Table of Contents

Resource Management Lab

In this lab, you learn how to manage OpenShift Enterprise resources.

- **Manage Users, Projects, and Quotas**

  In this section, you create projects and test the use of quotas and limits.

- **Create Services and Routes**

  In this section, you manually create services and routes for pods and review the changes to a service when scaling an application.

- **Explore Containers**

  In this section, you run commands within active pods and explore the `docker-registry` and `Default Router` containers.

- **Create Persistent Volume for Registry**

  In this section, you create a persistent volume for your registry, attach it to `deploymentConfiguration`, and redeploy the registry.

1. Manage Users, Projects, and Quotas

1.1. Create Project

1. Connect to the `master00` host:

   ```
   [root@oselab-GUID ~]# ssh master00-$guid
   ```

2. On the master host, run `oadm` to create and assign the administrative user `andrew` to a project:

   ```
   [root@master00-GUID ~]# oadm new-project resourcemanagement --display-
   name="Resources Management" \
       --description="This is the project we use to learn about resource management" \
       --admin=andrew  --node-selector='region=primary'
   ```

   `andrew` can create his own project with the `oc new-project` command, an option you will experiment with later in this course. Note that defining the `--node-selector` is optional, especially since you already defined the default `node-selector` in a previous lab.

1.2. View Resources in Web Console

Now have a look at the web console, which has been completely redesigned for version 3.

1. Open your web browser and go to `https://master00-GUID.oslab.opentlc.com:8443`.

   The web console could take up to 90 seconds to become available after a restart of the master.

   The first time you access the URL, you might need to accept the self-signed SSL certificate.

2. When prompted, type the username and password, as follows:
   - **Username**: `andrew`
   - **Password**: `r3dh4t1!`

3. In the web console, click the **Resources Management** project.

   The project is empty because it has no data. You enter data in this part of the lab.

1.3. Apply Quota to Project

1. Create a quota definition file:

   ```
   [root@master00-GUID ~]#  cat << EOF > quota.json
   {
   ```

```
  "apiVersion": "v1",
  "kind": "ResourceQuota",
  "metadata": {
    "name": "test-quota"
  },
  "spec": {
    "hard": {
      "memory": "512Mi",
      "cpu": "20",
      "pods": "3",
      "services": "5",
      "replicationcontrollers":"5",
      "resourcequotas":"1"
    }
  }
}
EOF
```

2. On the master host, do the following:

   a. Run `oc create` to apply the file you just created:

   ```
   [root@master00-GUID ~]# oc create -f quota.json --namespace=resourcemanagement
   ```

   b. Verify that the quota exists:

   ```
   [root@master00-GUID ~]# oc get -n resourcemanagement quota
   ```

   ```
   NAME         AGE
   test-quota   8s
   ```

   c. Verify the limits and examine the usage:

   ```
   [root@master00-GUID ~]# oc describe quota test-quota -n resourcemanagement
   ```

   ```
   Name:   test-quota
   Namespace:  resourcemanagement
   Resource  Used Hard
   --------  ---- ----
   cpu   0 20
   memory   0 512Mi
   pods   0 3
   replicationcontrollers 0 5
   resourcequotas  1 1
   services  0 5
   ```

3. On the web console, click the **Resource Management** project.
4. Click the **Settings** tab for information on the quota.

1.4. Apply Limit Ranges to Project

For quotas to be effective, you must create *limit ranges*. They allocate the maximum, minimum, and default memory and CPU at both the pod and container level. Absent defaults for containers, projects with quotas fail because the deployer and other infrastructure pods are unbounded and, therefore, forbidden.

1. Create the `limits.json` file:

```
[root@master00-GUID ~]# cat << EOF > limits.json
{
    "kind": "LimitRange",
    "apiVersion": "v1",
    "metadata": {
        "name": "limits",
        "creationTimestamp": null
    },
    "spec": {
        "limits": [
            {
                "type": "Pod",
                "max": {
                    "cpu": "500m",
                    "memory": "750Mi"
                },
                "min": {
                    "cpu": "10m",
                    "memory": "5Mi"
                }
            },
            {
                "type": "Container",
                "max": {
                    "cpu": "500m",
                    "memory": "750Mi"
                },
                "min": {
                    "cpu": "10m",
                    "memory": "5Mi"
                },
                "default": {
                    "cpu": "100m",
                    "memory": "100Mi"
                }
            }
        ]
    }
}
```

```
EOF
```

2. On the master host, run `oc create` against the `limits.json` file and the `resourcemanagement` project:

   ```
   [root@master00-GUID ~]# oc create -f limits.json --namespace=resourcemanagement
   ```

3. Review your limit ranges:

   ```
   [root@master00-GUID ~]# oc describe limitranges limits -n resourcemanagement
   ```

   ```
   Name:  limits
   Namespace: resourcemanagement
   Type  Resource Min Max Request Limit Limit/Request
   ----  -------- --- --- ------- ----- -------------
   Pod  memory  5Mi 750Mi - - -
   Pod  cpu  10m 500m - - -
   Container memory  5Mi 750Mi 100Mi 100Mi -
   Container cpu  10m 500m 100m 100m -
   ```

1.5. Test Quotas

> You are running commands as the Linux users `andrew` and `root` in a lab environment. In a real-word scenario, users, would, of course, issue `oc` commands from their workstations and not from the OpenShift Master.

