SL44232: OpenShift for Operators

Presenters:

- N. Harrison Ripps, Manager, Software Engineering, Atomic / OpenShift Team
- Erik Jacobs, Principal Technical Marketing Manager, OpenShift Enterprise
- Siamak Sadeghianfar, Senior Product Marketing Manager, OpenShift Enterprise
- Bob Kozdemba, Principal Solution Architect, Red Hat Public Sector

This series of self-guided labs is targeted towards the people who will deploy and manage a Red Hat OpenShift Enterprise environment. Over the next two hours, we will walk through a number of scenarios that will be helpful to operators deploying OpenShift on private or public clouds.

Lab 1: Deployment

Your lab environment consists of three hosts configured to run as a Red Hat OpenShift Enterprise cluster. But what if you are starting from scratch? Let's a look at how we would configure and run a deployment of our own.

1.1 Host Access

From your lab machine you have SSH access to the three virtual hosts that make up the cluster:

- ose3-master.example.com
- ose3-node1.example.com
- ose3-node2.example.com

During the following labs you will be directed to connect to these hosts either directly or indirectly as documented. The login credentials for all three hosts are:

Username: **`root`**
Password: **`redhat1!`**

And your SSH commands will be of the form:

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

Now let's see how we would configure and deploy software packages on these hosts if they weren't already set up…

1.2 Configuring and Installing OpenShift with Ansible

Ansible is a configuration and deployment tool that is similar to Puppet and Chef. It is *agentless*, meaning that it does not require special software to be installed target host systems in order to do its work.

By using one of the freely distributed Ansible *playbooks* to configure your cluster, your list of prerequisites for each target host is pretty small:

- RHEL 7.1
- An appropriate RHEL subscription
- SSH access to a privileged user account

Ansible was used in the initial deployment of the OpenShift cluster on your lab system. However, we've cleaned out the Ansible configuration so that you can set up the configuration yourself and then check your work.

To begin, from a terminal on your lab system, log into the master host. Remember that the password is 'redhat1!':

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

Once connected, Ansible's inventory of hosts lives at `/etc/ansible/hosts`. Have a look at the inventory on the master:

```
[root@ose3-master ~]# less /etc/ansible/hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers.

green.example.com
blue.example.com
192.168.100.1
192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group

[webservers]
alpha.example.org
beta.example.org
192.168.1.100
192.168.1.110

# If you have multiple hosts following a pattern you can specify
# them like this:

www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group

[dbservers]

db01.intranet.mydomain.net
db02.intranet.mydomain.net
10.25.1.56
10.25.1.57

# Here's another example of host ranges, this time there are no
# leading 0s:
db-[99:101]-node.example.com
```

This is the default hosts file, and it shows us a number of different ways of defining hosts and groups of hosts.

The OpenShift Ansible playbook defines some groupings that make it very easy to configure and deploy an initial cluster, and then to extend that cluster later on.

We're going to configure Ansible as though we were setting up our three-host cluster from scratch.

1. Move `/etc/ansible/hosts` to `/etc/ansible/hosts.orig` just in case you need to go back and start over.

2. Next, copy `/root/ansible.hosts.ose.example` to `/etc/ansible/hosts`.

1.2.1 Initial Setup

To begin, open the newly placed `/etc/ansible/hosts` file in an editor and have a look at the contents of the file. Note that the `OSEv3:children` group starting on line 4 contains four subgroups:

- masters - hosts in this group will run the cluster master processes, which include the API server and controller manager server.
- nodes - hosts in this group actually run containerized payloads and the services to support those payloads, including docker, the kubelet, and the Kubernetes service proxy.
- etcd - Kubernetes leverages `etcd` for configuration management, and `etcd` itself can run as a multi-instance cluster. This group is optional.
- lb - This load balancer group is used when you are installing multiple masters, and you would like Ansible to deploy and configure an *unsupported/non-production* HAProxy load balancer that is designed to handle the traffic routed to those masters. This group is also optional.

Comment out (prepend with '#') the `etcd` and `lb` lines in this section. We are doing a single-master deployment and so we will not use these groups.

1.2.2 Masters and Nodes

Next, we'll fill out the groups that we *will* use by adding our master and node hosts to the file.

1. Search on "`[masters]`" in the file. You should only find one result at around line 259.

2. Modify or replace the hostname in this list to read: `ose3-master.example.com`

3. Below the masters group, comment out the `[etcd]` and `[lb]` groups and their contents by adding a hash symbol (#) to the beginning of each of those lines. These groups are valuable for multi-master clusters, but our lab deployment is only using one master. In a single master situation, we don't need the load balancer at all, and without an explicit entry in the `[etcd]` group, the etcd server will be added to the single master.

4. Finally, jump down a few lines further to the `[nodes]` group. The note that precedes this group mentions that master hosts must also be included in the nodes list, but that you can prevent them from being targeted for running containers with the `openshift_schedulable=False` setting. For our lab we're actually going to use the master as a node, but we are going to tag it with a specific label so that it is only used for supporting services and not application deployment.

   Replace the existing multi-master host line (right below the `[nodes]` heading) with an entry for our single master host. Use the `openshift_node_labels` argument to indicate that the master should be treated as an 'infra' environment node, and the `openshift_schedulable` argument to specifically indicate that we want to use our master host for some container payloads:

   ```
   ose3-master.example.com
   openshift_node_labels="{'region':'tatooine','zone':'na','env':'in
   fra'}" openshift_schedulable=True
   ```

5. Now replace the existing multi-node host line with individual lines for each of our nodes. This time use the `openshift_node_labels` argument to specify 'user' environment nodes, but don't worry about providing `openshift_schedulable` because non-master hosts are schedulable by default:

```
ose3-node1.example.com openshift_node_labels="{'region':
'tatooine', 'zone': 'cantina','env':'user'}"
ose3-node2.example.com openshift_node_labels="{'region':
'tatooine', 'zone': 'farm','env':'user'}"
```

Be aware that the ‘env’ labels that we are using to distinguish between the master and nodes are *completely arbitrary*; we could have used practically any key/value pair to distinguish them. That said, while the specific label is arbitrary, the importance of having some way to distinguish between master hosts and node hosts is not, as you will see later on.

That’s it for masters and nodes. Keep the hosts file open in your editor because next we’re on to adding an LDAP server for identity management.

1.2.3 IDM Setup, Part 1

In our lab environment, we’ve already configured an LDAP server that is reachable at:

```
ose3-ldap.example.com
```

There are two primary steps to using LDAP with an OpenShift cluster:

1. Configure the cluster hosts with necessary information to communicate with the LDAP server.

2. Map LDAP users and groups to permissions within the cluster.

The first step is handled now, at cluster deployment time, with the Ansible playbooks. We’ll cover step two in the following lab.

To configure the ansible playbook to set up LDAP, we’ll make the following changes to the `/etc/ansible/hosts` file:

1. By default, the hosts file will configure the cluster to use basic htpasswd-based authentication. The cluster only supports the use of a single IDM provider per deployment,

so we need to disable this default before enabling LDAP. Search on '`htpasswd auth`' in the hosts file and comment out the line immediately below it (this should be line 84) that begins with "`openshift_master_identity_providers`"

2. Now we need to enable LDAP. Search on "LDAP auth", which is about 5 lines down from where you just commented out the htpasswd config. Uncomment and update the next line to identify the LDAP server that we're going to use with this information:

```
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'r3dh4t1!', 'ca': '/etc/origin/master/ipa-ca.crt',
'insecure': 'false', 'url':
'ldap://ose3-ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub
?(memberOf=cn=ose-users,cn=groups,cn=accounts,dc=example,dc=com)'}]
```

   Make sure there are no line breaks in this. Also note that this configuration refers to a certificate authority (CA) cert that lives at `/etc/origin/master/ipa-ca.crt`. This cert file has already been copied from the LDAP server host to all of the hosts in the cluster, but if you were setting things up from scratch, you would need to do this manually.

3. Save the file, but leave it open - we've got one last change to make.

1.2.4 Default Node Selector

Finally we will establish a system-wide default to let OpenShift know which nodes should be the default targets for user workloads. Back in section 1.2.2, recall that we assigned the `env=user` label to both of our node hosts, but not to our master host. By doing this, we can now specify that label as a default node selector. Unless we specifically override the default, our containers will always end up on one of the nodes with this label.

To establish the default, search on 'default project'; you should end up around line 141 in the file. Uncomment the setting line below it and modify it to read:

```
osm_default_node_selector='env=user'
```

That's it! Save the hosts file and let's test it out.

1.2.5 Test Your Setup

In our lab environment, the cluster that you are working in has already been deployed with the exact settings that you've configured in the steps outlined in this section. So, rather than rerunning the deployment playbook, instead run the following Ansible-based validation utility to confirm your settings:

```
ansible-playbook /root/configchecker.yml
```

If the Ansible playbook exits with "FATAL":

- During our instructor-led lab, raise your hand and have an instructor/proctor assist you. We can help, and we don't you to get too hung up on this - there's a lot of other content to cover!

- If you are taking this lab at a self-paced kiosk, you can work on debugging this, or start with a fresh copy of the /root/ansible.hosts.ose.example file, or just move on to the next section; the cluster is already deployed so you don't need to worry about being blocked by this step.

If the Ansible run looks successful, you know that deploying this configuration would result in the exact cluster configuration that you are currently using. Head on to the next lab.

Lab 2: User and Project Administration

In this lab, we'll pick up with the cluster at the point where an Ansible playbook deployment will have completed. We'll finish setting up LDAP-based authentication, deploy a few containers that are used by the cluster, and get our hands dirty with some project setups.

