

# HA Deployment Lab

---

In this lab's scenario, you are a consultant assigned to MitziCom, a telecommunications company. MitziCom provides hosting and cloud services to a variety of clients, ranging from medium-sized companies to enterprise giants.

MitziCom has asked you to lead a 30- to 40-hour proof-of-concept (POC) using Red Hat OpenShift Container Platform. The purpose of the POC is to determine the feasibility of using Red Hat OpenShift Container Platform as a target for internal and client workloads.

The lab details a recommended process for meeting MitziCom's requirements, as defined in the lab's goals. Information about the infrastructure, DNS names, passwords, and more is provided throughout the lab.



Because the lab does not include all of the required steps and commands, you will want to consult the {ocp\_docs} product documentation. This link is especially useful for research and troubleshooting. Your instructor is also available to help you.

## Goals

In this lab, you deploy and configure OpenShift Container Platform on a group of servers to meet these requirements:

- Configure Red Hat Enterprise Linux (RHEL) hosts for OpenShift deployment
- Deploy a highly available OpenShift Container Platform cluster
- Configure the OpenShift Container Platform cluster

## Provisioned Environment Hosts

- Bastion host: `bastion.$GUID.example.opentlc.com` and `bastion.$GUID.internal`
- Load balancer: `loadbalancer.$GUID.example.opentlc.com` and `loadbalancer.$GUID.internal`
- 3 OpenShift master nodes: `master{1-3}.$GUID.internal`
- 2 OpenShift infrastructure nodes: `infranode{1,2}.$GUID.example.opentlc.com` and `infranode{1,2}.$GUID.internal`
- 3 OpenShift worker nodes: `node{1-3}.$GUID.internal`
- NFS server: `support1.$GUID.internal`
- IPA Server: `ipa.shared.example.opentlc.com` (shared resource for all students)

---

# 1. Configure RHEL Hosts for OpenShift Deployment

---

In this section, you prepare the hosts for installation. You then make sure that the correct Docker (or CRI-O) version is installed and configured. You also configure yum repositories on the hosts.

## 1.1. Explore and Verify Infrastructure Deployment

---

Instances are already created for you in the Ansible inventory file, `/etc/ansible/hosts`. Examine the end of `/etc/ansible/hosts` and see that it is populated with the hosts in your cluster.

1. Verify that the Ansible hosts file is populated.

```
cat /etc/ansible/hosts
```

### Sample Output

```

#
# ansible inventory for OpenShift Container Platform 3.11.16
# AgnosticD ansible-config: ocp-ha-lab

[OSEv3:vars]

[ ... Lots of stuff omitted ... ]

#####
### OpenShift Hosts
#####
# openshift_node_labels DEPRECATED
# openshift_node_problem_detector_install

[OSEv3:children]
lb
masters
etcd
nodes

[lb]
loadbalancer.GUID.internal

[masters]
master1.GUID.internal
master2.GUID.internal
master3.GUID.internal

[etcd]
master1.GUID.internal
master2.GUID.internal
master3.GUID.internal

[nodes]
## These are the masters
master1.GUID.internal openshift_node_group_name='node-config-master' openshif
master2.GUID.internal openshift_node_group_name='node-config-master' openshif
master3.GUID.internal openshift_node_group_name='node-config-master' openshif

```

```
## These are infranodes
```

```
infranode1.GUID.internal openshift_node_group_name='node-config-infra' openshift
```

```
infranode2.GUID.internal openshift_node_group_name='node-config-infra' openshift
```

```
## These are regular nodes
```

```
node1.GUID.internal openshift_node_group_name='node-config-compute' openshift
```

```
node2.GUID.internal openshift_node_group_name='node-config-compute' openshift
```

```
node3.GUID.internal openshift_node_group_name='node-config-compute' openshift
```

2. Use the Ansible `--list-hosts` command to list the masters, nodes, and all of the host groups.

```
ansible masters --list-hosts
```

### Sample Output

```
hosts (3):
```

```
  master1.GUID.internal
```

```
  master2.GUID.internal
```

```
  master3.GUID.internal
```

```
ansible nodes --list-hosts
```

### Sample Output

```
hosts (8):
```

```
  master1.GUID.internal
```

```
  master2.GUID.internal
```

```
  master3.GUID.internal
```

```
  infranode1.GUID.internal
```

```
  infranode2.GUID.internal
```

```
  node1.GUID.internal
```

```
  node2.GUID.internal
```

```
  node3.GUID.internal
```

```
ansible all --list-hosts
```

### Sample Output

```
hosts (10):
  master1.GUID.internal
  master2.GUID.internal
  master3.GUID.internal
  infranode1.GUID.internal
  infranode2.GUID.internal
  node1.GUID.internal
  node2.GUID.internal
  node3.GUID.internal
  loadbalancer.GUID.internal
```

3. Verify that all of your hosts are running:

```
ansible all -m ping
```

## 1.2. Verify Installation and Configuration of Docker

---

1. Verify that Docker is running on all of the nodes in the cluster.

```
ansible nodes -m shell -a"systemctl status docker | grep Active"
```

### Sample Output

```
node1.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
node2.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
node3.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
master2.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
infranode1.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:04 UTC; 15h ago
```

```
infranode2.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
master1.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

```
master3.GUID.internal | SUCCESS | rc=0 >>
```

```
Active: active (running) since Wed 2018-10-17 23:25:03 UTC; 15h ago
```

## 2. Make sure that the Docker version is correct for the desired OpenShift version.