1. Authenticate to OpenShift Enterprise and choose your project:

   a. Connect to the OpenShift Enterprise master according to the procedure you followed previously.

   b. When prompted, type the username and password:

      - **Username**: `andrew`
      - **Password**: `r3dh4t1!`

      ```
      [root@master00-GUID ~]# su - andrew
      [andrew@master00-GUID ~]$ oc login -u andrew --insecure-skip-tls-verify --
      server=https://master00-${guid}.oslab.opentlc.com:8443
      ```

      - The output is as follows:

        ```
        Login successful.
        ```

```
Using project "resourcemanagement".
Welcome! See 'oc help' to get started.
```

This lab shows you the manual, step-by-step method of creating each object. There are easier ways to create a deployment and its components. One of those ways is the `oc new-app` command, which is covered later in this lab.

2. Create the `hello-pod.json` pod definition file:

```
[andrew@master00-GUID ~]$ cat <<EOF > hello-pod.json
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "hello-openshift",
    "creationTimestamp": null,
    "labels": {
      "name": "hello-openshift"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "hello-openshift",
        "image": "openshift/hello-openshift:v1.0.6",
        "ports": [
          {
            "containerPort": 8080,
            "protocol": "TCP"
          }
        ],
        "resources": {
        },
        "terminationMessagePath": "/dev/termination-log",
        "imagePullPolicy": "IfNotPresent",
        "capabilities": {},
        "securityContext": {
          "capabilities": {},
          "privileged": false
        }
      }
    ],
    "restartPolicy": "Always",
    "dnsPolicy": "ClusterFirst",
    "serviceAccount": ""
  },
  "status": {}
}

EOF
```

1.6. Run Pod

Here, you create a simple pod without a *route* or *service*:

1. Create and verify the `hello-openshift` pod:

```
[andrew@master00-GUID ~]$ oc create -f hello-pod.json
pods/hello-openshift

[andrew@master00-GUID ~]$ oc get pods
NAME              READY     STATUS    RESTARTS   AGE
hello-openshift   1/1       Running   0          8s
```

2. Run `oc describe` for details on your pod:

```
[andrew@master00-GUID ~]$ oc describe pod hello-openshift
Name:    hello-openshift
Namespace:   resourcemanagement
Image(s):   openshift/hello-openshift:v1.0.6
Node:    node00-GUID.oslab.opentlc.com/192.168.0.200
Start Time:   Thu, 26 Nov 2015 21:23:27 -0500
Labels:    name=hello-openshift
Status:    Running
Reason:
Message:
IP:    10.1.2.2
Replication Controllers: <none>
Containers:
  hello-openshift:
    Container ID:
docker://e36321aabeb1cb64e3da054128818dedd8ec3891dbf8aa758c72a96fc1180eee
    Image:  openshift/hello-openshift:v1.0.6
    Image ID:
docker://bba2117915baabfd05932dc916306bae2c51d15848592c3018e7af0308dee519
    QoS Tier:
      cpu: Guaranteed
      memory: Guaranteed
    Limits:
      cpu: 100m
      memory: 100Mi
    Requests:
      cpu:  100m
      memory:  100Mi
    State:  Running
      Started:  Thu, 26 Nov 2015 21:23:32 -0500
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
  default-token-rnadp:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-rnadp
Events:
  FirstSeen LastSeen Count From     SubobjectPath  Reason  Message
  --------- -------- ----- ----     -------------  ------  -------
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} implicitly required container
POD Pulled  Container image "openshift3/ose-pod:v3.1.0.4" already present on
machine
  4m  4m  1 {scheduler }       Scheduled Successfully assigned hello-openshift to
node00-GUID.oslab.opentlc.com
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} implicitly required container
POD Created  Created with docker id f19fdc8fb3c8
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} implicitly required container
POD Started  Started with docker id f19fdc8fb3c8
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} spec.containers{hello-
openshift} Pulled  Container image "openshift/hello-openshift:v1.0.6" already
present on machine
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} spec.containers{hello-
openshift} Created  Created with docker id e36321aabeb1
  4m  4m  1 {kubelet node00-GUID.oslab.opentlc.com} spec.containers{hello-
openshift} Started  Started with docker id e36321aabeb1
```

3. Test that your pod is responding with `Hello OpenShift`:

```
[andrew@master00-GUID ~]$ ip=`oc describe pod hello-openshift|grep IP:|awk '{print
$2}'`
[andrew@master00-GUID ~]$ curl http://${ip}:8080
```

   - This output denotes a correct response:

```
Hello OpenShift!
```

4. Delete all the objects in your `hello-pod.json` definition file, which, at this point, is the pod only:

```
[andrew@master00-GUID ~]$ oc delete -f hello-pod.json
```

You can also delete a pod using the following command format: `oc delete pod hello-`*`podname`*.