2.1 IDM Setup, Part 2

The next step in configuring your OpenShift cluster work with an external IDM like LDAP is to map groups defined within the IDM to groups within OpenShift.

After the Ansible deployment, it should already be possible to authenticate against the LDAP server. You can test this by logging in to the cluster master at ose3-master.example.com and then running the oc login command:

```
$ oc login -u andrew
```

When prompted, use the password `r3dh4t1!`

You should see output like:

```
Login successful.

You don't have any projects. You can try to create a new project, by
running

   $ oc new-project <projectname>
```

If that doesn't work, "Houston, we have a problem!" Please let a proctor know, because the LDAP server that we're using in these labs is centrally located for all machines in the lab environment.

2.1.1 Group Sync Configuration

So now on to group mappings. Users on the lab LDAP server are organized into the following groups:

Group	Description
`ose-users`	Users with access to OpenShift Enterprise
`portalapp`	Portal App project developers
`paymentapp`	Payment App project developers
`ose-production`	Administrators and operations team with access to modify production projects
`ose-platform`	Users with full cluster administration control

To map these groups to something within OpenShift, we'll start by creating a group sync file.

On the ose3-master host, start a file at `/etc/origin/master/groupsync.yaml` with this boilerplate:

```
[root@ose3-master ~]# cat << EOF > /etc/origin/master/groupsync.yaml
kind: LDAPSyncConfig
apiVersion: v1
url: "ldap://ose3-ldap.example.com"
insecure: false
ca: "/etc/origin/master/ipa-ca.crt"
bindDn: "uid=admin,cn=users,cn=accounts,dc=example,dc=com"
bindPassword: "r3dh4t1!"
rfc2307:
   groupsQuery:
       baseDN: "cn=groups,cn=accounts,dc=example,dc=com"
       scope: sub
       derefAliases: never
       filter: (&(!(objectClass=mepManagedEntry))(!(cn=trust
admins))(!(cn=groups))(!(cn=admins))(!(cn=ipausers))(!(cn=editors))(!(cn=ose-users
))(!(cn=evmgroup*))(!(cn=ipac*)))
   groupUIDAttribute: dn
   groupNameAttributes: [ cn ]
```

```
   groupMembershipAttributes: [ member ]
   usersQuery:
       baseDN: "cn=users,cn=accounts,dc=example,dc=com"
       scope: sub
       derefAliases: never
       filter: (memberOf=cn=ose-users,cn=groups,cn=accounts,dc=example,dc=com)
   userUIDAttribute: dn
   userNameAttributes: [ uid ]

EOF
```

Next, open the new file in an editor and add the mapping section, which will look like this:

```
groupUIDNameMapping:
 "cn=portalapp,cn=groups,cn=accounts,dc=example,dc=com": "portalapp"
 "cn=paymentapp,cn=groups,cn=accounts,dc=example,dc=com": "paymentapp"
 "cn=ose-production,cn=groups,cn=accounts,dc=example,dc=com": "ose-production"
 "cn=ose-platform,cn=groups,cn=accounts,dc=example,dc=com": "PlatformAdmins"
```

Remember that this is a YAML file, so indenting is important! The groupUIDNameMapping key should be at the same depth as the userNameAttributes key, and the subsequent mapping entries should be one level deeper.

2.1.2 Test and Deploy

Before we continue, make sure you are logged in to the cluster as a cluster administrator. For now we'll use the built-in 'system:admin' account. Note that you can only authenticate as 'system:admin' if you are logged into one of the master hosts as the root user:

```
[root@ose3-master ~]# oc login -u system:admin
```

Now, to validate the groupsync file and test the synchronization without making changes to the cluster, run the sync-groups command:

```
[root@ose3-master ~]# openshift ex sync-groups \
--sync-config=/etc/origin/master/groupsync.yaml
```

You should see YAML output for each of the groups; it will be similar to:

```
apiVersion: v1
items:
- apiVersion: v1
 kind: Group
 metadata:
   annotations:
     openshift.io/ldap.sync-time: 2016-03-18T02:46:26-0400
     openshift.io/ldap.uid: cn=paymentapp,cn=groups,cn=accounts,dc=example,dc=com
     openshift.io/ldap.url: idm.example.com:389
   creationTimestamp: null
   labels:
     openshift.io/ldap.host: idm.example.com
   name: paymentapp
 users:
 - andrew
 - payment1
 - payment2
...
```

Assuming that worked correctly, you can now re-run the sync and actually apply the settings by executing the previous command and adding the --confirm flag:

```
[root@ose3-master ~]# openshift ex sync-groups
--sync-config=/etc/origin/master/groupsync.yaml --confirm
```

You will see output like:

```
group/portalapp
group/paymentapp
group/ose-production
group/ose-platform
```

Finally, to verify that the groups were created:

```
[root@ose3-master ~]# oc get groups
```

Should return output like:

```
NAME             USERS
ose-platform     david, admin1, admin2
ose-production   karla, prod1, prod2
paymentapp       marina, payment1, payment2
portalapp        andrew, portal1, portal2
```

2.1.3 Policy Bindings

With IDM-to-OpenShift mapping completed, the last step is to actually elevate the permissions of those groups on the cluster. We'll set up most of the groups later, but for now, run the following command to set the privileges of the `ose-platform` group:

```
[~]# oadm policy add-cluster-role-to-group cluster-admin ose-platform
```

Simply put, this grants the cluster-admin role to the ose-platform group. Having done that, now you can switch from the built-in `system:admin` user to an LDAP-managed user with the same privileges. This is equivalent to working on a Linux system as a user with superuser privileges rather than running directly as root. Remember the `admin1` password is '`r3dh4t1!`':

```
[root@ose3-master ~]# oc login -u admin1
Authentication required for https://ose3-master.example.com:8443 (openshift)
Username: admin1
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

 * default (current)
 * management-infra
 * openshift
 * openshift-infra

Using project "default".
```

The project list above is identical to what is available to `system:admin`.

2.2 'default' Project Configuration

The 'default' project on an OpenShift cluster is the reserved project namespace for cluster management assets that are *not* part of user-owned projects. To say it another way, it is the namespace that contains pods and services that make the cluster work, rather than a namespace where users would deploy applications.

During the first lab, we noted that we want to use the master host as the landing place for infrastructure objects. However, we also configured the cluster with an `osm_default_node_selector` value of 'env=user'. As you will see, this may present a problem.

Have a look at the current node configuration:

```
[root@ose3-master ~]# oc get nodes
```

Recall that we configured the node hosts with the 'env=user' label, but not the master host:

```
NAME                      LABELS          STATUS    AGE
ose3-master.example.com   env=infra,...   Ready     1h
ose3-node1.example.com    env=user,...    Ready     1h
ose3-node2.example.com    env=user,...    Ready     1h
```

Because of this, if we start deploying cluster assets to the 'default' project, they are going to end up running on one of the nodes with a label of 'env=user'. That's not what we want, because those nodes are meant for user payloads, and we want our cluster assets to go to the master node.

In order to solve this problem, we need to specifically associate the 'default' project with the master node. We can accomplish this with an *annotation* that links the 'default' namespace to the 'env=infra' node label.

To do this, we'll modify the 'default' namespace with the `oc edit` command:

```
[root@ose3-master ~]# oc edit namespace/default
```

When you run this, you are dropped into an editor with a YAML representation of the ‘default’ namespace:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
will be
# reopened with the relevant failures.
#

apiVersion: v1
kind: Namespace
metadata:
 annotations:
   openshift.io/sa.initialized-roles: "true"
   openshift.io/sa.scc.mcs: s0:c3,c2
   openshift.io/sa.scc.supplemental-groups: 1000010000/10000
   openshift.io/sa.scc.uid-range: 1000010000/10000
 creationTimestamp: 2016-05-19T14:00:20Z
 name: default
 resourceVersion: "2685"
 selfLink: /api/v1/namespaces/default
 uid: 063e1563-1dca-11e6-b6ec-525400b33d1d
spec:
 finalizers:
 - kubernetes
 - openshift.io/origin
status:
 phase: Active
```

Now let’s walk through the change we want to make:

1. We’re going to add a new annotation that names a node-selector. The new line will look like this:
   `openshift.io/node-selector: env=infra`

2. Copy this line into the editor, adding it to the top of the annotations list. Again, this is a YAML file so be sure to use proper indentation, and take care not to change any other lines! The `oc edit` command is very powerful, but accidentally modifying something like

the project UID can land you in trouble.

3. Finally, save and exit from the editor.

If everything went well, you'll see a line like this and be returned to the terminal prompt:

```
namespaces/default
```

If you had an error, you will find yourself back in the editor. In the commented (#) field above the 'default' namespace object, you will see an error message about what went wrong.

When everything looks good, we can proceed to deploy some cluster management assets.

2.3 Registry Setup

OpenShift can build Docker images from your source code, deploy them, and manage their lifecycle. To enable this, OpenShift provides an internal, integrated Docker registry that can be deployed in your OpenShift environment to locally manage images.

To deploy the integrated Docker registry, use the `oadm registry` command as a user with cluster administrator privileges. You are currently authenticated as user admin1, which has the appropriate permissions:

```
[root@ose3-master ~]# oadm registry \
--config=/etc/origin/master/admin.kubeconfig \
--credentials=/etc/origin/master/openshift-registry.kubeconfig \
--images='registry.access.redhat.com/openshift3/ose-${component}:${version}'
```

You can confirm that the registry has deployed by checking out a few of the objects that the oadm registry command creates:

```
[root@ose3-master ~]# oc get pods
NAME                      READY     STATUS    RESTARTS   AGE
docker-registry-1-kvfls   1/1       Running   0          12m
```