```
ansible nodes -m shell -a"docker version|grep Version"
```

### Sample Output

```
node1.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

node2.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

node3.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

infranode1.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

infranode2.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

master1.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

master2.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1

master3.GUID.internal | SUCCESS | rc=0 >>
Version:           1.13.1
Version:           1.13.1
```

## 1.3. Verify Yum Repositories and NFS Shared Volumes on Hosts

---

The required Yum repositories and NFS shared volumes are already set up on the environment. In this section, you verify that they are set up properly.

1. List the repositories on all of the hosts.

```
ansible all -m shell -a"yum repolist"
```

### Sample Output

```
Loaded plugins: amazon-id, rhui-lb, search-disabled-repos
repo id                                repo name
status
rh-gluster-3-client-for-rhel-7-server-rpms  Red Hat Enterprise Linux
GlusterFS Client (RPMs)                  195
rhel-7-server-ansible-2.6-rpms           Red Hat Enterprise Linux
Ansible (RPMs)                           11
rhel-7-server-extras-rpms               Red Hat Enterprise Linux 7
Extras                                  923
rhel-7-server-optional-rpms             Red Hat Enterprise Linux 7
Optional                               15,358
rhel-7-server-ose-3.11-rpms             Red Hat Enterprise Linux 7 OSE
3.11                                   228
rhel-7-server-rh-common-rpms            Red Hat Enterprise Linux 7
Common                                 234
rhel-7-server-rpms                      Red Hat Enterprise Linux 7
21,065
repolist: 38,014
[...]
```

2. Examine the NFS server to see which NFS volumes are shared:

```
ansible nfs -m shell -a"exportfs"
```

### Sample Output

```
support1.GUID.internal | SUCCESS | rc=0 >>
/srv/nfs                <world>
```

---

## 2. Deploy Highly Available OpenShift Cluster

---



In this section, you use the Ansible advanced installer to deploy OpenShift as a clustered, highly available installation that includes a load balancer in front of the API servers. The environment includes three masters, two infrastructure nodes, three worker nodes, and the load balancer. The environment also supports several hosted components of OpenShift, such as the integrated container registry, the routers, the service catalog, metrics, and aggregated logging.

1. To prepare for the installation on the `bastion` host, install the `atomic-openshift-clients` package, which includes the OpenShift client software and the `openshift-ansible` package, which includes the installer and has Ansible and the playbooks as dependencies. You may find that these packages are already installed.

```
yum -y install atomic-openshift-clients openshift-ansible
```

## 2.1. Review Requirements for OpenShift Deployment

You can find the Ansible inventory file in the default location, `/etc/ansible/hosts`. It is used to define and configure the OpenShift cluster.



Red Hat has provided you with an inventory file in `/etc/ansible/hosts` that has the parameters in the proper place, but leaves it up to you to find their correct values.

Your OpenShift Container Platform cluster is expected to have the following characteristics defined in the inventory file:

- Three master hosts in the deployment
- A load balancer to access the masters that is configured with `loadbalancer.${GUID}.example.opentlc.com` as the external DNS entry and `loadbalancer.${GUID}.internal` as the internal DNS entry
- Two infrastructure nodes, each running a router
- One support node, for NFS
- An integrated registry pod backed by persistent volume (PV) storage
- Router pods deployed, configured, and running on each infrastructure node in the cluster
- Aggregated logging configured and working
- Metrics collection configured and working
- All hosted components (router, registry, Prometheus, logging, metrics, service brokers) running on infrastructure nodes
- A `*.apps.${GUID}.example.opentlc.com` wildcard DNS entry that points to the infrastructure nodes

## 2.2. Configure Authentication Against IPA (LDAP) Server

In this section, you edit the Ansible inventory file to configure the OpenShift master API servers to authenticate against an existing IPA (LDAP) server. After installation by the `openshift-ansible` deployer is complete, you synchronize or create OpenShift group objects to match the groups that are configured in IPA.

### 2.2.1. Distribute Identity Management (IdM) Certificate Authority Certificates

---

The IPA server sets up its own Certificate Authority (CA) and does not use the same CA as your RHOCF installation. In order for RHOCF to make secured LDAP requests to the IPA server, your master servers need the CA certificate.

1. On the `bastion` host, download the `ca.crt` file:

```
cd /root
wget http://ipa.shared.example.opentlc.com/ipa/config/ca.crt -O /root/ipa-
ca.crt
```

### 2.2.2. Set Up IPA as Authentication Provider

---

1. Configure LDAP authentication using the Ansible installer and the following information to set up the authentication provider in the Ansible inventory file:

- **bindDN:** `uid=admin,cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com`
- **bindPassword:** `r3dh4t1!`
- **ca:** `/etc/origin/master/ldap_ldap_ca.crt`
- **url:**

```
ldaps://ipa.shared.example.opentlc.com:636/cn=users,cn=accounts,dc=shared,d
```

2. Use the `openshift_master_ldap_ca_file` variable to copy the certificate to the masters.
3. Make sure that the `/etc/ansible/hosts` file contains the following in the `[OSEv3:vars]` section and that the file does not contain any other values for `openshift_master_identity_provider`:

```
openshift_master_identity_providers=[{'name': 'ldap', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN':
'uid=admin,cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com',
'bindPassword': 'r3dh4t1!', 'ca':
'/etc/origin/master/ldap_ldap_ca.crt', 'insecure': 'false', 'url':
'ldaps://ipa.shared.example.opentlc.com:636/cn=users,cn=accounts,dc=shared
,dc=example,dc=opentlc,dc=com?uid?sub?(memberOf=cn=ocp-
users,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com)'}]]

openshift_master_ldap_ca_file=/root/ipa-ca.crt
```

### 2.2.3. Set Up OpenShift Hosted Components

1. Continue to edit your Ansible inventory file, now addressing hosted components:

a. OpenShift registries

- As of version 3.11, OpenShift is installed by authenticating to the **registry.redhat.io** registries and the old registry, **registry.access.redhat.com**.
- Ask your instructor for the proper authentication credentials for your lab.

b. **openshift\_hosted\_infra\_selector=** ???

- Default node selector for infrastructure components (*only* for router/registry/template service broker/OpenShift Ansible Broker)
- Consider using the deprecated **openshift\_hosted\_infra\_selector=** variable if you do not set selectors for each hosted component.

c. Logging components: Elasticsearch, Kibana, Logging Curator

- Find several proper parameters for logging and associated storage.

d. Metrics components: Prometheus, Cassandra, Hawkular, Heapster

- Find several proper parameters for metrics and associated storage.

e. Service Catalog components: API server, Template Service Broker, OpenShift Ansible Broker

- Find several proper parameters for the service catalog and its associated storage.

f. Operator Lifecycle Manager (OLM) (Technology Preview)

- The OLM is necessary for manipulating operators in the web console.



Consult the fully populated and working solution hosts file, available in `/var/preserve/hosts` on your `bastion` host—the only item missing is the Service Account for the registries.

## 2.3. Execute `openshift-ansible` Installer

1. Run `ansible-playbook` to check that all prerequisites are met.

- Expect it to take less than five minutes.



Use `tmux` (<https://tmuxcheatsheet.com>) before any `ansible-playbook` runs. To enable scrolling in the `tmux` window, use **Ctrl+B** (then **Ctrl+B Q** to quit scroll mode).

On a Mac, use "iTerm2" (<https://www.iterm2.com>), then you can use `tmux -CC` to create a secondary tab with full scrollback capability.

```
cd /usr/share/ansible/openshift-ansible/  
ansible-playbook playbooks/prerequisites.yml
```

### Sample Output

## PLAY RECAP

\*\*\*\*\*

\*\*\*\*\*

```
infranode1.GUID.internal : ok=50    changed=19    unreachable=0
failed=0
infranode2.GUID.internal : ok=50    changed=19    unreachable=0
failed=0
loadbalancer.GUID.internal : ok=25    changed=7     unreachable=0
failed=0
localhost                : ok=13    changed=0     unreachable=0
failed=0
master1.GUID.internal    : ok=62    changed=13    unreachable=0
failed=0
master2.GUID.internal    : ok=62    changed=13    unreachable=0
failed=0
master3.GUID.internal    : ok=68    changed=14    unreachable=0
failed=0
node1.GUID.internal      : ok=61    changed=13    unreachable=0
failed=0
node2.GUID.internal      : ok=61    changed=13    unreachable=0
failed=0
node3.GUID.internal      : ok=61    changed=13    unreachable=0
failed=0
```

## INSTALLER STATUS

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

```
Initialization          : Complete (0:00:27)
master1.GUID.internal    : ok=64    changed=20    unreachable=0
failed=0
master2.GUID.internal    : ok=53    changed=19    unreachable=0
failed=0
master3.GUID.internal    : ok=53    changed=19    unreachable=0
failed=0
node1.GUID.internal      : ok=50    changed=19    unreachable=0
failed=0
node2.GUID.internal      : ok=50    changed=19    unreachable=0
failed=0
```

## INSTALLER STATUS

\*\*\*\*\*

\*\*\*\*\*

Initialization : Complete (0:01:35)

Thursday 18 October 2018 19:02:05 +0000 (0:00:00.389)

0:03:03.080 \*\*\*\*\*

=====

=====

Ensure openshift-ansible installer package deps are installed -----  
----- 38.10s

Run variable sanity checks -----  
----- 25.07s

container\_runtime : Install Docker -----  
----- 14.30s

os\_firewall : need to pause here, otherwise the iptables service  
starting can sometimes cause ssh to fail -----  
10.11s

openshift\_excluder : Install docker excluder - yum -----  
----- 5.29s

container\_runtime : restart container runtime -----  
----- 3.86s

container\_runtime : Create credentials for oreg\_url -----  
----- 3.78s

os\_firewall : Start and enable iptables service -----  
----- 3.75s

Query DNS for IP address of master1.GUID.internal -----  
----- 2.00s

Gathering Facts -----  
----- 1.66s

container\_runtime : Fixup SELinux permissions for docker -----  
----- 1.42s

os\_firewall : Install iptables packages -----  
----- 1.22s

Gathering Facts -----  
----- 1.06s

Gather Cluster facts -----  
----- 1.02s

openshift\_repos : refresh cache -----  
----- 0.94s

```
Initialize openshift.node.sdn_mtu -----
----- 0.90s
container_runtime : Configure Docker service unit file -----
----- 0.90s
openshift_repos : Ensure libselinux-python is installed -----
----- 0.80s
container_runtime : Get current installed Docker version -----
----- 0.79s
openshift_excluder : Enable docker excluder -----
----- 0.76s
```

2. Run **ansible-playbook** to deploy your cluster.

- This takes about 30 minutes.