5. Create a new definition file that launches four `hello-openshift` pods:

```
[andrew@master00-GUID ~]$  cat << EOF > hello-many-pods.json
{
  "metadata":{
    "name":"quota-pod-deployment-test"
  },
  "kind":"List",
  "apiVersion":"v1",
  "items":[
    {
      "kind": "Pod",
      "apiVersion": "v1",
      "metadata": {
        "name": "hello-openshift-1",
        "creationTimestamp": null,
        "labels": {
          "name": "hello-openshift"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "hello-openshift",
            "image": "openshift/hello-openshift:v1.0.6",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "resources": {
              "limits": {
                "cpu": "10m",
                "memory": "16Mi"
              }
            },
            "terminationMessagePath": "/dev/termination-log",
            "imagePullPolicy": "IfNotPresent",
            "capabilities": {},
            "securityContext": {
              "capabilities": {},
              "privileged": false
            }
          }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst",
        "serviceAccount": ""
      },
      "status": {}
    },
    {
      "kind": "Pod",
      "apiVersion": "v1",
      "metadata": {
        "name": "hello-openshift-2",
        "creationTimestamp": null,
        "labels": {
          "name": "hello-openshift"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "hello-openshift",
            "image": "openshift/hello-openshift:v1.0.6",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "resources": {
              "limits": {
                "cpu": "10m",
                "memory": "16Mi"
              }
            },
            "terminationMessagePath": "/dev/termination-log",
            "imagePullPolicy": "IfNotPresent",
            "capabilities": {},
            "securityContext": {
              "capabilities": {},
              "privileged": false
            }
          }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst",
        "serviceAccount": ""
      },
      "status": {}
    },
    {
      "kind": "Pod",
      "apiVersion": "v1",
      "metadata": {
        "name": "hello-openshift-3",
        "creationTimestamp": null,
        "labels": {
          "name": "hello-openshift"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "hello-openshift",
            "image": "openshift/hello-openshift:v1.0.6",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "resources": {
              "limits": {
                "cpu": "10m",
                "memory": "16Mi"
              }
            },
            "terminationMessagePath": "/dev/termination-log",
            "imagePullPolicy": "IfNotPresent",
            "capabilities": {},
            "securityContext": {
              "capabilities": {},
              "privileged": false
            }
          }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst",
        "serviceAccount": ""
      },
      "status": {}
    },
    {
      "kind": "Pod",
      "apiVersion": "v1",
      "metadata": {
        "name": "hello-openshift-4",
        "creationTimestamp": null,
        "labels": {
          "name": "hello-openshift"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "hello-openshift",
            "image": "openshift/hello-openshift:v1.0.6",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "resources": {
              "limits": {
                "cpu": "10m",
                "memory": "16Mi"
              }
            },
            "terminationMessagePath": "/dev/termination-log",
            "imagePullPolicy": "IfNotPresent",
            "capabilities": {},
            "securityContext": {
              "capabilities": {},
              "privileged": false
            }
          }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst",
        "serviceAccount": ""
      },
      "status": {}
    }
  ]
}
EOF
```

6. Create the items in the `hello-many-pods.json` file:

```
[andrew@master00-GUID ~]$ oc create -f hello-many-pods.json
pod "hello-openshift-1" created
pod "hello-openshift-2" created
pod "hello-openshift-3" created
Error from server: Pod "hello-openshift-4" is forbidden: limited to 3 pods
```

Because you defined a quota before, `oc create` created three pods only instead of four.

7. Delete the object in the `hello-many-pods.json` definition file (the four pods):

```
[andrew@master00-GUID ~]$ oc delete  -f hello-many-pods.json
```

8. (Optional) Create a project, set the quota with a pod value of `10`, and run `hello-many-pods.json`.

2. Create Services and Routes

1. As `andrew`, create a project called `scvslab`:

```
[andrew@master00-GUID ~]$ oc new-project svcslab --display-name="Services Lab" \
    --description="This is the project we use to learn about services"
```

  - The output looks like this:

    ```
    Now using project "svcslab" on server "https://master00-
    GUID.oslab.opentlc.com:8443".
    ```

    To switch between projects, run `oc project _projectname_`.

2. Create the `hello-service.json` file:

   ```
   [andrew@master00-GUID ~]$  cat <<EOF > hello-service.json
   {
     "kind": "Service",
     "apiVersion": "v1",
     "metadata": {
       "name": "hello-service",
       "labels": {
         "name": "hello-openshift"
       }
     },
     "spec": {
       "selector": {
         "name":"hello-openshift"
       },
       "ports": [
         {
           "protocol": "TCP",
           "port": 8888,
           "targetPort": 8080
         }
       ]
     }
   }
   EOF
   ```

3. Create the `hello-service` service:

   ```
   [andrew@master00-GUID ~]$ oc create -f hello-service.json
   service "hello-service" created
   ```

4. Display the services that are running in the current project:

   ```
   [andrew@master00-GUID ~]$ oc get services
   NAME            CLUSTER_IP      EXTERNAL_IP   PORT(S)     SELECTOR
   AGE
   hello-service   172.30.xxx.yyy  <none>        8888/TCP    name=hello-openshift
   20s
   ```

5. Examine the details of your service. Note the following:
   - **Selector**: Describes which pods the service selects or lists.
   - **Endpoints**: Displays all the pods that are currently listed (none in your current project).

