OpenShift Enterprise 3.0

Installation and Configuration

OpenShift Enterprise 3.0 Installation and Configuration

Last Updated: 2019-03-14

OpenShift Enterprise 3.0 Installation and Configuration

OpenShift Enterprise 3.0 Installation and Configuration

Legal Notice

Abstract

OpenShift Installation and Configuration topics cover the basics of installing and configuring OpenShift in your environment. Use these topics for the one-time tasks required to get OpenShift up and running.

Table of Contents

CHAPTER 1. OVERVIEW

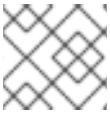
OpenShift Installation and Configuration topics cover the basics of installing and configuring OpenShift in your environment. Use these topics for the one-time tasks required to get OpenShift up and running. For day to day tasks, see Administration.

CHAPTER 2. INSTALLING

2.1. OVERVIEW

The quick installation method allows you to use an interactive CLI utility to install OpenShift across a set of hosts. The utility is a self-contained wrapper intended for usage on a Red Hat Enterprise Linux 7 host.

For production environments, a reference configuration implemented using Ansible playbooks is available as the advanced installation method.



NOTE

Before beginning either installation method, start with the Prerequisites topic.

2.2. PREREQUISITES

2.2.1. Overview

OpenShift infrastructure components can be installed across multiple hosts. The following sections outline the system requirements and instructions for preparing your environment and hosts before installing OpenShift.

2.2.2. System Requirements

You must have an active OpenShift Enterprise subscription on your Red Hat account to proceed. If you do not, contact your sales representative for more information.

The system requirements vary per host type:

Masters	• Physical or virtual system, or an instance running on a public or private IaaS. • Base OS: Red Hat Enterprise Linux (RHEL) 7.1 with "Minimal" installation option • 2 vCPU • Minimum 8 GB RAM • Minimum 30 GB hard disk space

Nodes	• Physical or virtual system, or an instance running on a public or private IaaS. • Base OS: Red Hat Enterprise Linux (RHEL) 7.1 with "Minimal" installation option • Docker 1.6.2 or later • 1 vCPU • Minimum 8 GB RAM • Minimum 15 GB hard disk space • An additional minimum 15 GB unallocated space to be configured using **docker-storage-setup**; see Configuring Docker Storage below.

Persistent Storage

The Kubernetes persistent volume framework allows you to provision an OpenShift cluster with persistent storage using networked storage available in your environment. This can be done after completing the initial OpenShift installation depending on your application needs, giving users a way to request those resources without having any knowledge of the underlying infrastructure.

Currently NFS is fully supported, however other options are available as Technology Preview.

Configuring Core Usage

By default, OpenShift masters and nodes use all available cores in the system they run on. You can choose the number of cores you want OpenShift to use by setting the **GOMAXPROCS** environment variable.

For example, run the following before starting the server to make OpenShift only run on one core:

```
# export GOMAXPROCS=1
```

Security Warning

OpenShift runs Docker containers on your hosts, and in some cases, such as build operations and the registry service, it does so using privileged containers. Furthermore, those containers access your host's Docker daemon and perform **`docker build`** and **`docker push`** operations. As such, you should be aware of the inherent security risks associated with performing **`docker run`** operations on arbitrary images as they effectively have root access.

For more information, see these articles:

- http://opensource.com/business/14/7/docker-security-selinux
- https://docs.docker.com/articles/security/

To address these risks, OpenShift uses security context constraints that control the actions that pods can perform and what it has the ability to access.

2.2.3. Environment Requirements

DNS

A wildcard for a DNS zone must ultimately resolve to the IP address of the OpenShift router.

For example, create a wildcard DNS entry for **cloudapps**, or something similar, that has a low TTL and points to the public IP address of the host where the router will be deployed:

```
*.cloudapps.example.com. 300 IN  A 192.168.133.2
```

In almost all cases, when referencing VMs you must use host names, and the host names that you use must match the output of the **`hostname -f`** command on each node.

Networking

A shared network must exist between the master and node hosts.

If you plan to configure multiple masters for high-availability using the advanced installation method, you must also select an IP to be configured as your virtual IP (VIP) during the installation process. The IP that you select must be routable between all of your nodes, and if you configure using a FQDN it should resolve on all nodes. The VIP is then managed by Pacemaker.

Git

You must have either Internet access and a GitHub account, or read and write access to an internal, HTTP-based Git server.

2.2.3.1. Persistent Storage

The Kubernetes persistent volume framework allows you to provision an OpenShift cluster with persistent storage using networked storage available in your environment. This can be done after completing the initial OpenShift installation depending on your application needs, giving users a way to request those resources without having any knowledge of the underlying infrastructure.

2.2.3.2. SELinux

Security-Enhanced Linux (SELinux) must be enabled on all of the servers before installing OpenShift or the installer will fail. Also, configure **`SELINUXTYPE=targeted`** in the ***/etc/selinux/config*** file:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes
are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

2.2.4. Host Preparation

Before installing OpenShift, you must first prepare each host per the following.

2.2.4.1. Software Prerequisites

Installing Red Hat Enterprise Linux 7

A base installation of Red Hat Enterprise Linux (RHEL) 7.1 is required for master or node hosts. See the Red Hat Enterprise Linux 7.1 Installation Guide for more information.

Registering the Hosts

Each host must be registered using Red Hat Subscription Manager (RHSM) and have an active OpenShift Enterprise subscription attached to access the required packages.

1. On each host, register with RHSM:

```
# subscription-manager register --username=<user_name> --password=
<password>
```

2. List the available subscriptions:

```
# subscription-manager list --available
```

3. In the output for the previous command, find the pool ID for an OpenShift Enterprise subscription and attach it:

```
# subscription-manager attach --pool=<pool_id>
```

4. Disable all repositories and enable only the required ones:

```
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.0-rpms"
```

5. If you plan to configure multiple masters for high-availability using the advanced installation method, you must also enable the High Availability Add-on for Red Hat Enterprise Linux repository:

```
# subscription-manager repos \
    --enable="rhel-ha-for-rhel-7-server-rpms"
```

Managing Base Packages

1. Install the following packages:

```
# yum install wget git net-tools bind-utils iptables-services
bridge-utils bash-completion
```

2. If you plan to use the quick installation method, you must also install the GNU Compiler Collection (**gcc**) and Python Virtual Environment (**python-virtualenv**) packages:

```
# yum install gcc python-virtualenv
```

3. Update the system to the latest packages:

```
# yum update
```

4. Install the following package, which provides OpenShift utilities and pulls in other tools required by the quick and advanced installation methods, such as Ansible and related configuration files:

```
# yum install atomic-openshift-utils
```

Installing Docker

Docker version 1.6.2 or later from the **rhel-7-server-ose-3.0-rpms** repository must be installed and running on master and node hosts before installing OpenShift.

1. Install Docker:

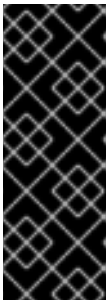
```
# yum install docker
```

2. Edit the ***/etc/sysconfig/docker*** file and add **`--insecure-registry 172.30.0.0/16`** to the **`OPTIONS`** parameter. For example:

```
OPTIONS='--selinux-enabled --insecure-registry 172.30.0.0/16'
```

The **`--insecure-registry`** option instructs the Docker daemon to trust any Docker registry on the indicated subnet, rather than requiring a certificate.

After installing OpenShift, you can choose to secure the integrated Docker registry, which involves adjusting the **`--insecure-registry`** option accordingly.



IMPORTANT

172.30.0.0/16 is the default value of the **`servicesSubnet`** variable in the ***master-config.yaml*** file. If this has changed, then the **`--insecure-registry`** value in the above step should be adjusted to match, as it is indicating the subnet for the registry to use. Note that the **`openshift_master_portal_net`** variable can be set in the Ansible inventory file and used during the advanced installation method to modify the **`servicesSubnet`** variable.

2.2.4.2. Configuring Docker Storage

Docker containers and the images they are created from are stored in Docker's storage back end. This storage is ephemeral and separate from any persistent storage allocated to meet the needs of your applications.