```
cd /usr/share/ansible/openshift-ansible
ansible-playbook playbooks/deploy_cluster.yml
```

## Sample Output

## PLAY RECAP

\*\*\*\*\*

\*\*\*\*\*

infranode1.GUID.internal : ok=114 changed=64 unreachable=0  
failed=0

infranode2.GUID.internal : ok=114 changed=64 unreachable=0  
failed=0

loadbalancer.GUID.internal : ok=27 changed=10 unreachable=0  
failed=0

localhost : ok=13 changed=0 unreachable=0  
failed=0

master1.GUID.internal : ok=1268 changed=543 unreachable=0  
failed=0

master2.GUID.internal : ok=286 changed=151 unreachable=0  
failed=0

master3.GUID.internal : ok=286 changed=151 unreachable=0  
failed=0

node1.GUID.internal : ok=113 changed=63 unreachable=0  
failed=0

node2.GUID.internal : ok=113 changed=63 unreachable=0  
failed=0

## INSTALLER STATUS

\*\*\*\*\*

\*\*\*\*\*

Initialization : Complete (0:00:33)

Health Check : Complete (0:00:16)

Node Bootstrap Preparation : Complete (0:03:37)

etcd Install : Complete (0:00:44)

Load Balancer Install : Complete (0:00:13)

Master Install : Complete (0:04:46)

Master Additional Install : Complete (0:00:41)

Node Join : Complete (0:00:53)

Hosted Install : Complete (0:00:59)

Cluster Monitoring Operator : Complete (0:00:39)

Web Console Install : Complete (0:00:29)

Console Install : Complete (0:00:24)

Metrics Install : Complete (0:01:31)

metrics-server Install : Complete (0:00:32)



```
Logging Install           : Complete (0:02:34)
Prometheus Install       : Complete (0:00:35)
Service Catalog Install  : Complete (0:01:34)
OLM Install              : Complete (0:00:31)
Node Problem Detector Install : Complete (0:00:07)
Thursday 18 October 2018 19:32:54 +0000 (0:00:00.048)
0:26:15.265 *****
```

```
=====
=====
openshift_control_plane : Wait for all control plane pods to become
ready ----- 77.39s
openshift_control_plane : Wait for control plane pods to appear -----
----- 45.94s
Approve node certificates when bootstrapping -----
----- 40.65s
openshift_cluster_monitoring_operator : Wait for the ServiceMonitor CRD
to be created ----- 30.86s
openshift_node : install needed rpm(s) -----
----- 29.99s
openshift_node : Install node, clients, and contrack packages -----
----- 29.92s
openshift_web_console : Verify that the console is running -----
----- 21.62s
openshift_ca : Install the base package for admin tooling -----
----- 16.85s
openshift_console : Waiting for console rollout to complete -----
----- 16.10s
Run health checks (install) - EL -----
----- 15.61s
openshift_service_catalog : oc_process -----
----- 14.76s
Run variable sanity checks -----
----- 14.11s
template_service_broker : Verify that TSB is running -----
----- 12.07s
openshift_hosted : Wait for registry pods -----
----- 11.17s
openshift_node : Install Ceph storage plugin dependencies -----
----- 11.09s
```



If your installation fails, check the following:

- If you changed the SDN or certificates, uninstall the entire cluster, and then reinstall the cluster with `deploy_cluster`.
- If you deployed OCS, uninstall OCS, and then reinstall OCS with `deploy_cluster`.

## 2.4. Uninstall OpenShift (Reference)

In case you need to uninstall OpenShift, follow the instructions at [https://docs.openshift.com/container-platform/3.11/install/uninstalling\\_cluster.html](https://docs.openshift.com/container-platform/3.11/install/uninstalling_cluster.html) ([https://docs.openshift.com/container-platform/3.11/install/uninstalling\\_cluster.html](https://docs.openshift.com/container-platform/3.11/install/uninstalling_cluster.html)).

1. Run the uninstall playbook:

```
ansible-playbook /usr/share/ansible/openshift-  
ansible/playbooks/adhoc/uninstall.yml
```

2. After the playbook run completes, it is also usually necessary to delete all leftover content, such as certificates, in the `/etc/origin` directories on all masters and nodes:

```
ansible nodes -a "rm -rf /etc/origin"
```

3. Finally, you must delete all data from the NFS server:

```
ansible nfs -a "rm -rf /srv/nfs/*"
```

## 3. Verify OpenShift Cluster

In this section, you verify that the proper configuration and components have been deployed in accordance with your Ansible inventory file.

1. Once the installation is complete, copy the `.kube` directory from `master1` to your `bastion` host:

```
ansible masters[0] -b -m fetch -a "src=/root/.kube/config  
dest=/root/.kube/config flat=yes"
```

- This enables you to run `oc` commands as `system:admin` on the `bastion` host without using SSH to access a master.

2. Verify that you are now `system:admin`:

```
oc whoami
```

### Sample Output

```
system:admin
```

## 3.1. Verify Configuration

1. Run `oc get nodes` to display the nodes:

```
oc get nodes --show-labels
```

### Sample Output

NAME	STATUS	ROLES	AGE	VERSION	LA
infranode1.GUID.internal	Ready	infra	8m	v1.10.0+b81c8f8	be
infranode2.GUID.internal	Ready	infra	8m	v1.10.0+b81c8f8	be
master1.GUID.internal	Ready	master	14m	v1.10.0+b81c8f8	be
master2.GUID.internal	Ready	master	14m	v1.10.0+b81c8f8	be
master3.GUID.internal	Ready	master	14m	v1.10.0+b81c8f8	be
node1.GUID.internal	Ready	compute	8m	v1.10.0+b81c8f8	be
node2.GUID.internal	Ready	compute	8m	v1.10.0+b81c8f8	be
node3.GUID.internal	Ready	compute	8m	v1.10.0+b81c8f8	be

2. Validate that all pods are running (not pending) and on the correct nodes:

```
oc get pod --all-namespaces -o wide
```

### Sample Output

```
oc get pods --all-namespaces -o wide
```

NAMESPACE	NAME
default	docker-registry-1-xzzmz
default	registry-console-1-5ttgw
default	router-1-kf78n
default	router-1-kgcnk
kube-service-catalog	apiserver-9skh4
kube-service-catalog	apiserver-txgqh
kube-service-catalog	apiserver-xv2n6
kube-service-catalog	controller-manager-89wkb
kube-service-catalog	controller-manager-jb2qc
kube-service-catalog	controller-manager-mvpkf
kube-system	master-api-master1.GUID.internal
kube-system	master-api-master2.GUID.internal
kube-system	master-api-master3.GUID.internal
kube-system	master-controllers-master1.GUID.internal
kube-system	master-controllers-master2.GUID.internal
kube-system	master-controllers-master3.GUID.internal
kube-system	master-etcd-master1.GUID.internal
kube-system	master-etcd-master2.GUID.internal
kube-system	master-etcd-master3.GUID.internal
openshift-ansible-service-broker	asb-1-lbntn
openshift-infra	hawkular-cassandra-1-6nsnd
openshift-infra	hawkular-metrics-rwr86
openshift-infra	hawkular-metrics-schema-szc8m
openshift-infra	heapster-xlz5l
openshift-logging	logging-curator-1-4jtgz
openshift-logging	logging-es-data-master-v6jijv2s-1-deploy
openshift-logging	logging-es-data-master-v6jijv2s-1-gwpzc
openshift-logging	logging-fluentd-46njb
openshift-logging	logging-fluentd-5qxx7
openshift-logging	logging-fluentd-b2rk2
openshift-logging	logging-fluentd-gv277
openshift-logging	logging-fluentd-hnssc
openshift-logging	logging-fluentd-t8fhn
openshift-logging	logging-fluentd-zlk47
openshift-logging	logging-kibana-1-nkqfv
openshift-metrics	prometheus-0
openshift-metrics	prometheus-node-exporter-42x46

openshift-metrics	prometheus-node-exporter-hdl6q
openshift-metrics	prometheus-node-exporter-jx4sj
openshift-metrics	prometheus-node-exporter-k8zvc
openshift-metrics	prometheus-node-exporter-l8nzn
openshift-metrics	prometheus-node-exporter-pg8lb
openshift-metrics	prometheus-node-exporter-xzh4f
openshift-node	sync-f7mdw
openshift-node	sync-k5t9l
openshift-node	sync-lpc6c
openshift-node	sync-stx2x
openshift-node	sync-t77rn
openshift-node	sync-v5xhb
openshift-node	sync-vjcm7
openshift-sdn	ovs-2jjkm
openshift-sdn	ovs-97g5d
openshift-sdn	ovs-9bc7g
openshift-sdn	ovs-kd9GUID
openshift-sdn	ovs-n9bmq
openshift-sdn	ovs-nk66h
openshift-sdn	ovs-rg7q6
openshift-sdn	sdn-92vgx
openshift-sdn	sdn-bgqvz
openshift-sdn	sdn-bp2b6
openshift-sdn	sdn-fjgjk
openshift-sdn	sdn-w46w8
openshift-sdn	sdn-xf65q
openshift-sdn	sdn-zztcr
openshift-template-service-broker	apiserver-7zcx8
openshift-template-service-broker	apiserver-fst29
openshift-template-service-broker	apiserver-wjsth
openshift-web-console	webconsole-7f944b7c85-ldvhz
openshift-web-console	webconsole-7f944b7c85-q7jrh
openshift-web-console	webconsole-7f944b7c85-zg4b9

- This shows the results for a fully configured OpenShift cluster.

## 3.2. Verify Authentication Provider Configuration

In this section, you verify the configuration of the authentication provider by attempting to log in to the web console.

1. Navigate to the OpenShift Container Platform web console.
2. Log in using `payment1` as the username and `r3dh4t1!` as the password.
3. If you are unable to authenticate successfully, double-check your configuration.

### 3.3. Synchronize Groups from IPA Server to OpenShift Cluster

---

1. Synchronize the following groups from the IPA server to your OpenShift cluster:

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

- a. Use the following hints to complete this task:

- **url:** `ldap://ipa.shared.example.opentlc.com` or `ldaps://ipa.shared.example.opentlc.com:636`
- **ca:** `/etc/origin/master/ldap_ldap_ca.crt`
- **bindDN:**  
`uid=admin,cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com`
- **bindPassword:** `r3dh4t1!`
- **baseDN for groupsQuery:**  
`cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com`
- **baseDN for usersQuery:**  
`cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com`
- **filter:**  