```
[root@ose3-master ~]# oc get services
NAME            CLUSTER_IP   EXTERNAL_IP PORT(S)                SELECTOR
docker-registry 172.30.23.43 <none>      5000/TCP              docker-regist...
kubernetes      172.30.0.1   <none>      443/TCP,53/UDP,53/TCP <none>

[root@ose3-master ~]# oc get rc
CONTROLLER        CONTAINER(S) IMAGE(S)           SELECTOR
docker-registry-1 registry     registry.access... deployment=docker-registry...
```

The last item in this list is a replication controller. If you are not familiar with how Kubernetes works, it is helpful to know that the replication controller is a mechanism that will automatically restart the docker registry in the event that the running pod terminates or is destroyed.

Once the registry pod (from `oc get pods`) status is 'Running', you should be able to run:

```
[root@ose3-master ~]# oadm registry --dry-run
```

And get the following output:

```
Docker registry "docker-registry" service exists
```

you can move on to the next step.

2.4 Router Setup

The OpenShift router is the ingress point for all external traffic destined for services in your OpenShift installation. OpenShift provides and supports the following two router plug-ins:

- The *HAProxy* template router is the default plug-in. It uses the `openshift3/ose-haproxy-router` image to run an HAProxy instance alongside the template router plug-in inside a container on OpenShift. It currently supports HTTP(S) traffic and TLS-enabled traffic via SNI. The router's container listens on the host network interface, unlike most containers that listen only on private IPs. The router proxies external requests for route names to the IPs of actual pods identified by the service associated with the route.

- The *F5* router integrates with an existing F5 BIG-IP® system in your environment to synchronize routes. F5 BIG-IP® version 11.4 or newer is required in order to have the F5 iControl REST API.

In our lab environment, we'll deploy the default HAProxy template router. Similar to the `oadm registry` command that enables us to deploy a registry on the cluster, there is an `oadm router` command that we can use here.

2.4.1 The Router Service Account

Before deploying an OpenShift cluster, you must have a service account for the router. However, in OpenShift Enterprise 3.1 and above, a router service account is automatically created during a quick or advanced installation (previously, this required manual creation). This service account, called `router`, has permissions to a security context constraint (SCC) that allows it to specify host ports.

To verify that the router service account has been created, you can do a dry-run naming the 'router' account in the command:

```
[root@ose3-master ~]# oadm router --dry-run \
--credentials='/etc/origin/master/openshift-router.kubeconfig' \
--service-account=router
```

This will exit with an error like:

```
error: router "router" does not exist (no service)
```

That's fine; the credentials worked but there is no deployed router yet. Head on to the next section.

2.4.2 Router Deployment

Having confirmed the existence of the router account, let's proceed with deployment. Multiple instances of the router can be deployed, but in our lab environment we'll deploy a single instance with the name 'router'. To do so, run the following command:

```
[root@ose3-master ~]# oadm router router --replicas=1 \
```

```
--credentials='/etc/origin/master/openshift-router.kubeconfig' \
--service-account=router
```

You should see output like:

```
password for stats user admin has been set to SDVZgCzySU
DeploymentConfig "router" created
Service "router" created
```

And oc get pods should reveal (eventually) a router pod in ‘Running’ status:

```
[root@ose3-master ~]# oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-kvfls    1/1       Running   0          1h
router-1-ajbs1             1/1       Running   0          3m
```

At this point, if you run:

```
[root@ose3-master ~]# oadm router router --dry-run
```

You should see:

```
Router "router" service exists
```

Looking good? On to the next section.

2.4 Managing Projects

Next let's look at some of the ways that administrators can manage user projects. *Projects* are the namespaces in which users define and deploy cluster assets (pods, services, etc.).

2.5.1 Creating Projects

To start, we'll create a number of projects using the `oadm new-project` command:

```
export APPNAME=portalapp
export APPTEXT="Portal App"
oadm new-project ${APPNAME}-dev --display-name="${APPTEXT} Development"
oadm new-project ${APPNAME}-test --display-name="${APPTEXT} Testing"
oadm new-project ${APPNAME}-prod --display-name="${APPTEXT} Production"

export APPNAME=paymentapp
export APPTEXT="Payment App"
oadm new-project ${APPNAME}-dev --display-name="${APPTEXT} Development"
oadm new-project ${APPNAME}-test --display-name="${APPTEXT} Testing"
oadm new-project ${APPNAME}-prod --display-name="${APPTEXT} Production"
```

When you have created these, you can confirm them by running:

```
[root@ose3-master ~]# oc get projects
```

You should see output like:

```
NAME               DISPLAY NAME              STATUS
management-infra                             Active
openshift                                    Active
openshift-infra                              Active
paymentapp-dev     Payment App Development   Active
portalapp-dev      Portal App Development    Active
portalapp-test     Portal App Testing        Active
paymentapp-prod    Payment App Production    Active
paymentapp-test    Payment App Testing       Active
portalapp-prod     Portal App Production     Active
```

2.5.2 Setting Group Access Through Policies

Right now, if you were to login in as the payment1 user, you would have no projects associated with your account (as a reminder, the password is ‘r3dh4t1!’):

```
[root@ose3-master ~]# oc login -u payment1
Authentication required for https://ose3-master.example.com:8443 (openshift)
Username: payment1
Password:
Login successful.

You don't have any projects. You can try to create a new project, by running

   $ oc new-project <projectname>
```

In order to fix this, you will associate the `paymentapp` user group with a few of the paymentapp-* projects that we’ve just created.

As the admin1 user:

```
[root@ose3-master ~]# oc login -u admin1
```

Run the oadm policy command to grant the paymentapp user group with administrative rights on the paymentapp-dev and paymentapp-test projects:

```
oadm policy add-role-to-group admin paymentapp -n paymentapp-dev
oadm policy add-role-to-group admin paymentapp -n paymentapp-test
```

And while we’re at it, let’s do the same for the portalapp user group and the portalapp-dev and portalapp-test projects:

```
oadm policy add-role-to-group admin portalapp -n portalapp-dev
oadm policy add-role-to-group admin portalapp -n portalapp-test
```

And finally, let’s grant the ose-production group with admin priveleges on the production versions of those two applications:

```
oadm policy add-role-to-group admin ose-production -n portalapp-prod
oadm policy add-role-to-group admin ose-production -n paymentapp-prod
```

To check our work, we'll use the `oc describe` command on the per-project policy bindings. In the output of each of the following commands, look under the section titled "`RoleBinding[admin]`". You should see a "`Groups:`" line that contains the expected user group for each project.

```
[root@ose3-master ~]# oc describe policybindings :default -n paymentapp-dev
[root@ose3-master ~]# oc describe policybindings :default -n paymentapp-test
[root@ose3-master ~]# oc describe policybindings :default -n paymentapp-prod
[root@ose3-master ~]# oc describe policybindings :default -n portalapp-dev
[root@ose3-master ~]# oc describe policybindings :default -n portalapp-test
[root@ose3-master ~]# oc describe policybindings :default -n portalapp-prod
```

If everything looks good, head on to the next section.

2.5.3 Setting Quotas

At this point we are ready to apply some control over the amount of resources that will be available to the projects running on this cluster. Quotas describe how much of a resource is available across an entire project, while Limits describe the minimums and maximums for individual pods and containers.

The quota system in OpenShift is fairly granular and offers control over a number of resources:

- cpu
- memory
- pods
- replicationcontrollers
- resourcequotas
- services
- secrets
- persistentvolumeclaims

A few quick notes on this list:

- CPU units: CPU is measured in units called *millicores*. Each node in the cluster introspects the operating system to determine the amount of CPU cores on the node and then multiples that value by 1000 to express its total capacity. For example, if a node has 2 cores, the node's CPU capacity would be represented as 2000m. If you wanted to use a 1/10 of a single core, you would represent that as 100m.

- Memory: memory is measured in bytes. In addition, it may be used with SI suffices (E, P, T, G, M, K, m) or their power-of-two-equivalents (Ei, Pi, Ti, Gi, Mi, Ki).

- The CPU and memory quotas are measured across *all containers* running within a project. The rest of the resource quotas and limits are expressed as straight counts of those resources within the project.

To define and apply quotas to the non-production projects that we've created, we will first create a JSON file with our desired limits.

Create a file called `resource-quotas-non-prod.json` and save the following contents to it:

```
{
 "apiVersion": "v1",
 "kind": "ResourceQuota",
 "metadata": {
   "name": "quota"
 },
 "spec": {
   "hard": {
     "cpu": "400m",
     "memory": "200M",
     "pods": "2",
     "services": "1",
     "replicationcontrollers":"1",
     "resourcequotas":"1"
   }
 }
}
```

Any projects that this quota configuration is applied to will be limited to 2 pods that will share 400 total millicores of CPU and 200MB of memory. That's not very much, but this lab *is* running on a laptop with limited resources.

To apply the quotas to our non-production laptops, we'll run the `oc create` command:

```
[ose3-master ~]# oc create -f resource-quotas-non-prod.json -n portalapp-dev
[ose3-master ~]# oc create -f resource-quotas-non-prod.json -n portalapp-test
[ose3-master ~]# oc create -f resource-quotas-non-prod.json -n paymentapp-dev
[ose3-master ~]# oc create -f resource-quotas-non-prod.json -n paymentapp-test
```

And we can check to make sure that the quota was applied by running the `oc describe` command:

```
[root@ose3-master ~]# oc describe quota -n paymentapp-dev
Name:                   quota
Namespace:              paymentapp-dev
Resource                Used  Hard
--------                ----  ----
cpu                     0     400m
memory                  0     200M
pods                    0     2
replicationcontrollers  0     1
resourcequotas          1     1
services                0     1
```

Note that the resourcequotas limit of 1 is already used up. In order to apply additional resource quotas we would first need to remove the quota definition that was just applied.

2.5.4 Setting Limits

Similar to the manner in which we set quotas, we will begin to set limits by creating a file called `resource-limits-non-prod.yaml` and populate it as follows:

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
 name: "limits"
spec:
 limits:
   -
     type: "Pod"
     max:
       cpu: "400m"
       memory: "200M"
     min:
       cpu: "100m"
       memory: "50M"
   -
     type: "Container"
     max:
       cpu: "200m"
       memory: "100M"
     min:
       cpu: "50m"
       memory: "10M"
     default:
       cpu: "200m"
       memory: "100M"
     defaultRequest:
       cpu: "200m"
       memory: "100M"
     maxLimitRequestRatio:
       cpu: "4"
```

This Limits configuration establishes minimum, maximum and default quanitites of CPU and memory resources. If a quota has been defined for a given project, the quota values will overrule any limits set here. For instance, if a quota of 100 millicores of CPU was set for the entire project, then the 200 millicore default stated in this Limits configuration would be too high for a simple deployment.