The default storage back end is a thin pool on loopback devices which is not supported for production use and only appropriate for proof of concept environments. For production environments, you must create a thin pool logical volume and re-configure Docker to use that volume.

You can use the **docker-storage-setup** script included with Docker to create a thin pool device and configure Docker's storage driver. This can be done after installing Docker and should be done before creating images or containers. The script reads configuration options from the ***/etc/sysconfig/docker-storage-setup*** file and supports three options for creating the logical volume:

- **Option A)** Use an additional block device.
- **Option B)** Use an existing, specified volume group.

- **Option C)** Use the remaining free space from the volume group where your root file system is located.

Option A is the most robust option, however it requires adding an additional block device to your host before configuring Docker storage. Options B and C both require leaving free space available when provisioning your host.

NOTE

See the Managing Storage with Docker Formatted Containers on Red Hat Enterprise Linux and Red Hat Enterprise Linux Atomic Host Knowledgebase article for more details about **docker-storage-setup** and basic instructions on storage management in Red Hat Enterprise Linux 7.

1. Create the **docker-pool** volume using one of the following three options:

   - **Option A) Use an additional block device.**
     In ***/etc/sysconfig/docker-storage-setup***, set **DEVS** to the path of the block device you wish to use. Set **VG** to the volume group name you wish to create; **docker-vg** is a reasonable choice. For example:

     ```
     # cat <<EOF > /etc/sysconfig/docker-storage-setup
     DEVS=/dev/vdc
     VG=docker-vg
     EOF
     ```

     Then run **docker-storage-setup** and review the output to ensure the **docker-pool** volume was created:

     ```
     # docker-storage-setup
     [5/1868]
     0
     Checking that no-one is using this disk right now ...
     OK

     Disk /dev/vdc: 31207 cylinders, 16 heads, 63 sectors/track
     sfdisk:  /dev/vdc: unrecognized partition table type

     Old situation:
     sfdisk: No partitions found

     New situation:
     Units: sectors of 512 bytes, counting from 0

        Device Boot    Start       End   #sectors  Id  System
     /dev/vdc1          2048  31457279   31455232  8e  Linux LVM
     /dev/vdc2             0         -          0   0  Empty
     /dev/vdc3             0         -          0   0  Empty
     /dev/vdc4             0         -          0   0  Empty
     Warning: partition 1 does not start at a cylinder boundary
     Warning: partition 1 does not end at a cylinder boundary
     Warning: no primary partition is marked bootable (active)
     This does not matter for LILO, but the DOS MBR will not boot this
     disk.
     Successfully wrote the new partition table
     ```

```
Re-reading the partition table ...

If you created or changed a DOS partition, /dev/foo7, say, then
use dd(1)
to zero the first 512 bytes:  dd if=/dev/zero of=/dev/foo7 bs=512
count=1
(See fdisk(8).)
  Physical volume "/dev/vdc1" successfully created
  Volume group "docker-vg" successfully created
  Rounding up size to full physical extent 16.00 MiB
  Logical volume "docker-poolmeta" created.
  Logical volume "docker-pool" created.
  WARNING: Converting logical volume docker-vg/docker-pool and
docker-vg/docker-poolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
  Converted docker-vg/docker-pool to thin pool.
  Logical volume "docker-pool" changed.
```

- **Option B) Use an existing, specified volume group.**
  In ***/etc/sysconfig/docker-storage-setup***, set **VG** to the desired volume group. For example:

```
# cat <<EOF > /etc/sysconfig/docker-storage-setup
VG=docker-vg
EOF
```

  Then run **docker-storage-setup** and review the output to ensure the **docker-pool** volume was created:

```
# docker-storage-setup
  Rounding up size to full physical extent 16.00 MiB
  Logical volume "docker-poolmeta" created.
  Logical volume "docker-pool" created.
  WARNING: Converting logical volume docker-vg/docker-pool and
docker-vg/docker-poolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
  Converted docker-vg/docker-pool to thin pool.
  Logical volume "docker-pool" changed.
```

- **Option C) Use the remaining free space from the volume group where your root file system is located.**
  Verify that the volume group where your root file system resides has the desired free space, then run **docker-storage-setup** and review the output to ensure the **docker-pool** volume was created:

```
# docker-storage-setup
  Rounding up size to full physical extent 32.00 MiB
  Logical volume "docker-poolmeta" created.
  Logical volume "docker-pool" created.
  WARNING: Converting logical volume rhel/docker-pool and
rhel/docker-poolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
  Converted rhel/docker-pool to thin pool.
  Logical volume "docker-pool" changed.
```

2. Verify your configuration. You should have a **dm.thinpooldev** value in the ***/etc/sysconfig/docker-storage*** file and a **docker-pool** logical volume:

```
# cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS=--storage-opt dm.fs=xfs --storage-opt
dm.thinpooldev=/dev/mapper/docker--vg-docker--pool

# lvs
  LV          VG   Attr       LSize  Pool Origin Data%  Meta%  Move
Log Cpy%Sync Convert
  docker-pool rhel twi-a-t---  9.29g             0.00   0.12
```

IMPORTANT

Before using Docker or OpenShift, verify that the **docker-pool** logical volume is large enough to meet your needs. The **docker-pool** volume should be 60% of the available volume group and will grow to fill the volume group via LVM monitoring.

3. Check if Docker is running:

```
# systemctl is-active docker
```

4. If Docker has not yet been started on the host, enable and start the service:

```
# systemctl enable docker
# systemctl start docker
```

If Docker is already running, re-initialize Docker:

WARNING

This will destroy any Docker containers or images currently on the host.

```
# systemctl stop docker
# rm -rf /var/lib/docker/*
# systemctl restart docker
```

If there is any content in ***/var/lib/docker/***, it must be deleted. Files will be present if Docker has been used prior to the installation of OpenShift.

Reconfiguring Docker Storage

Should you need to reconfigure Docker storage after having created the **docker-pool**, you should first remove the **docker-pool** logical volume. If you are using a dedicated volume group, you should also remove the volume group and any associated physical volumes before reconfiguring **docker-storage-setup** according to the instructions above.

See Logical Volume Manager Administration for more detailed information on LVM management.

2.2.5. Ensuring Host Access

The quick and advanced installation methods require a user that has access to all hosts. If you want to run the installer as a non-root user, passwordless **sudo** rights must be configured on each destination host.

For example, you can generate an SSH key on the host where you will invoke the installation process:

```
# ssh-keygen
```

Do **not** use a password.

An easy way to distribute your SSH keys is by using a **bash** loop:

```
# for host in master.example.com \
    node1.example.com \
    node2.example.com; \
    do ssh-copy-id -i ~/.ssh/id_rsa.pub $host; \
    done
```

Modify the host names in the above command according to your configuration.

2.2.6. What's Next?

Now that your environment and hosts are properly set up, you can install OpenShift Enterprise using the quick installation or advanced installation method.

2.3. QUICK INSTALLATION

2.3.1. Overview

The *quick installation* method allows you to use an interactive CLI utility to install OpenShift across a set of hosts. The installation utility can deploy OpenShift components on targeted hosts by installing RPMs.

This installation method is provided to make the installation experience easier by interactively gathering the data needed to run on each host. The utility is a self-contained wrapper intended for usage on a Red Hat Enterprise Linux (RHEL) 7 system.

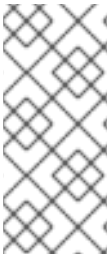
In addition to running interactive installations from scratch, the **`atomic-openshift-installer`** command can also be run or re-run using a predefined installation configuration file. This file can be used with the installation utility to:

- run an unattended installation,
- add nodes to an existing cluster,
- upgrade your cluster, or
- reinstall the OpenShift cluster completely.

Alternatively, you can use the advanced installation method for more complex environments.

2.3.2. Before You Begin

The installation utility allows you to install OpenShift master and node components on a defined set of hosts.



NOTE

By default, any hosts you designate as masters during the installation process are automatically also configured as nodes so that the masters are configured as part of the OpenShift SDN. The node component on the masters, however, are marked unschedulable, which blocks pods from being scheduled on it. After the installation, you can mark them schedulable if you want.

