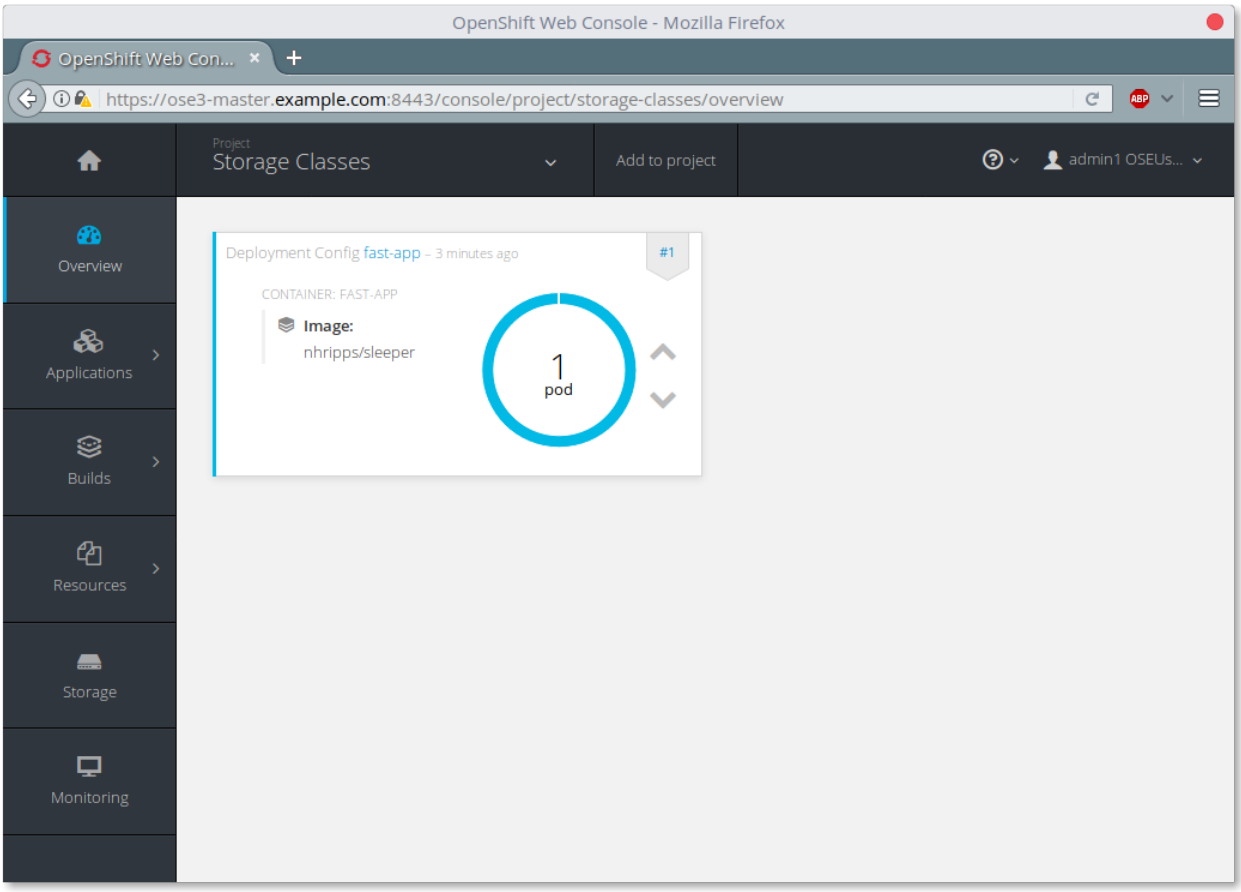
2. `./deploy_storage_project.sh`

3. Jump to the web console - in a web browser navigate to https://ose3-master.example.com:8443/

4. Log in using username `admin1` and password `r3dh4t1!`

5. You can see all of the projects visible to the admin1 user. Find and click on "Storage Classes"

6. You should see a summary view which shows Deployment Config ‘fast-app’ in which there is a running container based on an image called ‘nhripps/sleeper’:



Now that we’ve set everything up, we can walk through the user process for linking an app with a persistent volume:

1. In the left-hand navigation panel, click the Storage tab. Here you should see some PersistentVolumeClaims that were created by our deploy script. One is bound to the ‘fast’ PersistentVolume, and the other to the ‘slow’ PersistentVolume.

2. Now use the left-hand navigation panel to go to: Applications => Deployments. You should see the ‘fast-app’ Deployment. Click into ‘fast-app’ to see a summary of the Deployment configuration.