Once you have created the limits template, you can apply it to the various projects with an `oc create` command:

```
[ose3-master ~]# oc create -f resource-limits-non-prod.yaml -n paymentapp-dev
[ose3-master ~]# oc create -f resource-limits-non-prod.yaml -n paymentapp-test
[ose3-master ~]# oc create -f resource-limits-non-prod.yaml -n portalapp-dev
[ose3-master ~]# oc create -f resource-limits-non-prod.yaml -n portalapp-test
```

And you can verify the limits settings with the `oc describe` command:

```
[root@ose3-master ~]# oc describe limits -n paymentapp-dev
Name:       limits
Namespace:  paymentapp-dev
Type        Resource    Min   Max   Request  Limit  Limit/Request
----        --------    ---   ---   -------  -----  -------------
Pod         memory      50M   200M  -        -      -
Pod         cpu         100m  400m  -        -      -
Container   memory      10M   100M  100M     100M   -
Container   cpu         50m   200m  200m     200m   4
```

2.5.5 Deploying Applications

As a final step, we will deploy a “Hello World” application into each of the projects using the `oc new-app` command.

When quotas are defined in a project, we must specify the needed amounts of any resources governed by the quota. However, because our limits definition included default request sizes for CPU and memory, we can skip the complexities of requesting these resources at creation time and simply run an `oc new-app` command for each project:

```
[~]# oc new-app docker.io/openshift/hello-openshift:v1.1.6 -n paymentapp-dev
[~]# oc new-app docker.io/openshift/hello-openshift:v1.1.6 -n paymentapp-test
[~]# oc new-app docker.io/openshift/hello-openshift:v1.1.6 -n portalapp-dev
[~]# oc new-app docker.io/openshift/hello-openshift:v1.1.6 -n portalapp-test
```

Each should provide output similar to:

```
--> Found Docker image 4cc2d04 (6 weeks old) from Docker Hub for
"openshift/hello-openshift:v1.1.6"
   * An image stream will be created as "hello-openshift:v1.1.6" that will track
this image
   * This image will be deployed in deployment config "hello-openshift"
   * [WARNING] Image "hello-openshift" runs as the 'root' user which may not be
permitted by your cluster administrator
   * Ports 8080/tcp, 8888/tcp will be load balanced by service "hello-openshift"
--> Creating resources with label app=hello-openshift ...
   ImageStream "hello-openshift" created
   DeploymentConfig "hello-openshift" created
   Service "hello-openshift" created
--> Success
   Run 'oc status' to view your app.
```

After a few moments, you can check on each project's status with the `oc status` command:

```
[root@ose3-master ~]# oc status -n paymentapp-dev
In project Payment App Development (paymentapp-dev) on server
https://ose3-master.example.com:8443

svc/hello-openshift - 172.30.56.197 ports 8080, 8888
 dc/hello-openshift deploys imagestreamtag/hello-openshift:v1.1.6
   #1 deployed 43 seconds ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc get
all'.
```

Finally, you can test that the app is available by curling one of the svc/hello-openshift IP and port combinations. Note that the exact IP addresses used may be different in your lab environment:

```
[root@ose3-master ~]# curl 172.30.56.197:8888
Hello OpenShift!
```

Now that we've got apps running in the cluster, on to the next lab!

Lab 3: Node Management

So far we've focused almost entirely on user and group administration tasks, along with some basic project management exercises. In this lab, we move our focus to managing the cluster itself.

To begin, let's have a look at what we've deployed so far. Run the following command to see all of the pods currently deployed under any project on the cluster:

```
[root@ose3-master ~]# oc get pods --all-namespaces -o wide
```

The complete output is too wide to show on this page, but here are the fields of interest:

```
NAMESPACE         NAME                       NODE
default           docker-registry-1-7l4p8    ose3-master.example.com
default           router-1-iz4br             ose3-master.example.com
paymentapp-dev    hello-openshift-1-ilz6t    ose3-node2.example.com
paymentapp-test   hello-openshift-1-7ax8d    ose3-node2.example.com
portalapp-dev     hello-openshift-1-4lw82    ose3-node1.example.com
portalapp-test    hello-openshift-1-p9pox    ose3-node1.example.com
```

The exact names and node locations will vary in your deployment. Note that the registry and router pods, which are part of the 'default' namespace, both ended up on the master, while the project pods that we've deployed have been evenly distributed across all available node hosts. That is due to the work we did in setting up a default node selector for apps ('env=usr', which is associated the the node1 & node2 node hosts) and then overriding that default for cluster assets (mapping the 'default' namespace to 'env=infra', which is assocated with the master node host).

3.1 Reducing the Cluster Size

Now let's walk through a very common cluster management scenario: migrating pods away from a Node so that we can take it out of the cluster.

3.1.1 Node Maintenance Mode

The first step in preparing to take a node offline is to ensure that new pods are not deployed to it. We accomplish this by flagging the node as *unschedulable*.

To see the current state of the nodes, run:

```
[root@ose3-master ~]# oc get nodes
```

You should see output like:

```
NAME                      LABELS          STATUS    AGE
ose3-master.example.com   env=infra,...   Ready     3h
ose3-node1.example.com    env=user,...    Ready     3h
ose3-node2.example.com    env=user,...    Ready     3h
```

The status of all three node hosts is 'ready', meaning they are available to take on more containers.

Now we are going to change Node 2 to be unschedulable by running the following command:

```
[ose3-master ~]# oadm manage-node ose3-node2.example.com --schedulable=false
```

If you rerun the `oc get nodes` command, the change should be reflected there:

```
NAME                      LABELS          STATUS                      AGE
ose3-master.example.com   env=infra,...   Ready                       3h
ose3-node1.example.com    env=user,...    Ready                       3h
ose3-node2.example.com    env=user,...    Ready,SchedulingDisabled    3h
```

Now, to see only those pods that are running on Node 2 (and to confirm that they are still running even though the Node was marked as unschedulable), run the following:

```
[root@ose3-master ~]# oadm manage-node ose3-node2.example.com --list-pods
```

You should see the pods from the beginning of the exercise that were deployed on Node 2:

```
Listing matched pods on node: ose3-node2.example.com

NAME                      READY     STATUS    RESTARTS   AGE
hello-openshift-1-dlljm   1/1       Running   0          27m
hello-openshift-1-iaofb   1/1       Running   0          27m
```

On to the next step...

3.1.2 Migrating Containers

With Node 2 marked as unschedulable, now it is time to move the pods that are running there to other nodes on the cluster. This process is called *evacuation*. To perform an evacuation, we run another `oadm manage-node` command.

The `--dry-run` version of the command enables us to preview and confirm the results before making changes to the cluster:

```
[ose3-master ~]# oadm manage-node ose3-node2.example.com --evacuate --dry-run

Listing matched pods on node: ose3-node2.example.com

NAME                      READY     STATUS    RESTARTS   AGE
hello-openshift-1-dlljm   1/1       Running   0          32m
hello-openshift-1-iaofb   1/1       Running   0          31m
```

And then we can run it again without the `--dry-run` flag for effect:

```
[root@ose3-master ~]# oadm manage-node ose3-node2.example.com --evacuate

Migrating these pods on node: ose3-node2.example.com

NAME                      READY     STATUS    RESTARTS   AGE
hello-openshift-1-dlljm   1/1       Running   0          34m
hello-openshift-1-iaofb   1/1       Running   0          34m
```

After a moment, we can confirm that the pods were redistributed by rerunning the `oc get pods` command (again, the output has been limited to the fields of interest):

```
[root@ose3-master ~]# oc get pods --all-namespaces -o wide
NAMESPACE         NAME                      NODE
default           docker-registry-1-7l4p8   ose3-master.example.com
default           router-1-iz4br            ose3-master.example.com
paymentapp-dev    hello-openshift-1-88i53   ose3-node1.example.com
paymentapp-test   hello-openshift-1-ywou3   ose3-node1.example.com
portalapp-dev     hello-openshift-1-4lw82   ose3-node1.example.com
portalapp-test    hello-openshift-1-p9pox   ose3-node1.example.com
```

At this point, we are ready for the final step, which would be to delete the Node 2 host from the list of nodes known to the system. However! *We're not going to do that in this lab environment*. Adding a new node, or re-adding one that has been deleted, would require us to rerun the Ansible playbook. That runs a risk of failure in a network-isolated place like our lab environment.

That said, we've gone far enough along that we can now look at reversing the process to increase the cluster size.

3.2 Increasing the Cluster Size

In a normal deployment, we would begin the process of adding a node by entering information about it into the Ansible manifest file at `/etc/ansible/hosts` and the rerunning our playbook. However, in this lab, Node 2 is already known to the cluster.