```
(&(! (objectClass=mepManagedEntry))(! (cn=trust admins))(! (cn=groups))(!
```
- LDAP groups are referenced like this:  
`cn=portalapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com`



The OPENTLC IPA server has a lot of groups defined. To limit the number of groups that are synced, set up a whitelist.

- b. On the `master1` host, create the `/etc/origin/master/groupsync.yaml` file:

```

ssh master1.$GUID.internal
sudo -i

cat << EOF > /etc/origin/master/groupsync.yaml
kind: LDAPSyncConfig
apiVersion: v1
url: "ldap://ipa.shared.example.opentlc.com"
insecure: false
ca: "/etc/origin/master/ldap_ldap_ca.crt"
bindDN:
"uid=admin,cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com"
bindPassword: "r3dh4t1!"
rfc2307:
  groupsQuery:
    baseDN:
"cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com"
    scope: sub
    derefAliases: never
    filter: (&(!(objectClass=mepManagedEntry))(!(cn=trust
admins))(!(cn=groups))(!(cn=admins))(!(cn=ipausers))(!(cn=editors))(!(
cn=ocp-users))(!(cn=evmgroupp*))(!(cn=ipac*)))
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
    groupMembershipAttributes: [ member ]
    usersQuery:
      baseDN:
"cn=users,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com"
      scope: sub
      derefAliases: never
      userUIDAttribute: dn
      userNameAttributes: [ uid ]
EOF

```

- c. Map LDAP groups to specific names in RHOCF by adding this section to the `/etc/origin/master/groupsync.yaml` file:

```

echo '
groupUIDNameMapping:

"cn=portalapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com": "portalapp"

"cn=paymentapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com": "paymentapp"
  "cn=ocp-
production,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com": "ocp-production"
  "cn=ocp-
platform,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com": "ocp-platform"
' >>/etc/origin/master/groupsync.yaml

```

d. Create the `/etc/origin/master/whitelist.yaml` file:

```

cat << EOF > /etc/origin/master/whitelist.yaml
cn=portalapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com
cn=paymentapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com
cn=ocp-
platform,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com
cn=ocp-
production,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com
EOF

```

### 3.4. Test Group Synchronization (Optional)

1. Still on the `master1` host, verify user IDs:

```
oc get users
```

2. If there are users with the `htpasswd_auth` identity, delete them as shown in this example:



```
oc delete user andrew
```

3. Delete any `htpasswd` identities as shown in this example:

```
oc delete identity htpassword:uid=andrew
```

4. As a test, run the synchronization:

```
oc adm groups sync --sync-config=/etc/origin/master/groupsync.yaml --  
whitelist=/etc/origin/master/whitelist.yaml
```

### Sample Output

```
apiVersion: v1  
items:  
- apiVersion: v1  
  kind: Group  
  metadata:  
    annotations:  
      openshift.io/ldap.sync-time: 2018-04-18T20:32:48Z  
      openshift.io/ldap.uid:  
cn=portalapp,cn=groups,cn=accounts,dc=shared,dc=example,dc=opentlc,dc=com  
      openshift.io/ldap.url: ipa.shared.example.opentlc.com:389  
      creationTimestamp: null  
      labels:  
        openshift.io/ldap.host: ipa.shared.example.opentlc.com  
      name: portalapp  
  users:  
    - andrew  
    - portal1  
    - portal2  
    - chenzhen  
[...]
```

## 3.5. Synchronize Groups

1. Run the same `oc adm groups sync` command as shown in the optional section above, but add `--confirm` to create the groups:

```
oc adm groups sync --sync-config=/etc/origin/master/groupsync.yaml --
whitelist=/etc/origin/master/whitelist.yaml --confirm
```

### Sample Output

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

2. Verify that the groups are created:

```
oc get groups
```

### Sample Output

NAME	USERS
ocp-platform	david, admin1, admin2, luochen, loren, chenzhen
ocp-production	karla, prod1, prod2, luochen, loren, chenzhen
paymentapp	marina, payment1, payment2, chenzhen
portalapp	andrew, portal1, portal2, chenzhen

3. Disconnect from the `master1` host and return to the `bastion` host.

---

## 4. Perform Post-Installation Configuration

In this section, you perform the post-installation configuration for NFS. If you used NFS as a remote datastore, you need to create persistent volumes (PVs) for users to consume. The specifications for the PVs are as follows:

- 25 PVs with these parameters:
  - **Size:** 5 GB
  - **PV access:** `ReadWriteOnce`
  - **ReclaimPolicy:** `Delete`
- 25 PVs with these parameters:
  - **Size:** 10 GB

- **PV access:** `ReadWriteMany`
- **ReclaimPolicy:** `Retain`

Here are the steps to create PVs for users:

1. Create directories on the `support1.$GUID.internal` NFS server to be used as PVs in the OpenShift cluster.



These directories need to be under `/srv/nfs` because this directory is backed by a separate volume group.

2. As `root`, paste the following into your bastion host command line:

```
export GUID=`hostname|awk -F. '{print $2}'`
echo "export GUID=$GUID" >> ~/.bashrc
echo $GUID

ssh support1.$GUID.internal
sudo -i
mkdir -p /srv/nfs/user-vols/pv{1..200}

for pvnum in {1..50} ; do
echo /srv/nfs/user-vols/pv${pvnum} *(rw,root_squash) >>
/etc/exports.d/openshift-uservols.exports
chown -R nfsnobody.nfsnobody /srv/nfs
chmod -R 777 /srv/nfs
done

systemctl restart nfs-server
exit
exit
```

3. On your `bastion` host, create 25 definition files for PVs, labeled `pv1` to `pv25`, with a size of 5 GB and `ReadWriteOnce` access mode.