3. In the Template section of this Deployment summary page, find the Volumes heading and the 'Add storage' link below it. Click this link to move to the 'Add Storage' view.

4. In the 'Add Storage' view:
    a. You should see that you can choose which PersistentVolumeClaim to use for this deployment. Let's use 'pvc-nfs-fast'.
    b. Under the 'Volume' heading, set the value of 'Mount Path' to /mountpoint. This is an existing directory path inside of the container that will become a mount point for our volume.
    c. Leave the 'Volume Name' field blank
    d. Press the 'Add' button to apply this change.

5. You should be looking at the Deployment summary page again. Under the 'Volumes' heading, you should see an entry for the volume that you just added.

To check our work and confirm that we were mounted through to the right storage class, we can put a file in the mounted volume and see where it ended up on the NFS back end.

1. Using the left-hand navigation panel, now go to Applications => Pods. You should see at least one pod with a name like 'fast-app-2-.....`. Click on this pod to switch into the Pod summary view.

2. Click on the terminal tab. You will be connected to the running pod and presented with a shell.

3. In the shell, type:
    a. `echo fast >/mountpoint/fast.txt`

4. Now leave the web browser and go back to the terminal where you have been working through this exercise.

5. From the terminal, type:
    a. `cat /export/nfs_fast/fast.txt`
    b. You should see that 'fast.txt' was created and contains the string that you echoed into it from the pod environment.

StorageClasses become really powerful as the basis for auto-provisioning data stores. But, even in this simple example you can see that using named storage classes gives you.

Exercise 4: Quotas and Limits

Quotas describe how much of a resource is available across an entire project (or group of projects), while Limits describe the minimums and maximums for individual pods and containers within a project. Here is a rundown of the resources that can be controlled using these concepts:

Quotas	Limits

• cpu (how is it measured?)
  ○ total requests
  ○ total cpu
• memory
  ○ total requests
  ○ total memory
• storage
  ○ # requests
  ○ total storage
• cluster object totals:
  ○ pods
  ○ replicationcontrollers
  ○ resourcequotas
  ○ services
  ○ secrets
  ○ configmaps
  ○ persistentvolumeclaims
  ○ imagestreams
	• containers
	  ○ cpu (min/max/ratio)
	  ○ memory (min/max/ratio)
	• pods
	  ○ cpu (min/max/ratio)
	  ○ memory (min/max/ratio)
	• images
	  ○ storage (max)
	• image streams
	  ○ tags (max)
	  ○ images (max)
	• persistent volume claims
	  ○ storage (min/max)

If you want to set default quotas on limits for every new project on the cluster, you can modify the template for new projects following the instructions in our documentation. For this exercise, we're going to focus on applying them to a few projects that we created in Exercise 2.

In order to apply these quotas and limits, we'll need to create quota and limit definition files, and then apply them to relevant projects.

1. `oc login -u admin1 -p r3dh4t1!`

2. `cd ~/Exercise_04`

3. `cat resource-quotas-non-prod.yaml`
    a. We're going to set some hard limits on the non-production projects.
    b. In a real system these limits would probably be too low to be useful.

4. `cat apply_non_prod_quotas.sh`
    a. This script applies the quota definition to the four non-production projects.

5. `./apply_non_prod_quotas.sh`

6. `cat resource-limits-non-prod.yaml`
    a. These limits conform to the rules established in our quota definition.

7. `cat apply_non_prod_limits.sh`
    a. Similar to the other 'apply' script, this runs through and creates limits for each non-production project.

8. `./apply_non_prod_limits.sh`

9. `oc get resourcequota/quota -o yaml -n portalapp-dev`
    a. This lists out the quota that we applied against one of the non-production projects

10. `oc get limitrange/limits -o yaml -n portalapp-dev`
    a. This lists out the limitrange as applied to the same project.

11. `cat deploy_apps.sh`
    a. These instructions will deploy a basic "Hello World" app to our non-production projects

12. `./deploy_apps.sh`

That's the admin work. Now let's take a look at one of these apps in the web browser.

1. Navigate to https://ose3-master.example.com:8443/console/project/paymentapp-dev/quota

   a. This takes you to the Quota page for the Payment App Development project.

2. Now jump to Overview tab (left-hand navigation), and use the up-arrow button next to the Pods counter to raise the count to 4.

3. Head back to Resources => Quotas. You can see that with four instances of the app running, we've maxed out our allotment of CPU, memory, and pods.