In our lab environment, we'll bring Node 2 back into the fold by making it schedulable again:

```
[ose3-master ~]# oadm manage-node ose3-node2.example.com --schedulable=true
```

We confirm that the node is now schedulable with `oc get nodes`:

```
[root@ose3-master ~]# oc get nodes
NAME                      LABELS          STATUS    AGE
ose3-master.example.com   env=infra,...   Ready     1h
ose3-node1.example.com    env=user,...    Ready     1h
ose3-node2.example.com    env=user,...    Ready     1h
```

This is the same state that we would be in if we were to rerun the Ansible playbook in a normal (non-lab) environment, after adding an additional node to the `/etc/ansible/hosts` file.

3.3 Super-Secret Bonus Exercise

How did you find this carefully concealed bonus exercise? No matter - we're going to play a mini-game called "whack-a-pod".

Since you've evacuated and then re-added Node 2, have a look again at the pods running on this cluster:

```
[root@ose3-master ~]# oc get pods --all-namespaces -o wide
NAMESPACE         NAME                       NODE
default           docker-registry-1-7l4p8    ose3-master.example.com
default           router-1-iz4br             ose3-master.example.com
paymentapp-dev    hello-openshift-1-88i53    ose3-node1.example.com
paymentapp-test   hello-openshift-1-ywou3    ose3-node1.example.com
portalapp-dev     hello-openshift-1-4lw82    ose3-node1.example.com
portalapp-test    hello-openshift-1-p9pox    ose3-node1.example.com
```

They are all still running on Node 1! But why didn't they get evenly distributed between Node 1 and Node 2? Because Node 1 is not overloaded, the cluster is not going to automatically redistribute pods to other hosts. However, if any of these pods fails, crashes, or otherwise exits out, the cluster will detect that and launch a new copy. At that point, pods will start ending up on Node 2 again.

So, on to the game:

Cluster administrators can manually delete pods on the cluster with the command:

```
[root@ose3-master ~]# oc delete pod/<pod_name> -n <project_name>
```

Try it with the pod associated with the paymentapp-dev project and then re-list the pods:

```
[root@ose3-master ~]# oc get pods --all-namespaces -o wide
NAMESPACE         NAME                       NODE
```

```
default           docker-registry-1-lq17i   ose3-master.example.com
default           router-1-9h34w            ose3-master.example.com
paymentapp-dev    hello-openshift-1-njjyd   ose3-node2.example.com
paymentapp-test   hello-openshift-1-fllav   ose3-node1.example.com
portalapp-dev     hello-openshift-1-ozble   ose3-node1.example.com
portalapp-test    hello-openshift-1-5ej70   ose3-node1.example.com
```

A couple of points here:

- Surprise! You thought you deleted the pod, but it's back with a different name.
- There's no guarantee that it will end up on Node 2, but it is *likely* to do so.

You can keep trying this with other pods on Node 1. Eventually, you *may* only have pods running on Node 2, or they end up distributed between the two Nodes.

What's important to understand is the "self healing" nature of applications running on OpenShift. When you create applications with `oc new-app`, OpenShift creates a *replication controller* (RC) on the cluster, and it is the RC's job to make sure that your requested number of instances is always available.

For more on that, check out some of the great developer materials available for people who want to deploy apps on OpenShift.

Lab 4: OpenShift UI

Up to this point we've been focused entirely on what you can accomplish using the command line tools. Now let's take a tour of the web console.

Start by opening this URL in your browser:

https://ose3-master.example.com:8443/

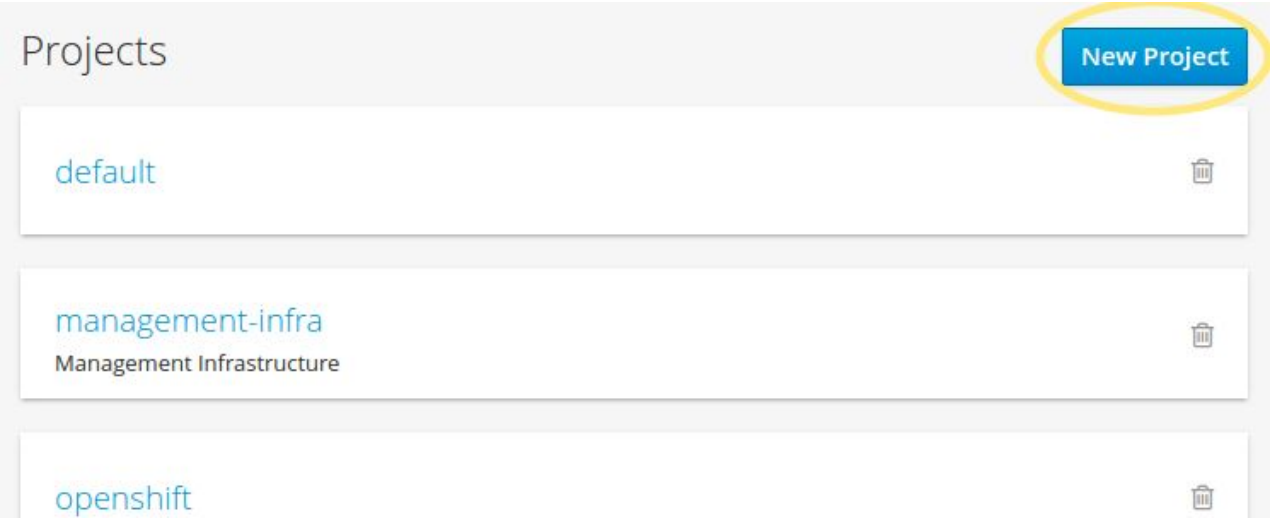
(Sometimes that won't work in a lab environment where DNS is not working, so you can also try https://192.168.133.2:8443/)

And log in with the admin1 credentials we've been using: admin1 / r3dh4t1!

You should see a list of all of the projects that you've created, plus the built-in projects that all OpenShift deployments contain.

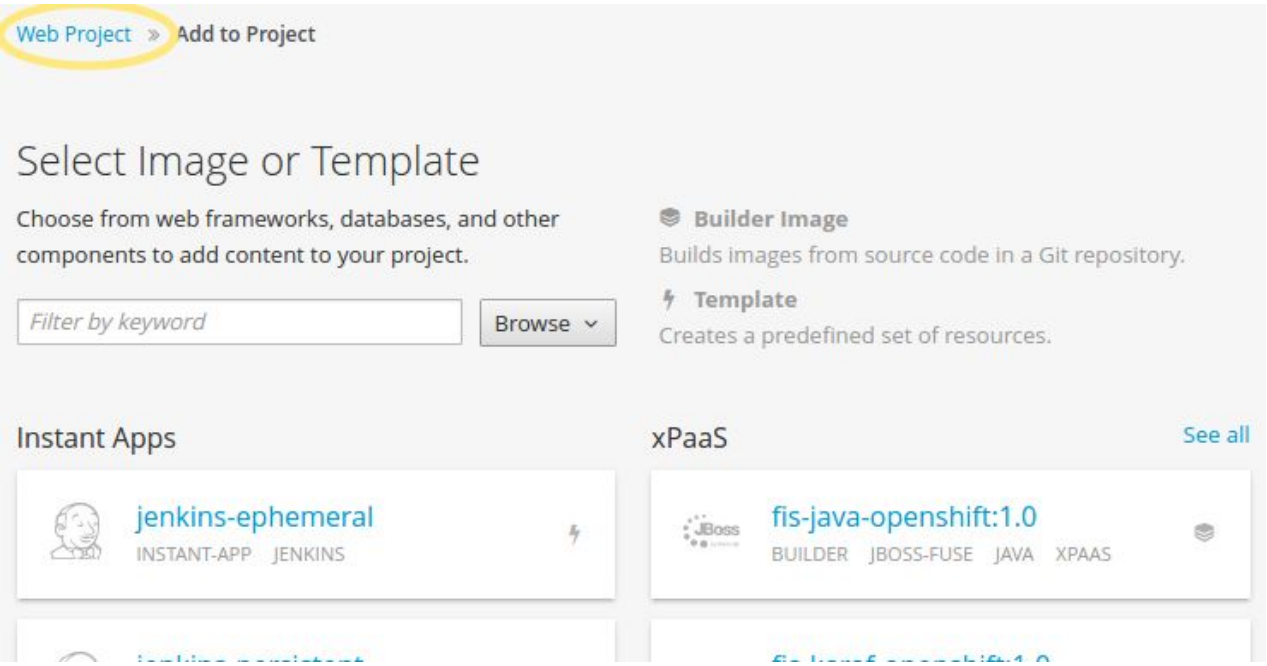
4.1 Creating a New Project

From the UI, create a new project namespace by pressing the blue New Project button at the top of the Projects list:

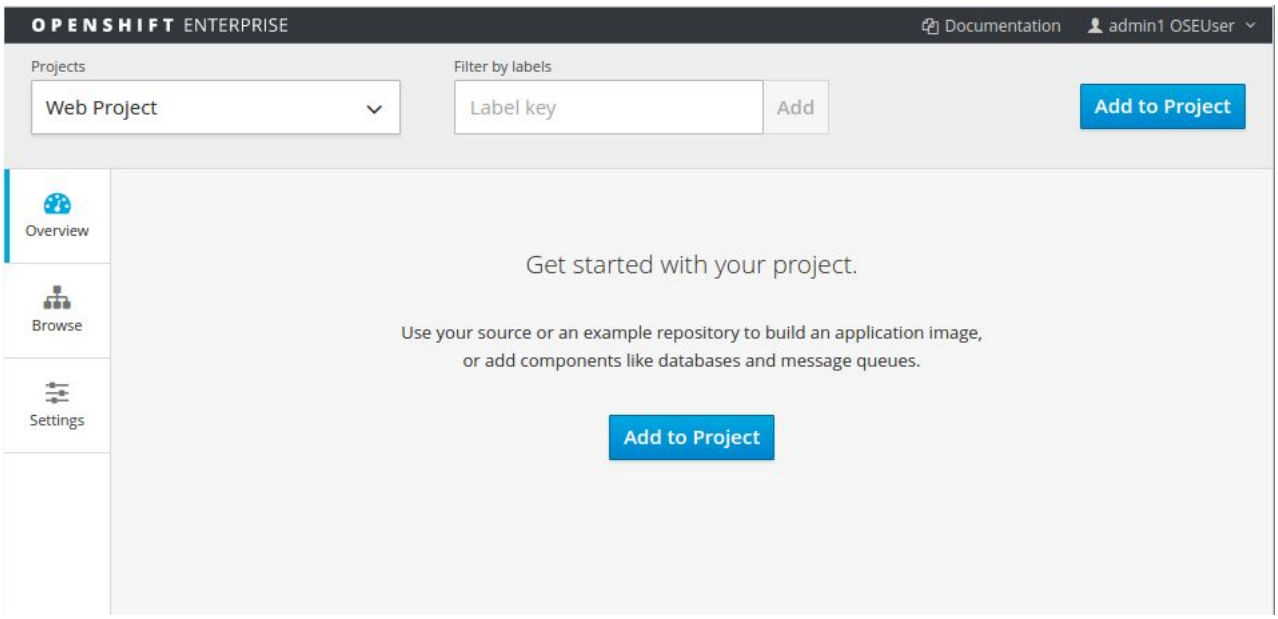


Next, give the project a system name and a display name, like web-project / Web Project:

On the next page, you will see a wide variety of options for starting off your application. Unfortunately, we won't get very far without access to the internet, but we can still build out a simple application with images that are already available in our lab environment. With that in mind, instead of selecting one of the images that are listed, head to the top left corner of the page and click on the name of your project:



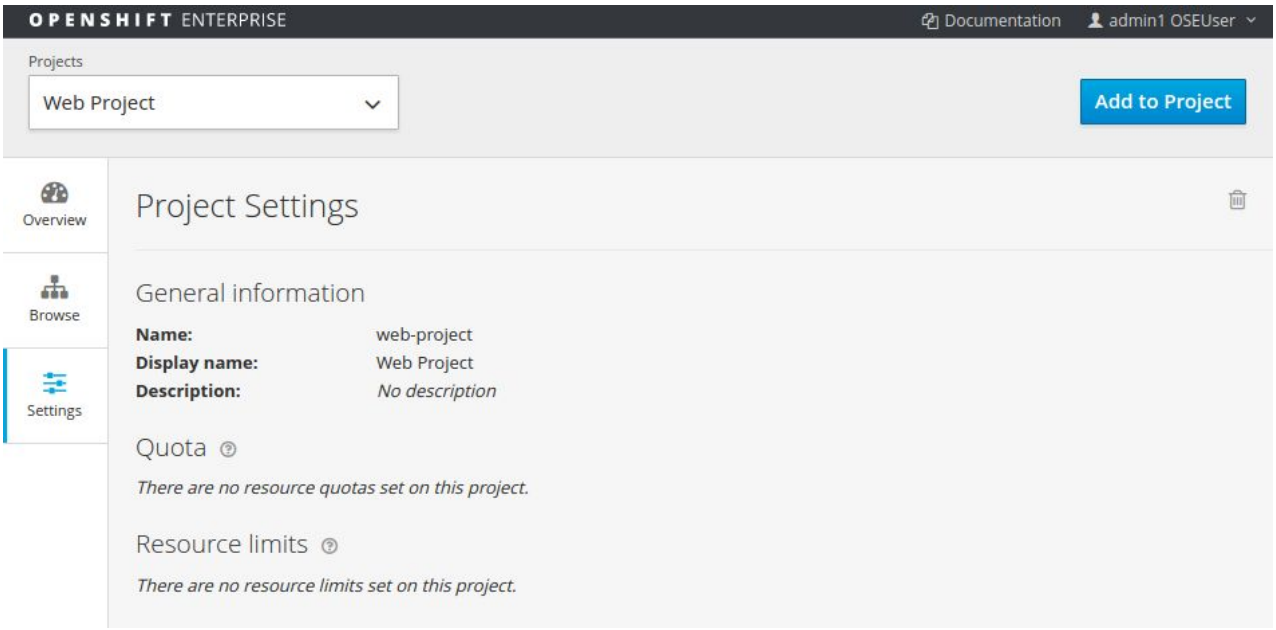
You should see an empty project summary space like this:



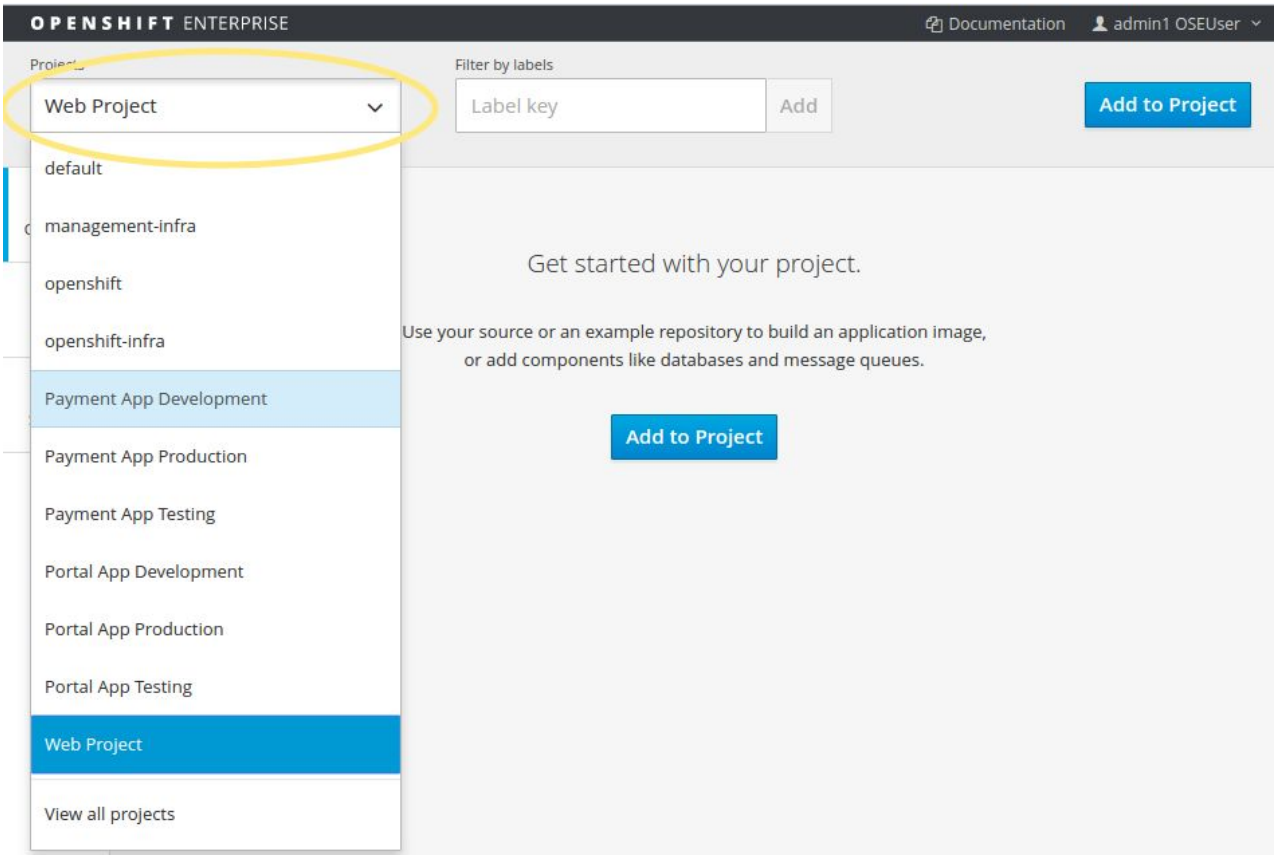
At this point you could either proceed from the command line or (network allowing) the internet to continue building out your app.