     ```
     [andrew@master00-GUID ~]$ oc describe service hello-service
     Name:   hello-service
     Namespace:  svcslab
     Labels:   name=hello-openshift
     Selector:  name=hello-openshift
     Type:   ClusterIP
     IP:   172.30.231.196
     Port:   <unnamed> 8888/TCP
     Endpoints:  <none>
     Session Affinity: None
     No events.
     ```

6. Create pods according to the `hello-many-pods.json` definition file:

   ```
   [andrew@master00-GUID ~]$ oc create -f hello-many-pods.json
   ```

7. Wait a few seconds and check the service again.
   - The pods that share the label `name=hello-openshift` are all listed:

     ```
     [andrew@master00-GUID ~]$ oc describe service hello-service
     Name:   hello-service
     Namespace:  svcslab
     Labels:   name=hello-openshift
     Selector:  name=hello-openshift
     Type:   ClusterIP
     IP:   172.30.231.196
     Port:   <unnamed> 8888/TCP
     Endpoints:  <none>
     Session Affinity: None
     No events.

     [andrew@master00-GUID ~]$  oc create -f hello-many-pods.json
     pod "hello-openshift-1" created
     pod "hello-openshift-2" created
     pod "hello-openshift-3" created
     pod "hello-openshift-4" created
     [andrew@master00-GUID ~]$  oc describe service hello-service
     Name:   hello-service
     Namespace:  svcslab
     Labels:   name=hello-openshift
     Selector:  name=hello-openshift
     Type:   ClusterIP
     ```

```
IP:   172.30.231.196
Port:   <unnamed> 8888/TCP
Endpoints:  10.1.1.2:8080,10.1.1.3:8080,10.1.2.5:8080 + 1 more...
Session Affinity: None
No events.
```

8. Test that your service is working:

```
[andrew@master00-GUID ~]$ ip=`oc describe service hello-service|grep IP:|awk
'{print $2}'`
[andrew@master00-GUID ~]$ curl http://${ip}:8888
Hello OpenShift!
```

9. Expose your service with the `oc expose` command to create routes for your application:

```
[andrew@master00-GUID ~]$ oc expose service/hello-service --hostname=hello2-
openshift.cloudapps-${guid}.oslab.opentlc.com
```

10. View the route:

```
[andrew@master00-6b80 ~]$ oc get routes
NAME            HOST/PORT                                          PATH
SERVICE         LABELS
hello-service   hello2-openshift.cloudapps-GUID.oslab.opentlc.com
hello-service
```

11. Test the route:

```
[andrew@master00-GUID ~]$ curl http://hello2-openshift.cloudapps-
${guid}.oslab.opentlc.com
Hello OpenShift!
```

3. Explore Containers

Next, take a look at the route and registry containers.

3.1. Explore Route Container

3.1.1. Create Applications As Examples

1. As `andrew`, create a project called `explore-example`:

```
[andrew@master00-GUID ~]$ oc new-project explore-example --display-name="Explore
Example" \
    --description="This is the project we use to learn about connecting to pods"
```

2. Applying the same image as before, run **`oc new-app`** to deploy **`hello-openshift`**:

```
[andrew@master00-GUID ~]$ oc new-app --docker-image=openshift/hello-
openshift:v1.0.6 -l "todelete=yes"
--> Found Docker image 7ce9d7b (10 weeks old) from Docker Hub for "openshift/hello-
openshift:v1.0.6"
    * An image stream will be created as "hello-openshift:v1.0.6" that will track
this image
    * This image will be deployed in deployment config "hello-openshift"
    * Ports 8080/tcp, 8888/tcp will be load balanced by service "hello-openshift"
--> Creating resources with label todelete=yes ...
    ImageStream "hello-openshift" created
    DeploymentConfig "hello-openshift" created
    Service "hello-openshift" created
--> Success
    Run 'oc status' to view your app.
```

3. Verify that **`oc new-app`** has created a pod and the service.

```
[andrew@master00-GUID ~]$ oc get service
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)             SELECTOR
AGE
hello-openshift   172.30.60.163   <none>        8080/TCP,8888/TCP
deploymentconfig=hello-openshift,todelete=yes   2m
[andrew@master00-GUID ~]$ oc get pods
NAME                      READY     STATUS    RESTARTS   AGE
hello-openshift-1-g3xow   1/1       Running   0          2m
```

4. Expose the service and create a route for the application:

```
[andrew@master00-GUID ~]$ oc expose service hello-openshift --
hostname=explore.cloudapps-${guid}.oslab.opentlc.com
```

5. In a later section, you explore the **`docker-registry`** container. To save time, start an S2I build now to push an image into the registry:

```
[andrew@master00-GUID ~]$ oc new-app https://github.com/openshift/sinatra-example -
l "todelete=yes"
```

3.1.2. Connect to Default Router Container

1. As **`root`**, execute the **`bash`** shell inside the router with the **`oc exec`** command along with the default router’s pod name. You have two options.