Before installing OpenShift, you must first satisfy the prerequisites on your hosts, which includes verifying system and environment requirements and properly installing and configuring Docker. You must also be prepared to provide or validate the following information for each of your targeted hosts during the course of the installation:

- User name on the target host that should run the Ansible-based installation (can be root or non-root)
- Host name
- Whether to install components for master, node, or both
- Internal and external IP addresses

After following the instructions in the Prerequisites topic, you can continue to running an interactive or unattended installation.

2.3.3. Running an Interactive Installation

NOTE

Ensure you have read through Before You Begin.

You can start the interactive installation by running:

```
$ atomic-openshift-installer install
```

Then follow the on-screen instructions to install a new OpenShift Enterprise cluster.

After it has finished, ensure that you back up the ***~/.config/openshift/installer.cfg.yml***installation configuration file that is created, as it is required if you later want to re-run the installation, add hosts to the cluster, or upgrade your cluster. Then, see What's Next for the next steps on configuring your OpenShift cluster.

2.3.4. Defining an Installation Configuration File

The installation utility can use a predefined installation configuration file, which contains information about your installation, individual hosts, and cluster. When running an interactive installation, an installation configuration file based on your answers is created for you in ***~/.config/openshift/installer.cfg.yml***. The file is created if you are instructed to exit the installation to manually modify the configuration or when the installation completes. You can also create the configuration file manually from scratch to perform an unattended installation.

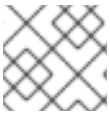
Example 2.1. Installation Configuration File Specification

```
version: v1 1
variant: openshift-enterprise 2
variant_version: 3.0 3
ansible_ssh_user: root 4
ansible_log_path: /tmp/ansible.log 5
hosts: 6
- ip: 10.0.0.1 7
  hostname: master-private.example.com 8
  public_ip: 24.222.0.1 9
  public_hostname: master.example.com 10
  master: true 11
  node: true 12
  connect_to: 24.222.0.1 13
- ip: 10.0.0.2
  hostname: node1-private.example.com
  public_ip: 24.222.0.2
  public_hostname: node1.example.com
  node: true
  connect_to: 10.0.0.2
- ip: 10.0.0.3
  hostname: node2-private.example.com
  public_ip: 24.222.0.3
  public_hostname: node2.example.com
  node: true
  connect_to: 10.0.0.3
```

1. The version of this installation configuration file.
2. The OpenShift variant to install. For OSE, set this to **`openshift-enterprise`**.
3. A valid version your selected variant. If not specified, this defaults to the newest version for the specified variant. For example: **`3.0`**.
4. Defines which user Ansible uses to SSH in to remote systems for gathering facts and for the installation. By default, this is the root user, but you can set it to any user that has **sudo** privileges.
5. Defines where the Ansible logs are stored. By default, this is the ***/tmp/ansible.log*** file.
6. Defines a list of the hosts onto which you want to install the OpenShift master and node components.
7. 8. Required. Allows the installer to connect to the system and gather facts before proceeding with the install.
9. 10. Required for unattended installations. If these details are not specified, then this information is pulled from the facts gathered by the installation utility, and you are asked to confirm the details. If undefined for an unattended installation, the installation fails.
11. 12. Determines the type of services that are installed. At least one of these must be set to **true** for the configuration file to be considered valid.

13 The IP address that Ansible attempts to connect to when installing, upgrading, or uninstalling the systems. If the configuration file was auto-generated, then this is the value you first enter for the host during that interactive install process.

2.3.5. Running an Unattended Installation



NOTE

Ensure you have read through the Before You Begin.

Unattended installations allow you to define your hosts and cluster configuration in an installation configuration file before running the installation utility so that you do not have to go through all of the interactive installation questions and answers. It also allows you to resume an interactive installation you may have left unfinished, and quickly get back to where you left off.

To run an unattended installation, first define an installation configuration file at ***~/.config/openshift/installer.cfg.yml***. Then, run the installation utility with the **-u** flag:

```
$ atomic-openshift-installer -u install
```

By default in interactive or unattended mode, the installation utility uses the configuration file located at ***~/.config/openshift/installer.cfg.yml*** if the file exists. If it does not exist, attempting to start an unattended installation fails. Alternatively, you can specify a different location for the configuration file using the **-c** option, but doing so will require you to specify the file location every time you run the installation:

```
$ atomic-openshift-installer -u -c </path/to/file> install
```

After the unattended installation finishes, ensure that you back up the ***~/.config/openshift/installer.cfg.yml*** file that was used, as it is required if you later want to re-run the installation, add hosts to the cluster, or upgrade your cluster. Then, see What's Next for the next steps on configuring your OpenShift cluster.

2.3.6. Adding Nodes or Reinstalling the Cluster

Whether you began the process using an interactive or unattended installation, you can re-run the installation as long as you have an installation configuration file at ***~/.config/openshift/installer.cfg.yml*** (or specify its location with the **-c** option).

To re-run an installation, use the **install** subcommand again in interactive or unattended mode:

```
$ atomic-openshift-installer install
```

The installer will detect your installed environment and allow you to either add an additional node or perform a clean install:

```
Gathering information from hosts...
Installed environment detected.
By default the installer only adds new nodes to an installed environment.
Do you want to (1) only add additional nodes or (2) perform a clean
install?:
```

Choose one of the options and follow the on-screen instructions to complete your desired task.

2.3.7. Uninstalling OpenShift

You can uninstall OpenShift using the installation utility by running:

```
$ atomic-openshift-installer uninstall
```

2.3.8. What's Next?

Now that you have a working OpenShift Enterprise instance, you can:

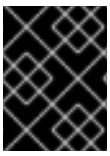
- Configure authentication; by default, authentication is set to Deny All.
- Deploy an integrated Docker registry.
- Deploy a router.

2.4. ADVANCED INSTALLATION

2.4.1. Overview

For production environments, a reference configuration implemented using Ansible playbooks is available as the *advanced installation* method for installing OpenShift hosts. Familiarity with Ansible is assumed, however you can use this configuration as a reference to create your own implementation using the configuration management tool of your choosing.

Alternatively, you can use the quick installation method for trial installations.



IMPORTANT

Running Ansible playbooks with the **`--tags`** or **`--check`** options is not supported by Red Hat.

2.4.2. Before You Begin

Before installing OpenShift, you must first see the Prerequisites topic to prepare your hosts, which includes verifying system and environment requirements per component type and properly installing and configuring Docker. It also includes installing Ansible version 1.8.4 or later, as the advanced installation method is based on Ansible playbooks and as such requires directly invoking Ansible.

After following the instructions in the Prerequisites topic, you can continue to Configuring Ansible.

2.4.3. Configuring Ansible

The ***/etc/ansible/hosts*** file is Ansible's inventory file for the playbook to use during the installation. The inventory file describes the configuration for your OpenShift cluster. You must replace the default contents of the file with your desired configuration.

The following sections describe commonly-used variables to set in your inventory file during an advanced installation, followed by example inventory files you can use as a starting point for your installation. The examples describe various environment topographies. You can choose an example that

matches your requirements, modify it to match your own environment, and use it as your inventory file when running the Ansible installer.

NOTE

Before running the Ansible installer, any hosts you intend to designate as masters during the installation process should also be configured as unschedulable nodes, in order to configure the masters as part of the OpenShift SDN.

Configuring Host Variables

To assign environment variables to hosts during the Ansible install, indicate the desired variables in the ***/etc/ansible/hosts*** file after the host entry in the **[masters]** or **[nodes]** sections. For example:

```
[masters]
ec2-52-6-179-239.compute-1.amazonaws.com openshift_public_hostname=ose3-
master.public.example.com
```

The following table describes variables for use with the Ansible installer that can be assigned to individual host entries:

Table 2.1. Host Variables

Variable	Purpose
openshift_hostname	This variable overrides the internal cluster host name for the system. Use this when the system's default IP address does not resolve to the system host name.
openshift_public_hostname	This variable overrides the system's public host name. Use this for cloud installations, or for hosts on networks using a network address translation (NAT).
openshift_ip	This variable overrides the cluster internal IP address for the system. Use this when using an interface that is not configured with the default route.
openshift_public_ip	This variable overrides the system's public IP address. Use this for cloud installations, or for hosts on networks using a network address translation (NAT).