4.2 Quota and Limits

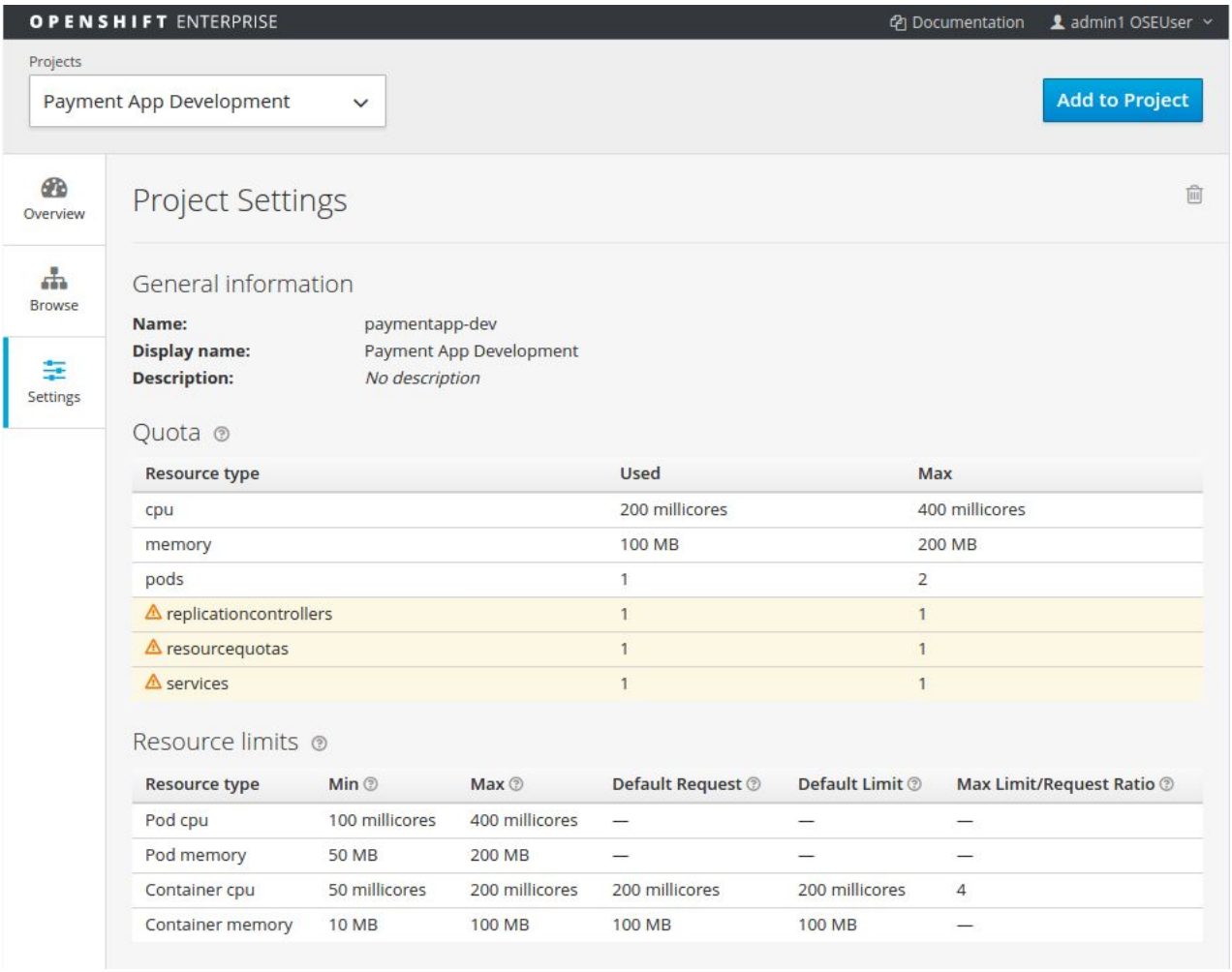
On the left side of the project Overview page, click on the Settings tab. Note that the project that you've created has no associated quotas or limits:


OPENSHIFT ENTERPRISE
Documentation
admin1 OSEUser
Projects
Web Project
Add to Project
Overview
Browse
Settings
Project Settings
General information
Name:
web-project
Display name:
Web Project
Description:
No description
Quota
There are no resource quotas set on this project.
Resource limits
There are no resource limits set on this project.

Now, from the Projects selection list at the top left of the page, select one of the projects that you created earlier, like the Payment App Development project:



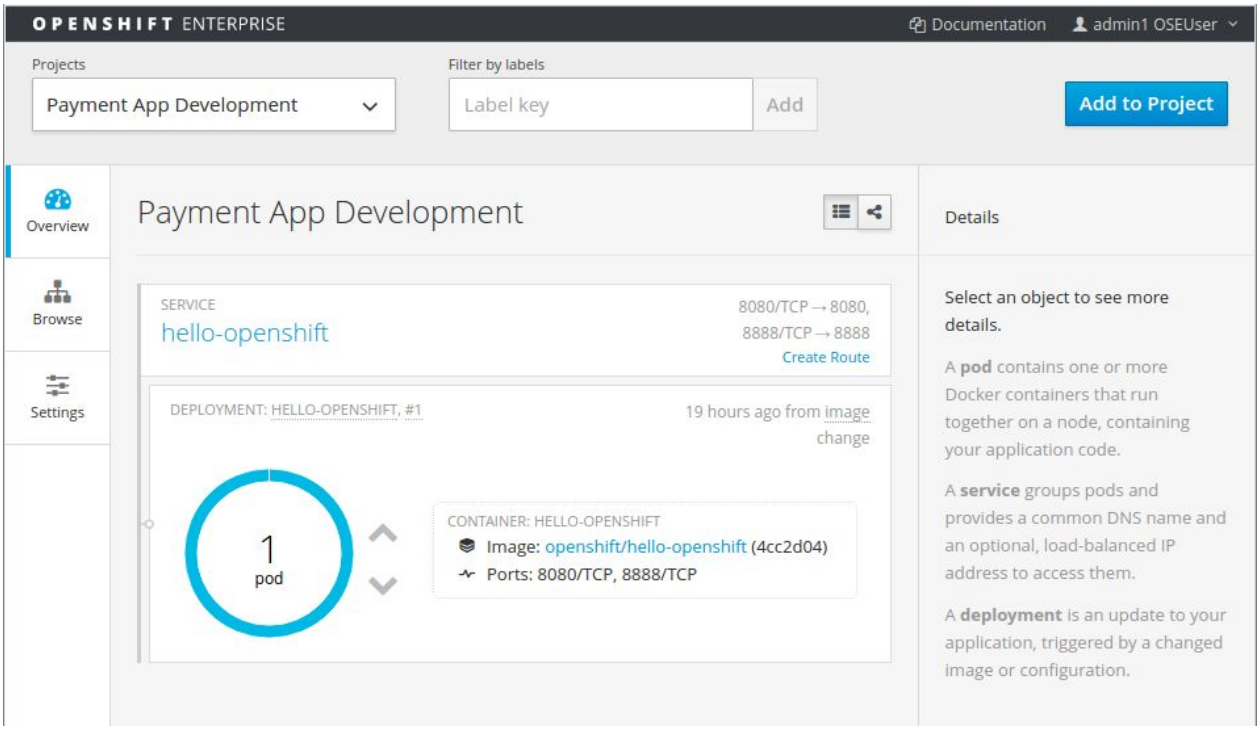
Click on the Settings tab again to see the quotas and limits that you applied there:



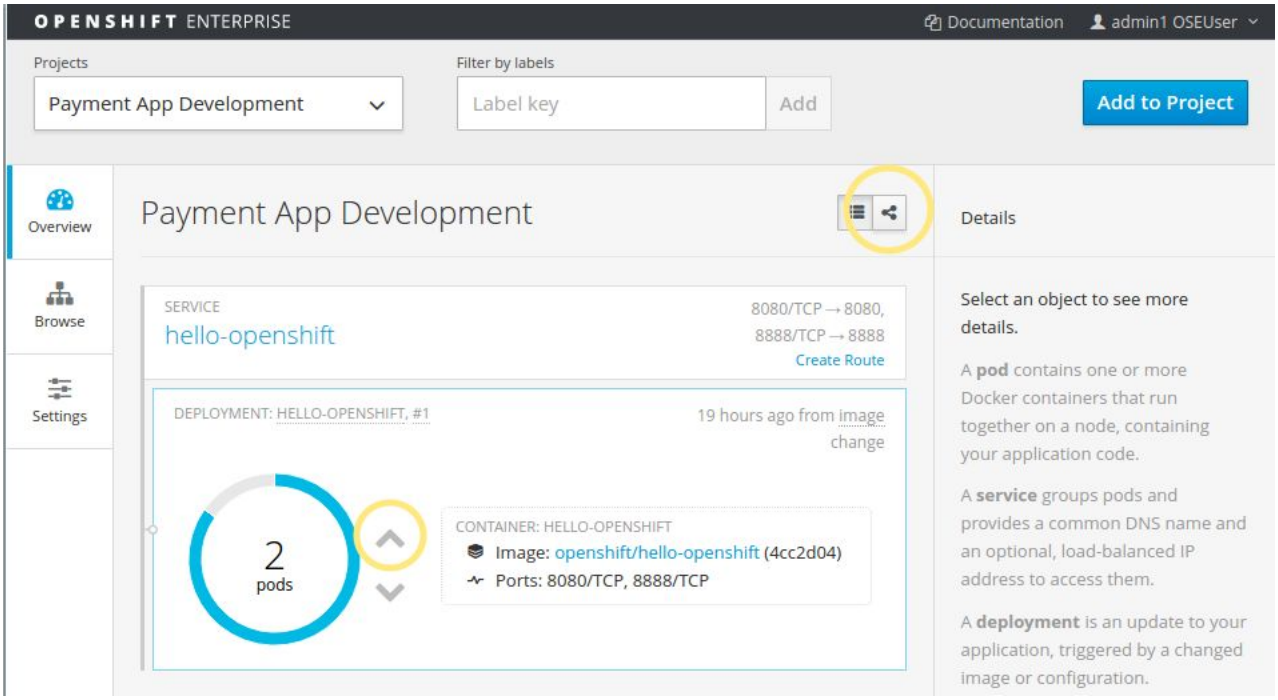
4.3 Scaling Applications

From the settings page, you can see that the application is running at half of its allowed capacity. This means that there's enough room for us to manually scale the app.