```

export GUID=`hostname|awk -F. '{print $2}'`

export volsize="5Gi"
mkdir /root/pvs
for volume in pv{1..25} ; do
cat << EOF > /root/pvs/${volume}
{
    "apiVersion": "v1",
    "kind": "PersistentVolume",
    "metadata": {
        "name": "${volume}"
    },
    "spec": {
        "capacity": {
            "storage": "${volsize}"
        },
        "accessModes": [ "ReadWriteOnce" ],
        "nfs": {
            "path": "/srv/nfs/user-vols/${volume}",
            "server": "support1.${GUID}.internal"
        },
        "persistentVolumeReclaimPolicy": "Recycle"
    }
}
EOF
echo "Created def file for ${volume}";
done;

```

4. On your **bastion** host, create 25 definition files for PVs, labeled **pv26** to **pv50**, with a size of 10 GB and **ReadWriteMany** access mode.

```
export GUID=`hostname|awk -F. '{print $2}'`

export volsize="10Gi"
for volume in pv{26..50} ; do
cat << EOF > /root/pvs/${volume}
{
  "apiVersion": "v1",
  "kind": "PersistentVolume",
  "metadata": {
    "name": "${volume}"
  },
  "spec": {
    "capacity": {
      "storage": "${volsize}"
    },
    "accessModes": [ "ReadWriteMany" ],
    "nfs": {
      "path": "/srv/nfs/user-vols/${volume}",
      "server": "support1.${GUID}.internal"
    },
    "persistentVolumeReclaimPolicy": "Retain"
  }
}
EOF
echo "Created def file for ${volume}";
done;
```

5. Use `oc create` to create all of the PVs you defined.

```
cat /root/pvs/* | oc create -f -
```

6. Double-check your PVs:

```
oc get pv
```

## Sample Output

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS
etcd-asb-volume	10G	RWO	Retain	Bound
logging-volume	10Gi	RWO	Retain	Bound
metrics-volume	10Gi	RWO	Retain	Bound
prometheus-alertbuffer-volume	10Gi	RWO	Retain	Bound
prometheus-alertmanager-volume	10Gi	RWO	Retain	Bound
prometheus-volume	10Gi	RWO	Retain	Bound
pv1	5Gi	RWO	Recycle	Available
pv10	5Gi	RWO	Recycle	Available
pv11	5Gi	RWO	Recycle	Available
pv12	5Gi	RWO	Recycle	Available
pv13	5Gi	RWO	Recycle	Available
[...]				

## 5. Test Cluster Setup

In order to verify that your cluster is set up correctly, it is always a good idea to execute a test. The test needs to verify that your PVs are working, you can build an application, the application image can be pushed to the registry, your pod is running, and the routers can route traffic to the application.

You can do a simple test using the `nodejs-mongo-persistent` template. This template creates a MongoDB database with persistent storage. It also builds a Node.js application from source code and pushes it into the registry. Finally, when the application is running the route can be used to validate that the routers are working correctly.

Make sure to create a new project for this test—it is generally a bad idea to deploy an application into the `default` project on the OpenShift cluster.

1. Create a new project:

```
oc new-project smoke-test
```

### Sample Output

```
Now using project "smoke-test" on server
"https://loadbalancer.GUID.internal:443".
```



You can add applications to this project with the `new-app` command. For example, try the following to build a new example application in Ruby:

```
oc new-app centos/ruby-22-  
centos7~https://github.com/openshift/ruby-ex.git
```

2. Create the Node.js application:

```
oc new-app nodejs-mongo-persistent
```

### Sample Output

--> Deploying template "openshift/nodejs-mongo-persistent" to project smoke-test

Node.js + MongoDB

-----

An example Node.js application with a MongoDB database. For more information about using this template, including OpenShift considerations, see <https://github.com/sclorg/nodejs-ex/blob/master/README.md>.

The following service(s) have been created in your project: nodejs-mongo-persistent, mongodb.

For more information about using this template, including OpenShift considerations, see <https://github.com/sclorg/nodejs-ex/blob/master/README.md>.

\* With parameters:

- \* Name=nodejs-mongo-persistent
- \* Namespace=openshift
- \* Version of NodeJS Image=8
- \* Version of MongoDB Image=3.4
- \* Memory Limit=512Mi
- \* Memory Limit (MongoDB)=512Mi
- \* Volume Capacity=1Gi
- \* Git Repository URL=<https://github.com/sclorg/nodejs-ex.git>
- \* Git Reference=
- \* Context Directory=
- \* Application Hostname=
- \* GitHub Webhook Secret=vJSPNfCK6P5p4kGcMIpyFagblGo8uCw1BJ3b0jld #

generated

- \* Generic Webhook Secret=j3nwYlRuVhbHe4LNEli0rkNV4mTV0RQeRoySewDN

# generated

- \* Database Service Name=mongodb
- \* MongoDB Username=user6I0 # generated
- \* MongoDB Password=PK2gdyNVHybN0mQP # generated
- \* Database Name=sampledb
- \* Database Administrator Password=1AuysNJpyCcyDYgC # generated
- \* Custom NPM Mirror URL=