Configuring Cluster Variables

To assign environment variables during the Ansible install that apply more globally to your OpenShift cluster overall, indicate the desired variables in the ***/etc/ansible/hosts*** file on separate, single lines within the **[OSEv3:vars]** section. For example:

```
[OSEv3:vars]

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider',
```

```
'filename': '/etc/openshift/openshift-passwd'}]

osm_default_subdomain=apps.test.example.com
```

The following table describes variables for use with the Ansible installer that can be assigned cluster-wide:

Table 2.2. Cluster Variables

Variable	Purpose
ansible_ssh_user	This variable sets the SSH user for the installer to use and defaults to **root**. This user should allow SSH-based authentication without requiring a password. If using SSH key-based authentication, then the key should be managed by an SSH agent.
ansible_sudo	If **ansible_ssh_user** is not **root**, this variable must be set to **true** and the user must be configured for passwordless **sudo**.
openshift_master_cluster_hostname	This variable overrides the host name for the cluster, which defaults to the host name of the master.
openshift_master_cluster_public_hostname	This variable overrides the public host name for the cluster, which defaults to the host name of the master. If you use an external load balancer, specify the address of the external load balancer. For example: ---- openshift_master_cluster_public_hostname=openshift-ansible.public.example.com ----
openshift_master_identity_providers	This variable overrides the identity provider, which defaults to Deny All.
osm_default_subdomain	This variable overrides the default subdomain to use for exposed routes.
osm_default_node_selector	This variable overrides the node selector that projects will use by default when placing pods.
osm_cluster_network_cidr	This variable overrides the SDN cluster network CIDR block. This is the network from which pod IPs are assigned. This network block should be a private block and should not conflict with existing network blocks in your infrastructure that pods may require access to. Defaults to 10.1.0.0/16 and **can not** be re-configured after deployment.
osm_host_subnet_length	This variable specifies the size of the per host subnet allocated for pod IPs by OpenShift SDN. Defaults to /8 which means that from the 10.1.0.0/16 cluster network a subnet of size /24 is allocated to each host (i.e., 10.1.0.0/24, 10.1.1.0/24, 10.1.2.0/24, and so on). This **can not** be re-configured after deployment.

Configuring Node Host Labels

You can assign labels to node hosts during the Ansible install by configuring the ***/etc/ansible/hosts*** file. Labels are useful for determining the placement of pods onto nodes using the scheduler.

To assign labels to a node host during an Ansible install, use the **`openshift_node_labels`** variable with the desired labels added to the desired node host entry in the **[nodes]** section. For example:

```
[nodes]
node1.example.com openshift_node_labels="{'region': 'primary', 'zone':
'east'}"
```

2.4.3.1. Single Master and Multiple Nodes

The following table describes an example environment for a single master and two nodes:

Host Name	Infrastructure Component to Install
master.example.com	Master and node
node1.example.com	Node
node2.example.com	

You can see these example hosts present in the **[masters]** and **[nodes]** sections of the following example inventory file:

Example 2.2. Single Master and Multiple Nodes Inventory File

```
# Create an OSEv3 group that contains the masters and nodes groups
[OSEv3:children]
masters
nodes

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
# SSH user, this user should allow ssh based auth without requiring a
password
ansible_ssh_user=root

# If ansible_ssh_user is not root, ansible_sudo must be set to true
#ansible_sudo=true

product_type=openshift
deployment_type=enterprise

# uncomment the following to enable htpasswd authentication; defaults to
DenyAllPasswordIdentityProvider
#openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider',
'filename': '/etc/openshift/openshift-passwd'}]

# host group for masters
[masters]
```

```
master.example.com

# host group for nodes, includes region info
[nodes]
master.example.com openshift_node_labels="{'region': 'infra', 'zone':
'default'}"
node1.example.com openshift_node_labels="{'region': 'primary', 'zone':
'east'}"
node2.example.com openshift_node_labels="{'region': 'primary', 'zone':
'west'}"
```

To use this example, modify the file to match your environment and specifications, and save it as ***/etc/ansible/hosts***.

NOTE

Moving from a single master cluster to multiple masters after installation is not supported.

2.4.3.2. Single Master, Multiple etcd, and Multiple Nodes

The following table describes an example environment for a single master, three **etcd** hosts, and two nodes:

Host Name	Infrastructure Component to Install
master.example.com	Master and node
etcd1.example.com	etcd
etcd2.example.com	
etcd3.example.com	
node1.example.com	Node
node2.example.com	

NOTE

When specifying multiple **etcd** hosts, external **etcd** is installed and configured. Clustering of OpenShift's embedded **etcd** is not supported. Also, moving from a single master cluster to multiple masters after installation is not supported.

You can see these example hosts present in the **[masters]**, **[nodes]**, and **[etcd]** sections of the following example inventory file:

Example 2.3. Single Master, Multiple etcd, and Multiple Nodes Inventory File

```
# Create an OSEv3 group that contains the masters and nodes groups
```

```
[OSEv3:children]
masters
nodes
etcd

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
product_type=openshift
deployment_type=enterprise

# uncomment the following to enable htpasswd authentication; defaults to
DenyAllPasswordIdentityProvider
#openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider',
'filename': '/etc/openshift/openshift-passwd'}]

# host group for masters
[masters]
master.example.com

# host group for etcd
[etcd]
etcd1.example.com
etcd2.example.com
etcd3.example.com

# host group for nodes, includes region info
[nodes]
master.example.com openshift_node_labels="{'region': 'infra', 'zone':
'default'}"
node1.example.com openshift_node_labels="{'region': 'primary', 'zone':
'east'}"
node2.example.com openshift_node_labels="{'region': 'primary', 'zone':
'west'}"
```

To use this example, modify the file to match your environment and specifications, and save it as ***/etc/ansible/hosts***.

2.4.3.3. Multiple Masters, Multiple etcd, and Multiple Nodes

The following describes an example environment for three masters, three **etcd** hosts, and two nodes:

Host Name	Infrastructure Component to Install
master1.example.com	Master (clustered using Pacemaker) and node
master2.example.com	
master3.example.com	

Host Name	Infrastructure Component to Install
etcd1.example.com	etcd
etcd2.example.com	
etcd3.example.com	
node1.example.com	Node
node2.example.com	

NOTE

When specifying multiple **etcd** hosts, external **etcd** is installed and configured. Clustering of OpenShift's embedded **etcd** is not supported.

You can see these example hosts present in the **[masters]**, **[nodes]**, and **[etcd]** sections of the following example inventory file:

Example 2.4. Multiple Masters, Multiple etcd, and Multiple Nodes Inventory File

```
# Create an OSEv3 group that contains the masters and nodes groups
[OSEv3:children]
masters
nodes
etcd

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
product_type=openshift
deployment_type=enterprise

# uncomment the following to enable htpasswd authentication; defaults to
DenyAllPasswordIdentityProvider
# openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/openshift/openshift-passwd'}]

# master cluster ha variables using pacemaker or RHEL HA
openshift_master_cluster_method=pacemaker
openshift_master_cluster_password=openshift_cluster
openshift_master_cluster_vip=192.168.133.25
openshift_master_cluster_public_vip=192.168.133.25
openshift_master_cluster_hostname=openshift-master.example.com
openshift_master_cluster_public_hostname=openshift-master.example.com


# host group for masters
```

```
[masters]
master1.example.com
master2.example.com
master3.example.com

# host group for etcd
[etcd]
etcd1.example.com
etcd2.example.com
etcd3.example.com

# host group for nodes, includes region info
[nodes]
master[1:3].example.com openshift_node_labels="{'region': 'infra',
'zone': 'default'}"
node1.example.com openshift_node_labels="{'region': 'primary', 'zone':
'east'}"
node2.example.com openshift_node_labels="{'region': 'primary', 'zone':
'west'}"
```

To use this example, modify the file to match your environment and specifications, and save it as ***/etc/ansible/hosts***.