   Option 1

```
[root@master00-GUID ~]# oc get pods
NAME                       READY     REASON    RESTARTS   AGE
docker-registry-2-snarn    1/1       Running   0          17h
trainingrouter-1-jm5zk     1/1       Running   0          18h
[root@master00-GUID ~]# oc exec -ti trainingrouter-1-jm5zk /bin/bash
```

Option 2

```
[root@master00-GUID ~]#  oc exec -ti `oc get pods |  awk '/route/ { print $1; }'`
"/bin/bash"
```

- With either option, this prompt is displayed:

  ```
  [root@infranode00-GUID conf]#
  ```

  You are now running `bash` inside the container. Also, the prompt specifies that you are on the `infranode` host. That is because the router container resolves the host name through the host's IP address.

2. Do the following:

   a. Run `id`.

   b. Run `pwd` and `ls` and note the directory you are in.

   c. Run `grep SERVERID` on the `haproxy.config` file.

   d. Run `cat haproxy.config` to verify that your configuration file is empty and then view the process status.

   ```
   [root@infranode00-GUID conf]# id
   uid=0(root) gid=0(root) groups=0(root)

   [root@infranode00-GUID conf]# pwd
   /var/lib/haproxy/conf

   [root@infranode00-GUID conf]# ls
   default_pub_keys.pem  os_edge_http_be.map     os_reencrypt.map
   error-page-503.html  os_edge_http_expose.map    os_sni_passthrough.map
   haproxy-config.template  os_edge_http_redirect.map  os_tcp_be.map
   haproxy.config   os_http_be.map

   [root@infranode00-GUID conf]#  grep SERVERID haproxy.config
       cookie OPENSHIFT_explore-example_hello-openshift_SERVERID insert indirect
   nocache httponly
       cookie OPENSHIFT_svcslab_hello-service_SERVERID insert indirect nocache
   httponly
   ```

```
[root@infranode00-GUID conf]# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 02:07 ?        00:00:14 /usr/bin/openshift-router
root        243      0  0 22:08 ?        00:00:00 /bin/bash
root        319      1  0 22:11 ?        00:00:00 /usr/sbin/haproxy -f /var/lib/
root        342    243  0 22:16 ?        00:00:00 ps -ef


[root@infranode00-GUID conf]# cat haproxy.config
```

e. Examine the output, which looks like this:

```
backend be_http_explore-example_hello-openshift

  mode http
  option redispatch
  option forwardfor
  balance leastconn
  timeout check 5000ms
  http-request set-header X-Forwarded-Host %[req.hdr(host)]
  http-request set-header X-Forwarded-Port %[dst_port]
  http-request set-header X-Forwarded-Proto https if { ssl_fc }

    cookie OPENSHIFT_explore-example_hello-openshift_SERVERID insert indirect
nocache httponly
    http-request set-header X-Forwarded-Proto http

  http-request set-header Forwarded for=%[src],host=%[req.hdr(host)],proto=%
[req.hdr(X-Forwarded-Proto)]

  server 10.1.1.7:8080 10.1.1.7:8080 check inter 5000ms cookie 10.1.1.7:8080

...
...
```

- Note the following:
  - The route is the one you created in the previous lab.
  - The route points to the endpoints directly.

3. As `andrew`, scale `hello-openshift` to have five replicas of its pod:

```
[andrew@master00-GUID ~]$ oc get deploymentconfig # or oc get dc
NAME              TRIGGERS                    LATEST
hello-openshift   ConfigChange, ImageChange   1

[andrew@master00-GUID ~]$ oc scale dc hello-openshift --replicas=5
deploymentconfig "hello-openshift" scaled
```

4. Go back to the router container and view the `haproxy.config` file again:

```
[root@infranode00-GUID conf]# grep -A 25 backend.*explore-example_hello-openshift
haproxy.config

backend be_http_explore-example_hello-openshift

  mode http
  option redispatch
  option forwardfor
  balance leastconn
  timeout check 5000ms
  http-request set-header X-Forwarded-Host %[req.hdr(host)]
  http-request set-header X-Forwarded-Port %[dst_port]
  http-request set-header X-Forwarded-Proto https if { ssl_fc }

    cookie OPENSHIFT_explore-example_hello-openshift_SERVERID insert indirect
nocache httponly
    http-request set-header X-Forwarded-Proto http

  http-request set-header Forwarded for=%[src],host=%[req.hdr(host)],proto=%
[req.hdr(X-Forwarded-Proto)]

  server 10.1.1.7:8080 10.1.1.7:8080 check inter 5000ms cookie 10.1.1.7:8080

  server 10.1.1.8:8080 10.1.1.8:8080 check inter 5000ms cookie 10.1.1.8:8080

  server 10.1.1.9:8080 10.1.1.9:8080 check inter 5000ms cookie 10.1.1.9:8080

  server 10.1.2.10:8080 10.1.2.10:8080 check inter 5000ms cookie 10.1.2.10:8080

  server 10.1.2.11:8080 10.1.2.11:8080 check inter 5000ms cookie 10.1.2.11:8080
```

- All of your pods within the `haproxy` configuration are listed.

> Remember, the router routes proxy connections to the pods directly and not through the service. The router uses the service only to obtain a list of the pod endpoints (IP addresses).

3.2. Explore Registry Container

Ensure that your build from earlier is complete.