```
--> Creating resources ...
secret "nodejs-mongo-persistent" created
service "nodejs-mongo-persistent" created
route.route.openshift.io "nodejs-mongo-persistent" created
imagestream.image.openshift.io "nodejs-mongo-persistent" created
buildconfig.build.openshift.io "nodejs-mongo-persistent" created
deploymentconfig.apps.openshift.io "nodejs-mongo-persistent" created
persistentvolumeclaim "mongodb" created
service "mongodb" created
deploymentconfig.apps.openshift.io "mongodb" created
persistentvolumeclaim "mongodb" created
service "mongodb" created
deploymentconfig.apps.openshift.io "mongodb" created
--> Success
Access your application via route 'nodejs-mongo-persistent-smoke-
test.apps.GUID.example.opentlc.com'
Build scheduled, use 'oc logs -f bc/nodejs-mongo-persistent' to track
its progress.
Run 'oc status' to view your app.
```

### 3. Examine the project status:

```
oc status
```

### Sample Output

In project smoke-test on server https://loadbalancer.GUID.internal:443

svc/mongodb - 172.30.76.184:27017

dc/mongodb deploys openshift/mongodb:3.4

deployment #1 *pending 1 second ago*

http://nodejs-mongo-persistent-smoke-test.apps.GUID.example.opentlc.com  
(svc/nodejs-mongo-persistent)

dc/nodejs-mongo-persistent deploys istag/nodejs-mongo-persistent:latest

<-

bc/nodejs-mongo-persistent source builds

https://github.com/sclorg/nodejs-ex.git on openshift/nodejs:8

build #1 *pending for 2 seconds*

deployment #1 *waiting on image or update*

1 info identified, use 'oc status --suggest' to see details.

4. Watch the build and **mongodb** pods:

oc get pods -w

## Sample Output

NAME	READY	STATUS	RESTARTS
AGE			
mongodb-1-deploy	0/1	ContainerCreating	0
3s			
nodejs-mongo-persistent-1-build	0/1	Init:0/2	0
4s			
nodejs-mongo-persistent-1-build	0/1	Init:0/2	0
7s			
mongodb-1-6z7f6	0/1	Pending	0
1s			
mongodb-1-6z7f6	0/1	Pending	0
1s			
mongodb-1-deploy	1/1	Running	0
8s			
mongodb-1-6z7f6	0/1	ContainerCreating	0
1s			
nodejs-mongo-persistent-1-build	0/1	Init:1/2	0
14s			
nodejs-mongo-persistent-1-build	0/1	PodInitializing	0
15s			
nodejs-mongo-persistent-1-build	1/1	Running	0
17s			
nodejs-mongo-persistent-1-build	0/1	Completed	0
53s			
mongodb-1-6z7f6	0/1	Running	0
45s			
nodejs-mongo-persistent-1-deploy	0/1	Pending	0
0s			
nodejs-mongo-persistent-1-deploy	0/1	Pending	0
0s			
nodejs-mongo-persistent-1-deploy	0/1	ContainerCreating	0
0s			
nodejs-mongo-persistent-1-skw82	0/1	Pending	0
0s			
nodejs-mongo-persistent-1-skw82	0/1	Pending	0
0s			
nodejs-mongo-persistent-1-skw82	0/1	ContainerCreating	0
0s			
nodejs-mongo-persistent-1-deploy	1/1	Running	0
3s			
mongodb-1-6z7f6	1/1	Running	0
52s			
mongodb-1-deploy	0/1	Completed	0
59s			
mongodb-1-deploy	0/1	Terminating	0
59s			
mongodb-1-deploy	0/1	Terminating	0
59s			
nodejs-mongo-persistent-1-skw82	0/1	Running	0
18s			
nodejs-mongo-persistent-1-skw82	1/1	Running	0
25s			
nodejs-mongo-persistent-1-deploy	0/1	Completed	0
28s			
nodejs-mongo-persistent-1-deploy	0/1	Terminating	0
28s			
nodejs-mongo-persistent-1-deploy	0/1	Terminating	0
28s			

5. Verify that the storage was set up correctly:

```
oc get pvc
```

### Sample Output

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
mongodb	Bound	vol151	1Gi	RWO		22m

6. When the build is finished and the `mongodb` and `nodejs-mongo-persistent` pods are running—showing **1/1** in the **READY** column—retrieve the route:

```
oc get route
```

### Sample Output

NAME	HOST/PORT
nodejs-mongo-persistent	nodejs-mongo-persistent-smoke-
test.apps.GUID.example.opentlc.com	nodejs-mongo-persistent
<all>	None

7. In a web browser, navigate to the route (replacing `GUID` with your specific GUID).
- Expect your application to be running and showing a database connection in the bottom right portion of the page.
8. Delete your `smoke-test` project:

```
oc delete project smoke-test
```

This completes the lab. You can be reasonably certain that most aspects of your cluster work satisfactorily.

Build Version: 2.12 : Last updated 2019-03-27 09:54:17 EDT