Note the following when using this configuration:

- Installing multiple masters requires that you configure a fencing device after running the installer.
- When specifying multiple masters, the installer handles creating and starting the high availability (HA) cluster. If during that process the **`pcs status`** command indicates that an HA cluster already exists, the installer skips HA cluster configuration.

NOTE

Moving from a single master cluster to multiple masters after installation is not supported.

2.4.4. Running the Ansible Installer

After you've configured Ansible by defining an inventory file in ***/etc/ansible/hosts***, you can run the Ansible installer:

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/config.yml
```

If for any reason the installation fails, before re-running the installer, see Known Issues to check for any specific instructions or workarounds.

2.4.5. Configuring Fencing

If you installed OpenShift using a configuration for multiple masters, you must configure a fencing device. See Fencing: Configuring STONITH in the High Availability Add-on for Red Hat Enterprise Linux documentation for instructions, then continue to Verifying the Installation.

2.4.6. Verifying the Installation

After the installer completes, you can verify that the master is started and nodes are registered and reporting in **Ready** status by running the following as **root**:

```
# oc get nodes

NAME                      LABELS
STATUS
master.example.com
kubernetes.io/hostname=master.example.com,region=infra,zone=default
Ready,SchedulingDisabled
node1.example.com
kubernetes.io/hostname=node1.example.com,region=primary,zone=east
Ready
node2.example.com
kubernetes.io/hostname=node2.example.com,region=primary,zone=west
Ready
```

Multiple etcd Hosts

If you installed multiple **etcd** hosts:

1. On a etcd host, verify the **etcd** cluster health, substituting for the FQDNs of your **etcd** hosts in the following:

```
# etcdctl -C \

https://etcd1.example.com:2379,https://etcd2.example.com:2379,https:
//etcd3.example.com:2379 \
    --ca-file=/etc/openshift/master/master.etcd-ca.crt \
    --cert-file=/etc/openshift/master/master.etcd-client.crt \
    --key-file=/etc/openshift/master/master.etcd-client.key cluster-
health
```

2. Also verify the member list is correct:

```
# etcdctl -C \

https://etcd1.example.com:2379,https://etcd2.example.com:2379,https:
//etcd3.example.com:2379 \
    --ca-file=/etc/openshift/master/master.etcd-ca.crt \
    --cert-file=/etc/openshift/master/master.etcd-client.crt \
    --key-file=/etc/openshift/master/master.etcd-client.key member
list
```

Multiple Masters

If you installed multiple masters:

1. On a master host, determine which host is currently running as the active master:

```
# pcs status
```

2. After determining the active master, put the specified host into standby mode:

```
# pcs cluster standby <host1_name>
```

3. Verify the master is now running on another host:

```
# pcs status
```

4. After verifying the master is running on another node, re-enable the host on standby for normal operation by running:

```
# pcs cluster unstandby <host1_name>
```

Red Hat recommends that you also verify your installation by consulting the High Availability Add-on for Red Hat Enterprise Linux documentation.

2.4.7. Adding Nodes to an Existing Cluster

After your cluster is installed, you can install additional nodes (including masters) and add them to your cluster by running the ***scaleup.yml*** playbook. This playbook queries the master, generates and distributes new certificates for the new nodes, then runs the configuration playbooks on the new nodes only.

This process is similar to re-running the installer in the quick installation method to add nodes, however you have more configuration options available when using the advanced method and running the playbooks directly.

You must have an existing inventory file (for example, ***/etc/ansible/hosts***) that is representative of your current cluster configuration in order to run the ***scaleup.yml*** playbook. If you previously used the **`atomic-openshift-installer`** command to run your installation, you can check ***~/.config/openshift/.ansible/hosts*** for the last inventory file that the installer generated and use or modify that as needed as your inventory file. You must then specify the file location with **`-i`** when calling **`ansible-playbook`** later.

IMPORTANT

The recommended maximum number of nodes is 300.

To add nodes to an existing cluster:

1. Ensure you have the latest playbooks by updating the **atomic-openshift-utils** package:

```
# yum update atomic-openshift-utils
```

2. Edit your ***/etc/ansible/hosts*** file and add **new_nodes** to the **[OSEv3:children]** section:

```
[OSEv3:children]
masters
nodes
new_nodes
```

3. Then, create a **[new_nodes]** section much like the existing **[nodes]** section, specifying host information for any new nodes you want to add. For example:

```
[nodes]
master[1:3].example.com openshift_node_labels="{'region': 'infra',
```

```
'zone': 'default'}"
node1.example.com openshift_node_labels="{'region': 'primary',
'zone': 'east'}"
node2.example.com openshift_node_labels="{'region': 'primary',
'zone': 'west'}"

[new_nodes]
node3.example.com openshift_node_labels="{'region': 'primary',
'zone': 'west'}"
```

See Configuring Host Variables for more options.

4. Now run the ***scaleup.yml*** playbook. If your inventory file is located somewhere other than the default */**etc**/**ansible**/**hosts***, specify the location with the **`-i option`**:
   For additional nodes:

```
# ansible-playbook [-i /path/to/file] \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
cluster/scaleup.yml
```

   For additional masters:

```
# ansible-playbook [-i /path/to/file] \
    usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
master/scaleup.yml
```

5. After the playbook completes successfully, verify the installation.

6. Finally, move any hosts you had defined in the **[new_nodes]** section up into the **[nodes]** section (but leave the **[new_nodes]** section definition itself in place) so that subsequent runs using this inventory file are aware of the nodes but do not handle them as new nodes. For example:

```
[nodes]
master[1:3].example.com openshift_node_labels="{'region': 'infra',
'zone': 'default'}"
node1.example.com openshift_node_labels="{'region': 'primary',
'zone': 'east'}"
node2.example.com openshift_node_labels="{'region': 'primary',
'zone': 'west'}"
node3.example.com openshift_node_labels="{'region': 'primary',
'zone': 'west'}"

[new_nodes]
```

2.4.8. Known Issues

The following are known issues for specified installation configurations.

Multiple Masters

- On failover, it is possible for the controller manager to overcorrect, which causes the system to run more pods than what was intended. However, this is a transient event and the system does correct itself over time. See https://github.com/GoogleCloudPlatform/kubernetes/issues/10030 for details.

- On failure of the Ansible installer, you must start from a clean operating system installation. If you are using virtual machines, start from a fresh image. If you are use bare metal machines:

  1. Run the following on a master host:

  ```
  # pcs cluster destroy --all
  ```

  2. Then, run the following on all node hosts:

  ```
  # yum -y remove openshift openshift-* etcd docker

  # rm -rf /etc/openshift /var/lib/openshift /etc/etcd \
      /var/lib/etcd /etc/sysconfig/openshift*
  /etc/sysconfig/docker* \
      /root/.kube/config /etc/ansible/facts.d /usr/share/openshift
  ```

2.4.9. What's Next?

Now that you have a working OpenShift instance, you can:

- Configure authentication; by default, authentication is set to Deny All.
- Deploy an integrated Docker registry.
- Deploy a router.
- Populate your OpenShift installation with a useful set of Red Hat-provided image streams and templates.

2.5. DEPLOYING A DOCKER REGISTRY

2.5.1. Overview

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.

2.5.2. Deploying the Registry

To deploy the integrated Docker registry, use the **`oadm registry`** command as a user with cluster administrator privileges. For example:

```
$ oadm registry --config=/etc/openshift/master/admin.kubeconfig \(1)
    --credentials=/etc/openshift/master/openshift-registry.kubeconfig \(2)
    --images='registry.access.redhat.com/openshift3/ose-
${component}:${version}' (3)
```

(1) **`--config`** is the path to the CLI configuration file for the cluster administrator.

(2) **`--credentials`** is the path to the CLI configuration file for the **openshift-registry**.

(3) Required to pull the correct image for OpenShift Enterprise.

This creates a service and a deployment configuration, both called **docker-registry**. Once deployed successfully, a pod is created with a name similar to **docker-registry-1-cpty9**.

To see a full list of options that you can specify when creating the registry:

```
$ oadm registry --help
```

2.5.2.1. Storage for the Registry

The registry stores Docker images and metadata. If you simply deploy a pod with the registry, it uses an ephemeral volume that is destroyed if the pod exits. Any images anyone has built or pushed into the registry would disappear.

2.5.2.1.1. Production Use

For production use, attach a remote volume or define and use persistent storage using NFS.

For example, to use an existing persistent volume claim:

```
$ oc volume deploymentconfigs/docker-registry --add --name=v1 -t pvc \
     --claim-name=<pvc_name> --overwrite
```

Or, to attach an existing NFS volume to the registry:

```
$ oc volume deploymentconfigs/docker-registry \
     --add --overwrite --name=registry-storage --mount-path=/registry \
     --source='{"nfs": { "server": "<fqdn>", "path": "/path/to/export"}}'
```

For non-production use, you can use the **`--mount-host=<path>`** option to specify a directory for the registry to use for persistent storage. The registry volume is then created as a host-mount at the specified **`<path>`**.

IMPORTANT

The **`--mount-host`** option mounts a directory from the node on which the registry container lives. If you scale up the **docker-registry** deployment configuration, it is possible that your registry pods and containers will run on different nodes, which can result in two or more registry containers, each with its own local storage. This will lead to unpredictable behavior, as subsequent requests to pull the same image repeatedly may not always succeed, depending on which container the request ultimately goes to.

The **`--mount-host`** option requires that the registry container run in privileged mode. This is automatically enabled when you specify **`--mount-host`**. However, not all pods are allowed to run privileged containers by default. If you still want to use this option:

1. Create a new service account in the **default** project for the registry to run as. The following example creates a service account named **registry**:

```
$ echo \
    '{"kind":"ServiceAccount","apiVersion":"v1","metadata":
{"name":"registry"}}' \
    | oc create -n default -f -
```

2. To add the new **registry** service account to the list of users allowed to run privileged containers:

   a. Edit the **privileged**security context constraint (SCC):

   ```
   $ oc edit scc privileged
   ```

   b. Add a line under **users** with the user name **system:serviceaccount:default:registry**.

3. Create the registry and specify that it use the new **registry** service account:

```
$ oadm registry --service-account=registry \
    --config=/etc/openshift/master/admin.kubeconfig \
    --credentials=/etc/openshift/master/openshift-
registry.kubeconfig \
    --images='registry.access.redhat.com/openshift3/ose-
${component}:${version}' \
    --mount-host=<path>
```

2.5.3. Viewing Logs

To view the logs for the Docker registry, run the **oc logs** indicating the desired pod:

```
$ oc logs docker-registry-1-da73t
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info
msg="redis not configured" instance.id=9ed6c43d-23ee-453f-9a4b-
031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info
msg="using inmemory layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-
031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info
msg="Using OpenShift Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info
msg="listening on :5000" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

2.5.4. File Storage

Tag and image metadata is stored in OpenShift, but the registry stores layer and signature data in a volume that is mounted into the registry container at ***/registry***. As **oc exec** does not work on privileged containers, to view a registry's contents you must manually SSH into the node housing the registry pod's container, then run **docker exec** on the container itself:

1. List the current pods to find the pod name of your Docker registry:

   ```
   # oc get pods
   ```

   Then, use **oc describe** to find the host name for the node running the container:

   ```
   # oc describe pod <pod_name>
   ```

2. Log into the desired node:

```
# ssh node.example.com
```

3. List the running containers on the node host and identify the container ID for the Docker registry:

```
# docker ps | grep ose-docker-registry
```

4. List the registry contents using the **`docker exec`** command:

```
# docker exec -it 4c01db0b339c find /registry
/registry/docker
/registry/docker/registry
/registry/docker/registry/v2
/registry/docker/registry/v2/blobs (1)
/registry/docker/registry/v2/blobs/sha256
/registry/docker/registry/v2/blobs/sha256/ed
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
d83c648c24f92cece5f89d95ac6c34ce751111810/data (2)
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845/data
/registry/docker/registry/v2/repositories (3)
/registry/docker/registry/v2/repositories/p1
/registry/docker/registry/v2/repositories/p1/pause (4)
/registry/docker/registry/v2/repositories/p1/pause/_manifests
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures (5)
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810/link (6)
```

```
/registry/docker/registry/v2/repositories/p1/pause/_uploads 7
/registry/docker/registry/v2/repositories/p1/pause/_layers 8
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4/link
9
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845/link
```

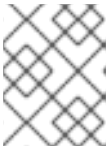
1. This directory stores all layers and signatures as blobs.
2. This file contains the blob's contents.
3. This directory stores all the image repositories.
4. This directory is for a single image repository **p1**/**pause**.
5. This directory contains signatures for a particular image manifest revision.
6. This file contains a reference back to a blob (which contains the signature data).
7. This directory contains any layers that are currently being uploaded and staged for the given repository.
8. This directory contains links to all the layers this repository references.
9. This file contains a reference to a specific layer that has been linked into this repository via an image.

2.5.5. Accessing the Registry

To access the registry directly, such as to perform **`docker push`** or **`docker pull`** operations, you must first log in to the registry using an access token.

1. Ensure you are logged in to OpenShift as a regular user:

```
$ oc login
```



NOTE

System users, such as **system:admin**, cannot obtain access tokens, and therefore cannot be used to access the registry directly.

2. Get your access token:

```
$ oc whoami -t
```

3. Log in to the Docker registry:

```
$ docker login -u <username> -e <any_email_address> \
    -p <token_value> <registry_service_host:port>
```

You can now perform **`docker pull`** and **`docker push`** operations against your registry. For example:

1. Pull an arbitrary image:

```
$ docker pull docker.io/busybox
```

2. Tag the new image with the form **`<registry_ip:port>/<project>/<image>`**:

```
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
```

3. Push the newly-tagged image to your registry:

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403ca
b55
```

2.5.6. Securing the Registry

Optionally, you can secure the registry so that it serves traffic via TLS:

1. Deploy the registry.

2. Fetch the service IP and port of the registry:

```
$ oc get svc docker-registry
NAME              LABELS
SELECTOR                  IP(S)            PORT(S)
docker-registry   docker-registry=default                    docker-
registry=default   172.30.124.220   5000/TCP
```

3. You can use an existing server certificate, or create a key and server certificate valid for specified IPs and host names, signed by a specified CA. To create a server certificate for the registry service IP and the **docker-registry.default.svc.cluster.local** host name:

```
$ oadm ca create-server-cert --signer-cert=ca.crt \
    --signer-key=ca.key --signer-serial=ca.serial.txt \
    --hostnames='docker-
registry.default.svc.cluster.local,172.30.124.220' \
    --cert=registry.crt --key=registry.key
```

4. Create the secret for the registry certificates:

```
$ oc secrets new registry-secret registry.crt registry.key
```

5. Add the secret to the registry pod's service account (i.e., the **default** service account):

```
$ oc secrets add serviceaccounts/default secrets/registry-secret
```

6. Add the secret volume to the registry deployment configuration:

```
$ oc volume dc/docker-registry --add --type=secret \
    --secret-name=registry-secret -m /etc/secrets
```

7. Enable TLS by adding the following environment variables to the registry deployment configuration:

```
$ oc env dc/docker-registry \
    REGISTRY_HTTP_TLS_CERTIFICATE=/etc/secrets/registry.crt \
    REGISTRY_HTTP_TLS_KEY=/etc/secrets/registry.key
```

See more details on overriding registry options.

8. Validate the registry is running in TLS mode. Wait until the **docker-registry** pod status changes to **Running** and verify the Docker logs for the registry container. You should find an entry for **listening on :5000, tls**.