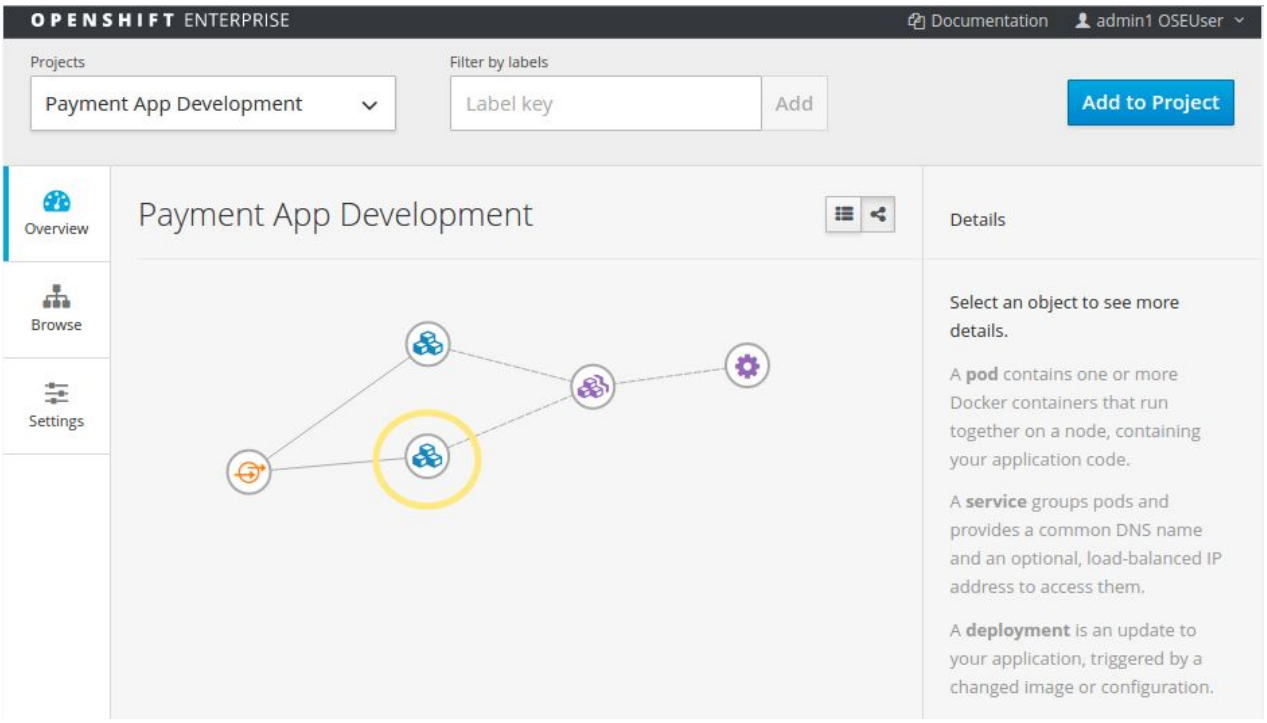
Head back to the Overview page, where you can see the Service, Deployment, and Container that make up this application:



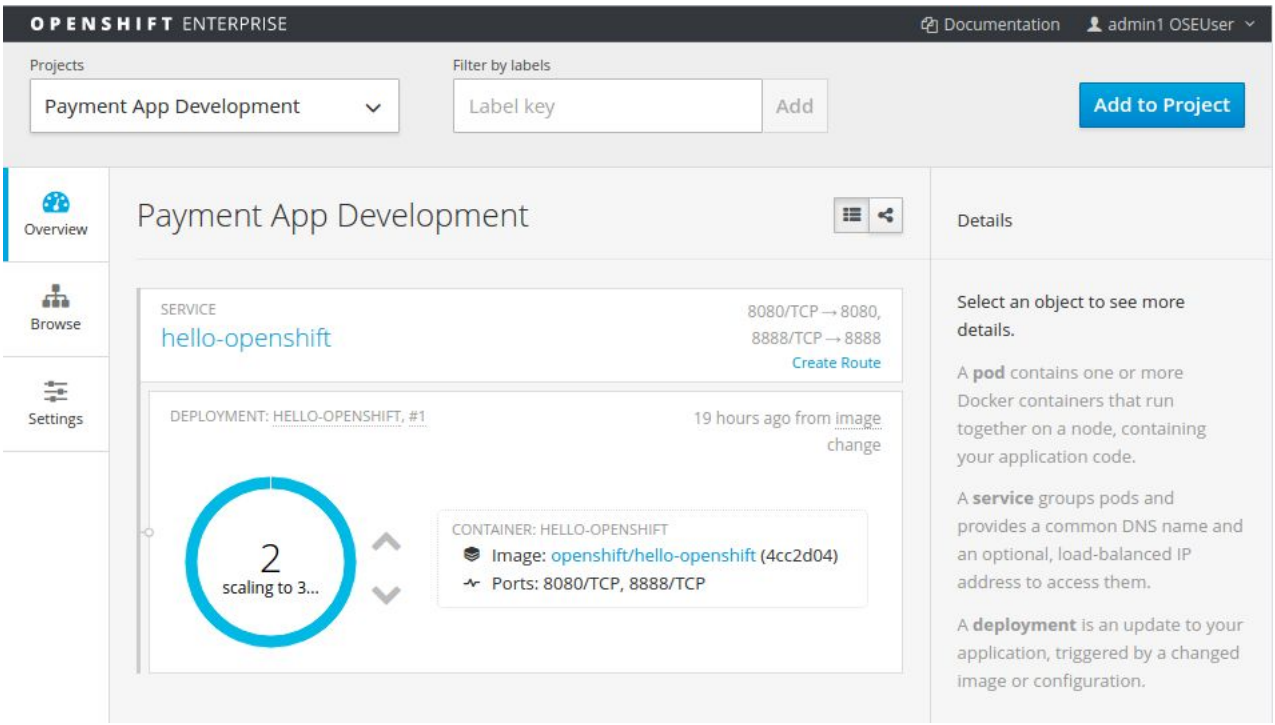
Next to the blue circle indicating the current pod count, press the upward pointing arrow to manually scale the application, and then click on the Topology View tab to see how this affects the application:



You should see a second instance of the hello-world pod pop into existence:

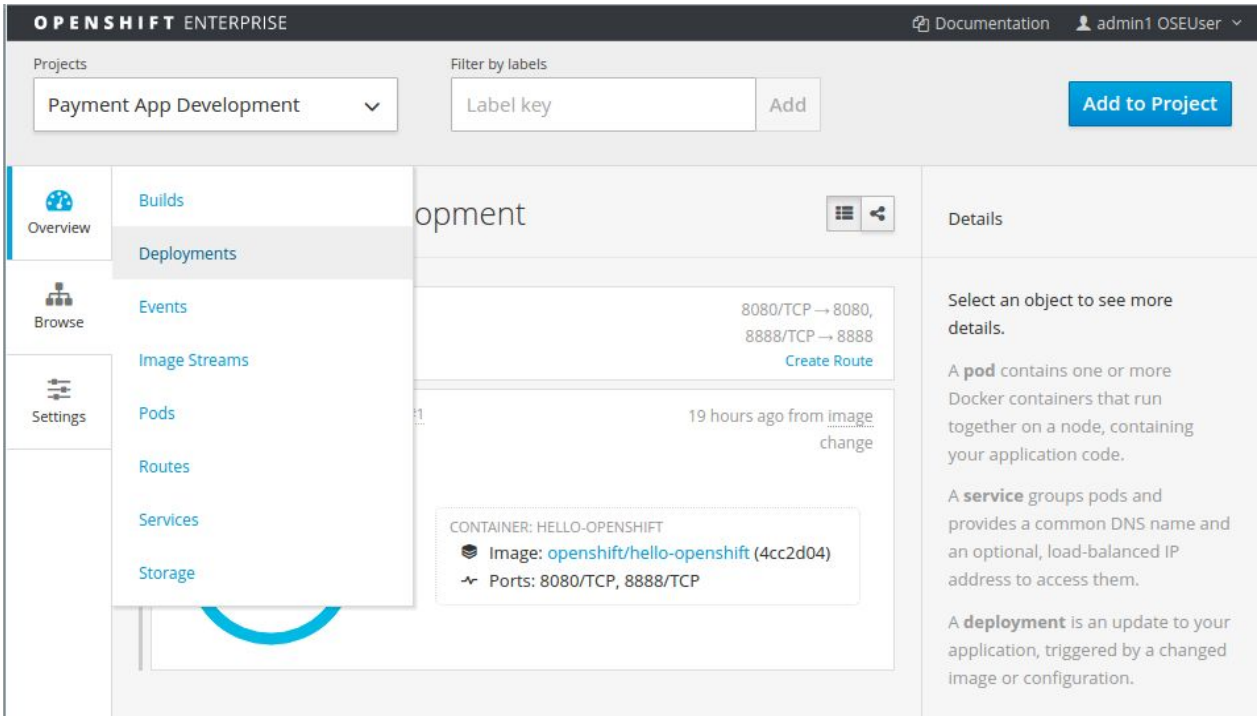


However, if you pop back to the Tile View and attempt to scale the app to 3 pods, it will never succeed. This is our limits definition in action:



4.4 Other Details

When you mouse over the Browse tab on the left side of the screen, you will see several sub-items:



These are the various objects that make up your application. Click through to see which items were created for you when you ran the `oc new-app` command.

OpenShift for Operators: Lab Summary

Over the course of this lab we:

- Set up an Ansible configuration to deploy an OpenShift cluster
- Configured the cluster to use an external IDM system for user authentication
- Mapped groups from the IDM system to roles and projects within the cluster
- Created projects and applications
- Set quotas and limits
- Shrunk and grew the cluster
- Took a tour of the OpenShift web console

But in two hours, we've really only scratched the surface of what is possible with OpenShift.

Where to Learn More

There are several resources available to you:

- Red Hat Training offers a course for OpenShift Administration
- OpenShift Enterprise documentation is available online
- OpenShift gurus hang out on the #openshift channel on FreeNode
- You can join and ask questions on the OpenShift Users mailing list

And if you are interested in development resources, you can also check out developers.openshift.com!

Thank You!

We really appreciate your interest in this lab! Loved it? Hated it? Either way, please leave your feedback so that we can keep improving our training offerings.