Exercise 5: Multi-Tenant Networking

The OpenShift Multitenant SDN plug-in enables a true isolated multi-tenant network infrastructure inside OpenShift's software defined network. While we have seen projects isolate resources through OpenShift's RBAC, the multitenant SDN plugin isolates projects using separate virtual network IDs within Open vSwitch.

The multitenant network plugin was introduced in OpenShift 3.1, and more information about it and its configuration can be found in the networking documentation. Additionally, other vendors are working with the upstream kubernetes community to implement their own SDN plugins, and several of these are supported by the vendors for use with OpenShift. These plugin implementations make use of appc/CNI, which is outside the scope of this lab.

Exercise Scenario
The default behavior of the multitenant SDN is to isolate projects from one another, but they are all joined to the "default" project (where, typically, the registry and router live). We will now create two projects, verify that they are network isolated, and then use various "oc" commands in order to manipulate the project networks and observe the subsequent behavior.

Let's create two projects, each running a single pod:

1. `oc login -u admin1 -p r3dh4t1!`

2. `cd ~/Exercise_05`

3. `cat net-proj.sh`
    a. This script will create two projects and load the pods

4. `./net-proj.sh`

Now that you have some networks and pods, you will need to find the IP address of the pod in the "networkb". The following command will show you the IP address:

5. `./podbip.sh`

The output will simply be the IP address of the pod in the “networkb” project. The everyday way to do this would be with a combination of the “get” and “describe” verbs. Feel free to do the following to verify what the script did:

6. `oc get pod -n networkb`

7. `oc describe pod` ***`oc-1-f0deb`***
    a. Make sure to substitute the correct pod name in the describe command.

“describe” will show you a lot of information about the pod, including its IP address on the software defined network. Either way, make note of the IP address you found above. It will look something like ***10.1.4.12***. The CIDR for pod IPs is configurable at the time the cluster is installed.

Export the IP address of your pod:

8. `export POD_B_IP=`***`10.1.4.12`***
    a. **Make sure you use the IP address you found earlier**

The OpenShift command-line tool (and the web console) provide mechanisms to execute commands inside containers. This is a useful feature for both developers as well as for cluster and application operators/administrators. We will use that feature in order to test network connectivity between the two pods you created.

To test connectivity from the pod in networka to networkb, execute the following, taking careful note of where you need to change the commands:

9. `oc get pod -n networka`
    a. This will show you the singular pod in the project “networka”. Note the name

10. `export POD_A_NAME=`***`oc-1-zr3bn`***
    a. **Make sure you use the pod name you found in #1**

11. `oc exec -n networka $POD_A_NAME -- ping -c1 -W1 $POD_B_IP`

You will see 100% packet loss (your ping command sends 1 packet, waits 1 second, and gets no response). This is because the networks are not connected to one

another. Now simply execute the following:

12. `ping -c1 -W1 $POD_B_IP`

You will see a successful ping. This is because the master (the system you are on) is also a node attached to the SDN. At the host level you are able to reach across all networks, virtual or otherwise. This is important to keep in mind when you consider the overall network-level security of your cluster. Now it's time to join the networks. Execute the following:

13. `oc adm pod-network`
    a. You will see a bunch of options. We are going to work with "join-projects".

14. `oc get netnamespaces`
    a. You will see the networka / networkb projects have separate net ids.

Joining the two projects will collapse them onto a single network namespace (network id). Execute the following:

15. `oc adm pod-network join-projects --to=networka networkb`

16. `oc get netnamespaces`

You will note that the network ID for networkb was changed to match that for networka. Now, execute your ping again:

17. `oc exec -n networka $POD_A_NAME -- ping -c1 -W1 $POD_B_IP`

You should see your ping now works! This is a demonstration of manipulating the software defined network with a few simple OpenShift CLI commands. If you like, isolate the projects again and re-test with the following:

1. `oc adm pod-network isolate-projects networkb`

2. `oc get netnamespaces`

3. `oc exec -n networka $POD_A_NAME -- ping -c1 -W1 $POD_B_IP`

Your ping should now fail again.

The “make-projects-global” option effectively makes a project’s network accessible from all other projects. If you wish, you can try using it. Just don’t make any changes to the “default”, “openshift”, “openshift-infra”, or “management-infra” projects.

Exercise 6: Pruning Old Data

Over time as OpenShift users build and deploy applications, it is likely that old builds, deployments, and unused container images will build up. On the one hand, this can be a good thing, because among other things it enables users to immediately rollback their applications to any previous version. On the other hand, especially in a large production cluster, too many of these objects can take up excessive disk space, both in etcd and in the OpenShift registry.