1. As user `andrew`, run the following to see the build:

```
[andrew@master00-GUID ~]$ oc logs builds/sinatra-example-1
...
...
...
I1120 02:16:05.875303       1 sti.go:298] Successfully built
172.30.41.32:5000/svcslab/sinatra-example:latest
I1120 02:16:06.512944       1 cleanup.go:23] Removing temporary directory /tmp/s2i-
```

```
build079968192
I1120 02:16:06.513477       1 fs.go:99] Removing directory '/tmp/s2i-
build079968192'
I1120 02:16:06.546932       1 sti.go:213] Using provided push secret for pushing
172.30.41.32:5000/svcslab/sinatra-example:latest image
I1120 02:16:06.547064       1 sti.go:217] Pushing
172.30.41.32:5000/svcslab/sinatra-example:latest image ...
I1120 02:19:58.237018       1 sti.go:233] Successfully pushed
172.30.41.32:5000/svcslab/sinatra-example:latest
```

This step takes a while on the lab environment's hardware. If the build is not yet complete, feel free to take a quick break here.

2. As `root`, execute the `bash` shell inside the registry container by running `oc exec` along with the `docker-registry` pod name:

```
[root@master00-GUID ~]#  oc exec -ti  `oc get pods |  awk '/registry/ { print $1;
}'` /bin/bash
```

3. Do the following:

   a. Run `id`.

   b. Run `pwd` and `ls` and note the directory you are in.

   c. Run `cat config.yml` to verify that your configuration file is empty.

```
bash-4.2$ id
uid=1000000000 gid=0(root) groups=0(root)
bash-4.2$ pwd
/
bash-4.2$ ls
bin    config.yml  etc lib    media  opt    registry  run    srv   tmp   var
boot   dev    home lib64  mnt    proc  root      sbin  sys   usr
bash-4.2$ cat config.yml
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    layerinfo: inmemory
  filesystem:
    rootdirectory: /registry
auth:
  openshift:
    realm: openshift
middleware:
  repository:
    - name: openshift
bash-4.2$
```

4. View the repositories and images that are available:

```
bash-4.2$  cd /registry/docker/registry/v2/repositories
bash-4.2$ ls
explore-example
bash-4.2$ ls explore-example/sinatra-example/_layers/
sha256
bash-4.2$ ls explore-example/sinatra-example/_layers/sha256/
50c4d0284685934ca2920fd6e056318cac1187773e8a239dd02d8f248a59d382
50de3644a809b46b344074ca0a691524eb06af3af6a07d25e90c25b50a00980f
9320560b540438b82b1bb1a51d035490812ad9298b945c041da3d0a4b646abf6
e1e04a46f510bf9b3fb68e6cf3fc027100cec875a7ff02e6d0da5206fa7f6b8c
```

Alternatively, if you configured persistent storage for your registry before, view the same in `/var/export/registry-storage/docker/registry/v2/`.

5. As user `andrew`, look at one of the pods you started earlier:

```
[andrew@master00-GUID ~]$ oc get pods
NAME                        READY     STATUS      RESTARTS   AGE
hello-openshift-1-1ecah     1/1       Running     0          27m
hello-openshift-1-b8o3d     1/1       Running     0          27m
hello-openshift-1-g3xow     1/1       Running     0          45m
hello-openshift-1-rbfri     1/1       Running     0          27m
hello-openshift-1-yxidw     1/1       Running     0          27m
sinatra-example-1-build     0/1       Completed   0          11m
sinatra-example-1-yxyod     1/1       Running     0          8m
```

6. Connect to the container:

```
[andrew@master00-GUID ~]$ oc exec -ti sinatra-example-1-yxyod "/bin/bash"
bash-4.2$
```

7. Explore the container:

   a. Run `id`.

   b. Run `pwd` and `ls` and note the directory you are in.

   c. Run `ps -ef` to see what processes are running.

   ```
   bash-4.2$ id
   uid=1000050000 gid=0(root) groups=0(root)
   ```

```
bash-4.2$ pwd
/opt/app-root/src

bash-4.2$ ls
Gemfile       README.md  config.ru   example-mustache  public
Gemfile.lock  app.rb  example-model   example-views   tmp
README       bundle  example-modular  example-views-modular

bash-4.2$ ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
1000050+      1      0  0 22:41 ?        00:00:01 ruby /opt/app-root/src/bundle/
1000050+     33      0  0 22:51 ?        00:00:00 /bin/bash
1000050+     62     33  0 22:51 ?        00:00:00 ps -ef
```

Your pod names and output differ slightly.

4. Create Persistent Volume for Registry

You learn in this lab how to create an NFS export for the registry and to attach the persistent volume to the registry.