Administrators can use the oadm prune command, potentially from a cron job, to keep control over old builds, deployments, and unused container images across an entire cluster. See the documentation at https://docs.openshift.com/container-platform/3.4/admin_guide/pruning_resources.html for full details.

Log in to the "cleanme" project to see a project in need of a clean-up. In a large production cluster, there could be a lot of projects just like this!

1. `oc login -u admin1 -p r3dh4t1!`

2. `oc project cleanme`

3. `oc get bc,builds,pods`
    a. Although all the build pods are marked completed, their logs will be taking up disk space on the OpenShift nodes. Each build object also takes up a small amount of space in etcd.

4. `oc describe imagestream sampleapp`
    a. The sampleapp imagestream references every sampleapp container image successfully built. Each container image will be taking up disk space in the OpenShift registry.

5. `oc get dc,rc`
    a. Each previous application deployment holds on to a replication controller, in case a rollback is required later. Each rc object takes up a small amount of space in etcd.

Let's start by trying to prune unused images from the registry. To do this, we need our admin user to be a member of the system:image-pruner group.

6. `oadm policy add-cluster-role-to-group system:image-pruner ocp-platform`

7. `oadm prune images`
    a. Running oadm prune without `--confirm` will do a dry run to show what would be removed.
    b. Right now, no images will be removed, because all the old images are still referenced by the replication controllers.

So let's remove the old deployments. Note that on all calls to oadm prune, the admin user can define policy around how many old items to keep (e.g. using `--keep-younger-than=` or `--keep-complete=`). See `oadm prune -h` for more details.

8. `oadm prune deployments --keep-complete=3 --confirm`

9. `oc get dc,rc`
    a. All the old replication controllers should have been removed, except the currently running deployment and the 3 previous ones.

Now we should be able to prune those images.

10. `oadm prune images --confirm`

11. `oc describe imagestream sampleapp`
    a. Just like the replication controllers, just the image for the currently running deployment and the 3 previous ones should be left.

Finally, let's also prune the old builds, leaving the (default) 5 previous builds.

12. `oadm prune builds --confirm`

13. `oc get bc,builds,pods`

Exercise 7: Node Evacuation

So far we've focused almost entirely on a number of user- and project-level operations. In this lab, we move our focus to managing the cluster itself. Specifically, we're going to evacuate one of the nodes and move the applications that were running there to the remaining node in the cluster.

1. `oc get pods --all-namespaces -o wide`
    a. The 'hello-world' applications that were previously deployed should be distributed evenly between ose3-node1 and ose3-node2

2. `oc get nodes`
    a. currently all nodes should have the 'Ready' status

3. `oadm manage-node ose3-node2.example.com --schedulable=false`
    a. This command will change ose3-node2's status to Ready,SchedulingDisabled
    b. In this state, the node continues to run existing pods, but will not take on new pods

4. `oadm manage-node ose3-node2.example.com --list-pods`
    a. This command specifically lists out the pods running on ose3-node2

5. `oadm manage-node ose3-node2.example.com --evacuate --dry-run`
    a. The `--evacuate` flag will force pods on ose3-node2 to be moved to other nodes
    b. The `--dry-run` flag prevents the utility from actually evacuating but shows you what actions would occur during an evacuation.

6. `oadm manage-node ose3-node2.example.com --evacuate`
    a. This will cause the system to evacuate ose3-node2

7. `oc get pods --all-namespaces -o wide`
    a. After a few moments, all of the pods should have been terminated on ose3-node2 and new copies restarted elsewhere.

   b. Note that In a production environment, grace periods can and may be set on container termination; this may translate to longer wait times before the node is evacuated.

To add a net-new node to the cluster, we could add information about the new node to the inventory file for our ansible playbook and re-run the playbook. However, in situations such as this one, where we have a node that is currently unschedulable but otherwise configured for cluster use, we can simply change the scheduling status to make it available again.

1. `oadm manage-node ose3-node2.example.com --schedulable=true`
   a. This changes the state of ose3-node2 in the cluster.

2. `oc get nodes`
   a. Confirm that the status of ose3-node2 is now 'ready'

3. `oc get pods --all-namespaces -o wide`
   a. Even though ose3-node2 is schedulable, existing pods are not automatically migrated.

4. `oc get pods -n portalapp-dev -o wide`
   a. Let's try deleting the hello-world app running in the portalapp-dev project.

5. Copy the pod name that is printed in the output of this command (something like "hello-openshift-1-abc12")

6. `oc delete pod/<paste pod name here> -n portalapp-dev`
   a. This will delete the running portalapp-dev copy of hello-world

7. `oc get pods -n portalapp-dev -o wide`
   a. In short order you should see another running copy of the hello-world app.
   b. It is not guaranteed that this pod will be running on ose3-node2, but it is very likely.

In any OpenShift cluster, the node where a pod lands is dependent on the cluster scheduler. In our lab environment, the scheduler's configuration (in /etc/origin/master/scheduler.json, using the out of the box defaults) includes the

LeastRequestedPriority and BalancedResourceAllocation priorities, both of which will favor balancing the pod across ose3-node1 and ose3-node2.

Exercise 8: Readiness and Liveness Probes

As we have seen before in the UI via warnings, there is a concept of application health checks in OpenShift. These come in two flavors:

- Readiness probe
- Liveness probe

From the Application Health section of the documentation, we see these definitions:

Liveness Probe
A liveness probe checks if the container in which it is configured is still running. If the liveness probe fails, the kubelet kills the container, which will be subjected to its restart policy. Set a liveness check by configuring the `template.spec.containers.livenessprobe` stanza of a pod configuration.

Readiness Probe
A readiness probe determines if a container is ready to service requests. If the readiness probe fails a container, the endpoints controller ensures the container has its IP address removed from the endpoints of all services. A readiness probe can be used to signal to the endpoints controller that even though a container is running, it should not receive any traffic from a proxy. Set a readiness check by configuring the `template.spec.containers.readinessprobe` stanza of a pod configuration.

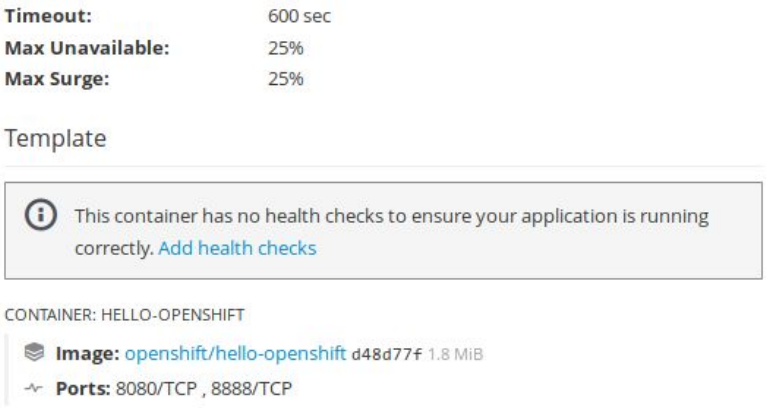
We will use the web console to add these probes to our portalapp-dev application.

1. Log into the web console at https://ose3-master.example.com:8443/ as the `admin1` user, password `r3dh4t1!` After logging in, you will see a list of projects that are visible to the admin1 user.

2. Select 'Portal App Development' from the project list. You will be presented with a summary view of the project.

3. From the left-hand navigation panel, select Applications => Deployments. You will see a list of deployments defined for this project.

4. Select the hello-openshift deployment from the deployments list. The browser will display a summary of the deployment definition. Under the Template heading, note the warning about missing health checks:



5. In the warning note about missing health checks, click on the 'Add health checks' link. You will see the 'Health Checks: hello-openshift' page with headings for Readiness Probe and Liveness Probe. Click both of the links - 'Add Liveness Probe' and 'Add Readiness Probe' - to expand the forms on this page.

6. Use the following values for the Readiness and Liveness probes:
    a. Type: HTTP
    b. Path: /healthz
    c. Port: 8080
    d. Initial Delay: 20 for the Readiness probe, 120 for the Liveness probe
    e. Timeout: 1

7. At the bottom of the form, click the 'Save' button. The deployment summary is updated to reflect the new probe definitions.

8. From the left-hand navigation panel, select Overview.

By updating the deployment object with liveness and readiness probes, we've indirectly triggered a redeployment of the hello-openshift app. On the Overview page you can watch as a new pod is deployed and the old one is taken down.

Readiness and liveness settings can also be set from the command line:

1. From a terminal, log in to ose3-master.example.com.

2. `oc login -u admin1 -p r3dh4t1!`

3. `oc project portalapp-dev`
    a. This switches the oc client to the project of interest.

4. `oc set probe dc/hello-openshift --liveness --initial-delay-seconds=60 -n portalapp-dev`
    a. This command updates the existing deployment configuration for our app, which also triggers another deployment.