4.1. Create NFS Export for Registry

1. As **`root`** on the **`oselab`** host, create a directory for your NFS export:

```
[root@oselab-GUID ~]# export volname=registry-storage
[root@oselab-GUID ~]# mkdir -p /var/export/pvs/${volname}
[root@oselab-GUID ~]# chown nfsnobody:nfsnobody /var/export/pvs/${volname}
[root@oselab-GUID ~]# chmod 700 /var/export/pvs/${volname}
```

2. Add this line to **`/etc/exports`**:

```
[root@oselab-GUID ~]# echo "/var/export/pvs/${volname} *(rw,sync,all_squash)" >>
/etc/exports
```

3. Restart NFS services:

```
[root@oselab-GUID ~]# systemctl restart rpcbind nfs-server nfs-lock nfs-idmap
```

4. As **`root`** on the **`master`** host, create a persistent volume-definition file named **`registry-volume.json`**:

```
[root@oselab-GUID ~]# ssh master00-$guid
[root@master00-GUID ~]# cat << EOF > registry-volume.json
    {
      "apiVersion": "v1",
      "kind": "PersistentVolume",
      "metadata": {
        "name": "registry-storage"
      },
      "spec": {
        "capacity": {
            "storage": "15Gi"
            },
        "accessModes": [ "ReadWriteMany" ],
        "nfs": {
            "path": "/var/export/pvs/registry-storage",
            "server": "oselab-${GUID}.oslab.opentlc.com"
        }
      }
    }

EOF
```

5. In the `default` project, create the `registry-storage` persistent volume from the definition file:

   > You are creating the persistent volume in the `default` project because that is the project in which the registry runs.

```
[root@master00-GUID ~]# oc create -f registry-volume.json -n default
persistentvolume "registry-storage" created
```

6. View the persistent volume you just created:

```
[root@master00-GUID ~]# oc get pv
NAME                LABELS    CAPACITY    ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv21                <none>    5Gi         RWO           Available
20h
pv22                <none>    5Gi         RWO           Available
20h
pv23                <none>    5Gi         RWO           Available
20h
registry-storage    <none>    15Gi        RWX           Available
43s
```

7. Create a `registry-volume-claim.json` claim-definition file to claim your volume:

```
[root@master00-GUID ~]# cat << EOF > registry-volume-claim.json
    {
      "apiVersion": "v1",
      "kind": "PersistentVolumeClaim",
      "metadata": {
        "name": "registry-claim"
      },
      "spec": {
        "accessModes": [ "ReadWriteMany" ],
        "resources": {
          "requests": {
            "storage": "15Gi"
          }
        }
      }
    }

EOF
```

8. Create the `registry-claim` claim from the definition file:

```
[root@master00-GUID ~]# oc create -f registry-volume-claim.json -n default
persistentvolumeclaim "registry-claim" created
```

9. View the persistent volume you created, whose status is `Bound`:

```
[root@master00-GUID ~]# oc get pv
NAME               LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv21               <none>    5Gi        RWO           Available
20h
pv22               <none>    5Gi        RWO           Available
20h
pv23               <none>    5Gi        RWO           Available
20h
registry-storage   <none>    15Gi       RWX           Bound       default/registry-
claim             2m
```

10. View the persistent volume claim you created, whose status is also `Bound`:

```
[root@master00-GUID ~]# oc get pvc
NAME             LABELS    STATUS    VOLUME             CAPACITY   ACCESSMODES
AGE
registry-claim   <none>    Bound     registry-storage   15Gi       RWX
43s
```

4.2. Attach Persistent Volume to Registry

1. Assuming that your registry is already running, obtain the names of `deploymentConfigurations`:

   ```
   [root@master00-GUID ~]# oc get dc
   NAME              TRIGGERS       LATEST
   docker-registry   ConfigChange   1
   trainingrouter    ConfigChange   1
   ```

2. Run `oc volume` to modify `DeploymentConfiguration`.
3. Add the `registry-storage` volume to the registry's `DeploymentConfiguration`, hence redeploying the registry:

   ```
   [root@master00-GUID ~]# oc volume dc/docker-registry --add --overwrite -t
   persistentVolumeClaim \
   --claim-name=registry-claim --name=registry-storage
   ```

4. Run `oc get pods`:

   ```
   [root@master00-GUID ~]# oc get pods
   NAME                        READY     STATUS    RESTARTS   AGE
   docker-registry-2-d9niy     1/1       Running   0          31s
   trainingrouter-1-xcz9o      1/1       Running   0          21h
   ```

   Along with the deletion of the first `docker-registry` container, all the images it stored were also deleted. Now that your registry contains a persistent volume, images are saved even if you delete or replace the `docker-registry` pod.

5. As `andrew` on the `master` host, start an application based on the `https://github.com/openshift/sti-php` repository that would require an S2I build:

   ```
   [root@master00-GUID ~]# su - andrew
   [andrew@master00-GUID ~]$ oc new-app
   openshift/php~https://github.com/openshift/sti-php -l "todelete=yes"
   --> Found image 355eabc (2 weeks old) in image stream "php in project openshift"
   under tag :latest for "openshift/php"
       * A source build using source code from https://github.com/openshift/sti-php
   will be created
         * The resulting image will be pushed to image stream "sti-php:latest"
       * This image will be deployed in deployment config "sti-php"
       * Port 8080/tcp will be load balanced by service "sti-php"
   --> Creating resources with label todelete=yes ...
       ImageStream "sti-php" created
       BuildConfig "sti-php" created
   ```

```
    DeploymentConfig "sti-php" created
    Service "sti-php" created
--> Success
    Build scheduled for "sti-php" - use the logs command to track its progress.
    Run 'oc status' to view your app.
```

6. Check the build logs to ensure that the build is complete and has been pushed into the registry:

```
[andrew@master00-GUID ~]$ oc logs -f builds/sti-php-1
I1126 23:24:28.604316       1 sti.go:298] Successfully built
172.30.42.118:5000/default/sti-php:latest
I1126 23:24:28.716843       1 cleanup.go:23] Removing temporary directory /tmp/s2i-
build491090638
I1126 23:24:28.717016       1 fs.go:99] Removing directory '/tmp/s2i-
build491090638'
I1126 23:24:28.740315       1 sti.go:213] Using provided push secret for pushing
172.30.42.118:5000/default/sti-php:latest image
I1126 23:24:28.740431       1 sti.go:217] Pushing 172.30.42.118:5000/default/sti-
php:latest image ...
I1126 23:25:51.808905       1 sti.go:233] Successfully pushed
172.30.42.118:5000/default/sti-php:latest
```

The `-f` flag sets `oc logs` to "follow" the log, similar to `tail -f`.

7. On the NFS server, `oselab`, verify that the registry is using the `registry-storage` volume:

```
[root@oselab-GUID ~]# find /var/export/pvs/registry-storage | grep sti-php
... Omitted output ...
... Omitted output ...
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_uploads
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_layers
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_layers/sha256
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/812413b2241fa8ff63cb2747bf62e516ff4dc953b1332014faa551655c0ed608
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/812413b2241fa8ff63cb2747bf62e516ff4dc953b1332014faa551655c0ed608
/link
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/b18d4a50300b72f417496313920eff6d4bad00c0f1446686e3d5f157d255d0d2
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
```

```
example/sti-
php/_layers/sha256/b18d4a50300b72f417496313920eff6d4bad00c0f1446686e3d5f157d255d0d2
/link
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/50c4d0284685934ca2920fd6e056318cac1187773e8a239dd02d8f248a59d382
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/50c4d0284685934ca2920fd6e056318cac1187773e8a239dd02d8f248a59d382
/link
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/9320560b540438b82b1bb1a51d035490812ad9298b945c041da3d0a4b646abf6
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_layers/sha256/9320560b540438b82b1bb1a51d035490812ad9298b945c041da3d0a4b646abf6
/link
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_manifests
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_manifests/revisions
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-php/_manifests/revisions/sha256
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_manifests/revisions/sha256/5b8677660e3f1959a0eb44f1ac87200329c721ff4acd8c59f78
a8d0afa5dd425
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_manifests/revisions/sha256/5b8677660e3f1959a0eb44f1ac87200329c721ff4acd8c59f78
a8d0afa5dd425/signatures
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_manifests/revisions/sha256/5b8677660e3f1959a0eb44f1ac87200329c721ff4acd8c59f78
a8d0afa5dd425/signatures/sha256
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_manifests/revisions/sha256/5b8677660e3f1959a0eb44f1ac87200329c721ff4acd8c59f78
a8d0afa5dd425/signatures/sha256/561fd3acac303de8a9c4de202a2e3169bb47f5c03586358d13d
374832e983df5
/var/export/pvs/registry-storage/docker/registry/v2/repositories/explore-
example/sti-
php/_manifests/revisions/sha256/5b8677660e3f1959a0eb44f1ac87200329c721ff4acd8c59f78
a8d0afa5dd425/signatures/sha256/561fd3acac303de8a9c4de202a2e3169bb47f5c03586358d13d
374832e983df5/link
... Omitted output ...
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/53
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/53/53aca6d1d55ccf8f9074725396099dc9592641a2
ae233cb8b1b2de2c800410cb
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/53/53aca6d1d55ccf8f9074725396099dc9592641a2
ae233cb8b1b2de2c800410cb/data
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/b1
```

```
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/b1/b18d4a50300b72f417496313920eff6d4bad00c0
f1446686e3d5f157d255d0d2
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/b1/b18d4a50300b72f417496313920eff6d4bad00c0
f1446686e3d5f157d255d0d2/data
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/50
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/50/50c4d0284685934ca2920fd6e056318cac118777
3e8a239dd02d8f248a59d382
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/50/50c4d0284685934ca2920fd6e056318cac118777
3e8a239dd02d8f248a59d382/data
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/93
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/93/9320560b540438b82b1bb1a51d035490812ad929
8b945c041da3d0a4b646abf6
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/93/9320560b540438b82b1bb1a51d035490812ad929
8b945c041da3d0a4b646abf6/data
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/93/931b7ebd6c92756356ae4174a02b845480c5c548
84875533ffa4cbef3872199a
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/93/931b7ebd6c92756356ae4174a02b845480c5c548
84875533ffa4cbef3872199a/data
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/81
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/81/812413b2241fa8ff63cb2747bf62e516ff4dc953
b1332014faa551655c0ed608
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/81/812413b2241fa8ff63cb2747bf62e516ff4dc953
b1332014faa551655c0ed608/data
/var/export/pvs/registry-storage/docker/registry/v2/blobs/sha256/56
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/56/561fd3acac303de8a9c4de202a2e3169bb47f5c0
3586358d13d374832e983df5
/var/export/pvs/registry-
storage/docker/registry/v2/blobs/sha256/56/561fd3acac303de8a9c4de202a2e3169bb47f5c0
3586358d13d374832e983df5/data
```

You can see that previously created images are not in the registry, they were created before the registry was restarted and given a persistent volume.