```
$ oc get pods
POD                        IP           CONTAINER(S)   IMAGE(S)
HOST                            LABELS
STATUS    CREATED    MESSAGE
docker-registry-1-da73t   172.17.0.1
openshiftdev.local/127.0.0.1   deployment=docker-registry-
4,deploymentconfig=docker-registry,docker-registry=default   Running
38 hours

$ oc logs docker-registry-1-da73t | grep tls
time="2015-05-27T05:05:53Z" level=info msg="listening on :5000, tls"
instance.id=deeba528-c478-41f5-b751-dc48e4935fc2
```

9. Copy the CA certificate to the Docker certificates directory. This must be done on all nodes in the cluster:

```
$ sudo mkdir -p /etc/docker/certs.d/172.30.124.220:5000
$ sudo cp ca.crt /etc/docker/certs.d/172.30.124.220:5000

$ sudo mkdir -p /etc/docker/certs.d/docker-
registry.default.svc.cluster.local:5000
$ sudo cp ca.crt /etc/docker/certs.d/docker-
registry.default.svc.cluster.local:5000
```

10. Remove the **--insecure-registry** option only for this particular registry in the */etc/sysconfig/docker* file. Then, reload the daemon and restart the **docker** service to reflect this configuration change:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

11. Validate the **docker** client connection. Running **docker push** to the registry or **docker pull** from the registry should succeed. Make sure you have logged into the registry.

```
$ docker tag|push <registry/image> <internal_registry/project/image>
```

For example:

```
$ docker pull busybox
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403cab55
```

2.5.7. Exposing the Registry

To expose your internal registry externally, it is recommended that you run a secure registry. To expose the registry you must first have deployed a router.

1. Deploy the registry.
2. Secure the registry.
3. Deploy a router.
4. Create your passthrough route with **`oc create -f <filename>.json`**. The passthrough route will point to the registry service that you have created.

```
apiVersion: v1
kind: Route
metadata:
  name: registry
spec:
  host: <host> (1)
  to:
    kind: Service
    name: docker-registry (2)
  tls:
    termination: passthrough (3)
```

(1) The host for your route. You must be able to resolve this name externally via DNS to the router's IP address.

(2) The service name for your registry.

(3) Specify this route as a passthrough route.

NOTE

Passthrough is currently the only type of route supported for exposing the secure registry.

5. Next, you must trust the certificates being used for the registry on your host system. The certificates referenced were created when you secured your registry.

```
$ sudo mkdir -p /etc/docker/certs.d/<host>
$ sudo cp <ca certificate file> /etc/docker/certs.d/<host>
$ sudo systemctl restart docker
```

6. Log in to the registry using the information from securing the registry. However, this time point to the host name used in the route rather than your service IP. You should now be able to tag and push images using the route host.

```
$ oc get imagestreams -n test
NAME      DOCKER REPO   TAGS      UPDATED

$ docker pull busybox
$ docker tag busybox <host>/test/busybox
$ docker push <host>/test/busybox
The push refers to a repository [<host>/test/busybox] (len: 1)
8c2e06607696: Image already exists
6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image successfully pushed
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31
Status: Image is up to date for <host>/test/busybox:latest

$ oc get imagestreams -n test
NAME      DOCKER REPO                        TAGS      UPDATED
busybox   172.30.11.215:5000/test/busybox   latest    2 seconds ago
```

NOTE

Your image streams will have the IP address and port of the registry service, not the route name and port. See **`oc get imagestreams`** for details.

NOTE

In the **`<host>/test/busybox`** example above, **`test`** refers to the project name.

2.5.8. What's Next?

After you have a registry deployed, you can:

- Configure authentication; by default, authentication is set to Deny All.
- Deploy a router.

- Populate your OpenShift installation with a useful set of Red Hat-provided image streams and templates.

2.6. DEPLOYING A ROUTER

2.6.1. Overview

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.

NOTE

The F5 router plug-in is available starting in OpenShift Enterprise 3.0.2.

2.6.2. Creating the Router Service Account

Starting in OpenShift Enterprise 3.0.1.0, you must first create a service account for the router before deploying. This service account must have permissions to a security context constraint (SCC) that allows it to specify host ports.

Create a service account, for example named **router**:

```
$ echo \
    '{"kind":"ServiceAccount","apiVersion":"v1","metadata":
{"name":"router"}}' \
    | oc create -f -
```

Edit the **privileged** SCC:

```
$ oc edit scc privileged
```

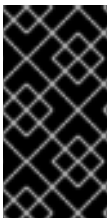
Add the **router** service account in the form of **system:serviceaccount:<project>:<name>** to the **users** section:

```
...
users:
- system:serviceaccount:openshift-infra:build-controller
- system:serviceaccount:default:router
```

2.6.3. Deploying the Default HAProxy Router

The **oadm router** command is provided with the administrator CLI to simplify the tasks of setting up

routers in a new installation. Just about every form of communication between OpenShift components is secured by TLS and uses various certificates and authentication methods. Use the **`--credentials`** option to specify what credentials the router should use to contact the master.



IMPORTANT

Routers directly attach to port 80 and 443 on all interfaces on a host. Restrict routers to hosts where port 80/443 is available and not being consumed by another service, and set this using node selectors and the scheduler configuration. As an example, you can achieve this by dedicating infrastructure nodes to run services such as routers.

First, ensure you have created the router service account before deploying a router.

To check if a default router, named **router**, already exists:

```
$ oadm router --dry-run \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router
```

To see what the default router would look like if created:

```
$ oadm router -o yaml \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router
```

To create a router if it does not exist:

```
$ oadm router <router_name> --replicas=<number> \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router
```

Multiple instances are created on different hosts according to the scheduler policy.

To use a different router image and view the router configuration that would be used:

```
$ oadm router <router_name> -o <format> --images=<image> \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router
```

For example:

```
$ oadm router region-west -o yaml --images=myrepo/somerouter:mytag \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router
```

2.6.3.1. High Availability

You can set up a highly-available router on your OpenShift cluster using IP failover.

2.6.3.2. Customizing the Default Routing Subdomain

You can customize the suffix used as the default routing subdomain for your environment using the master configuration file (the */etc/openshift/master/master-config.yaml* file by default). The following example shows how you can set the configured suffix to **v3.openshift.test**:

Example 2.5. Master Configuration Snippet

```
routingConfig:
  subdomain: v3.openshift.test
```

NOTE

This change requires a restart of the master if it is running.

With the OpenShift master(s) running the above configuration, the generated host name for the example of a host added to a namespace **mynamespace** would be:

Example 2.6. Generated Host Name

```
myroute-mynamespace.v3.openshift.test
```

2.6.3.3. Using Wildcard Certificates

A TLS-enabled route that does not include a certificate uses the router's default certificate instead. In most cases, this certificate should be provided by a trusted certificate authority, but for convenience you can use the OpenShift CA to create the certificate. For example:

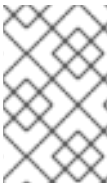
```
$ CA=/etc/openshift/master
$ oadm ca create-server-cert --signer-cert=$CA/ca.crt \
      --signer-key=$CA/ca.key --signer-serial=$CA/ca.serial.txt \
      --hostnames='*.cloudapps.example.com' \
      --cert=cloudapps.crt --key=cloudapps.key
```

The router expects the certificate and key to be in PEM format in a single file:

```
$ cat cloudapps.crt cloudapps.key $CA/ca.crt > cloudapps.router.pem
```

From there you can use the **`--default-cert`** flag:

```
$ oadm router --default-cert=cloudapps.router.pem \
    --credentials="$KUBECONFIG" --service-account=router
```



NOTE

Browsers only consider wildcards valid for subdomains one level deep. So in this example, the certificate would be valid for *a.cloudapps.example.com* but not for *a.b.cloudapps.example.com*.

2.6.3.4. Using Secured Routes

Currently, password protected key files are not supported. HAProxy prompts for a password upon starting and does not have a way to automate this process. To remove a passphrase from a keyfile, you can run:

```
# openssl rsa -in <passwordProtectedKey.key> -out <new.key>
```

Here is an example of how to use a secure edge terminated route with TLS termination occurring on the router before traffic is proxied to the destination. The secure edge terminated route specifies the TLS certificate and key information. The TLS certificate is served by the router front end.

First, start up a router instance:

```
# oadm router --replicas=1 --credentials=$KUBECONFIG --service-
account=router
```

Next, create a private key, csr and certificate for our edge secured route. The instructions on how to do that would be specific to your certificate authority and provider. For a simple self-signed certificate for a domain named **`www.example.test`**, see the example shown below:

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr  \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=www.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
      -signkey example-test.key -out example-test.crt
```

Generate a route configuration file using the above certificate and key. Make sure to replace servicename **`my-service`** with the name of your service.

```
# servicename="my-service"
# echo "
apiVersion: v1
kind: Route
metadata:
  name:  secured-edge-route
spec:
  host: www.example.test
  to:
    kind: Service
    name: $servicename
  tls:
    termination: edge
    key: |
$(openssl rsa -in example-test.key | sed 's/^/      /')
    certificate: |
$(openssl x509 -in example-test.crt | sed 's/^/      /')

" > example-test-route.yaml
```

Finally add the route to OpenShift (and the router) via:

```
# oc create -f example-test-route.yaml
```

Make sure your DNS entry for **`www.example.test`** points to your router instance(s) and the route to your domain should be available. The example below uses curl along with a local resolver to simulate the DNS lookup:

```
# routerip="4.1.1.1"  #  replace with IP address of one of your router
instances.
# curl -k --resolve www.example.test:443:$routerip
https://www.example.test/
```

2.6.3.5. Using the Container Network Stack

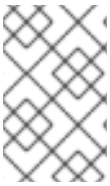
The OpenShift router runs inside a Docker container and the default behavior is to use the network stack of the host (i.e., the node where the router container runs). This default behavior benefits performance because network traffic from remote clients does not need to take multiple hops through user space to reach the target service and container.

Additionally, this default behavior enables the router to get the actual source IP address of the remote connection rather than getting the node's IP address. This is useful for defining ingress rules based on the originating IP, supporting sticky sessions, and monitoring traffic, among other uses.

This host network behavior is controlled by the **`--host-network`** router command line option, and the default behaviour is the equivalent of using **`--host-network=true`**. If you wish to run the router with the container network stack, use the **`--host-network=false`** option when creating the router. For example:

```
$ oadm router \
    --credentials='/etc/openshift/master/openshift-router.kubeconfig' \
    --service-account=router \
    --host-network=false
```

Internally, this means the router container must publish the 80 and 443 ports in order for the external network to communicate with the router.



NOTE

Running with the container network stack means that the router sees the source IP address of a connection to be the NATed IP address of the node, rather than the actual remote IP address.

2.6.4. Deploying a Customized HAProxy Router

The HAProxy router is based on a **golang** template that generates the HAProxy configuration file from a list of routes. If you want a customized template router to meet your needs, you can customize the template file, build a new Docker image, and run a customized router.

One common case for this might be implementing new features within the application back ends. For example, it might be desirable in a highly-available setup to use stick-tables that synchronizes between peers. The router plug-in provides all the facilities necessary to make this customization.

You can obtain a new ***haproxy-config.template*** file from the latest router image by running:

```
# docker run --rm --interactive=true --tty --entrypoint=cat \
    registry.access.redhat.com/openshift3/ose-haproxy-router:v3.0.2.0
haproxy-config.template
```

-

Save this content to a file for use as the basis of your customized template.

2.6.4.1. Using Stick Tables

The following example customization can be used in a highly-available routing setup to use stick-tables that synchronize between peers.

Adding a Peer Section

In order to synchronize stick-tables amongst peers you must a define a peers section in your HAProxy configuration. This section determines how HAProxy will identify and connect to peers. The plug-in provides data to the template under the **`.PeerEndpoints`** variable to allow you to easily identify members of the router service. You may add a peer section to the ***haproxy-config.template*** file inside the router image by adding:

```
{{ if (len .PeerEndpoints) gt 0 }}
peers openshift_peers
  {{ range $endpointID, $endpoint := .PeerEndpoints }}
    peer {{$endpoint.TargetName}} {{$endpoint.IP}}:1937
  {{ end }}
{{ end }}
```

Changing the Reload Script

When using stick-tables, you have the option of telling HAProxy what it should consider the name of the local host in the peer section. When creating endpoints, the plug-in attempts to set the **`TargetName`** to the value of the endpoint's **`TargetRef.Name`**. If **`TargetRef`** is not set, it will set the **`TargetName`** to the IP address. The **`TargetRef.Name`** corresponds with the Kubernetes host name, therefore you can add the **`-L`** option to the **`reload-haproxy`** script to identify the local host in the peer section.

```
peer_name=$HOSTNAME (1)

if [ -n "$old_pid" ]; then
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name -sf
$old_pid
else
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name
fi
```

(1) Must match an endpoint target name that is used in the peer section.

Modifying Back Ends

Finally, to use the stick-tables within back ends, you can modify the HAProxy configuration to use the stick-tables and peer set. The following is an example of changing the existing back end for TCP connections to use stick-tables:

```
            {{ if eq $cfg.TLSTermination "passthrough" }}
backend be_tcp_{{$cfgIdx}}
  balance leastconn
  timeout check 5000ms
  stick-table type ip size 1m expire 5m{{ if (len $.PeerEndpoints) gt 0 }}
peers openshift_peers {{ end }}
```

```
  stick on src
                {{ range $endpointID, $endpoint :=
$serviceUnit.EndpointTable }}
  server {{$endpointID}} {{$endpoint.IP}}:{{$endpoint.Port}} check inter
5000ms
                {{ end }}
            {{ end }}
```

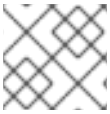
After this modification, you can rebuild your router.

2.6.4.2. Rebuilding Your Router

After you have made any desired modifications to the template, such as the example stick tables customization, you must rebuild your router for your changes to go in effect:

1. Rebuild the Docker image to include your customized template.
2. Push the resulting image to your repository.
3. Create the router specifying your new image, either:
   a. in the pod's object definition directly, or
   b. by adding the **`--images=<repo>/<image>:<tag>`** flag to the **`oadm router`** command when creating a highly-available routing service.

2.6.5. Deploying the F5 Router



NOTE

The F5 router plug-in is available starting in OpenShift Enterprise 3.0.2.

The F5 router plug-in is provided as a Docker image and run as a pod, just like the default HAProxy router. Deploying the F5 router is done similarly as well, using the **`oadm router`** command but providing additional flags (or environment variables) to specify the following parameters for the **F5 BIG-IP®** host:

Flag	Description
`--type=f5-router`	Specifies that an F5 router should be launched (the default **`--type`** is **haproxy-router**).
`--external-host`	Specifies the **F5 BIG-IP®** host's management interface's host name or IP address.
`--external-host-username`	Specifies the **F5 BIG-IP®** user name (typically **admin**).
`--external-host-password`	Specifies the **F5 BIG-IP®** password.

Flag	Description
`--external-host-http-vserver`	Specifies the name of the F5 virtual server for HTTP connections.
`--external-host-https-vserver`	Specifies the name of the F5 virtual server for HTTPS connections.
`--external-host-private-key`	Specifies the path to the SSH private key file for the **F5 BIG-IP®** host. Required to upload and delete key and certificate files for routes.
`--external-host-insecure`	A Boolean flag that indicates that the F5 router should skip strict certificate verification with the **F5 BIG-IP®** host.

As with the HAProxy router, the **`oadm router`** command creates the service and deployment configuration objects, and thus the replication controllers and pod(s) in which the F5 router itself runs. The replication controller restarts the F5 router in case of crashes. Because the F5 router is only watching routes and endpoints and configuring **F5 BIG-IP®** accordingly, running the F5 router in this way along with an appropriately configured **F5 BIG-IP®** deployment should satisfy high-availability requirements.

To deploy the F5 router:

1. First, establish a tunnel using a ramp node, which allows for the routing of traffic to pods through the OpenShift SDN.
2. Ensure you have created the router service account.
3. Run the **`oadm router`** command with the appropriate flags. For example:

```
$ oadm router \
    --type=f5-router \
    --external-host=10.0.0.2 \
    --external-host-username=admin \
    --external-host-password=mypassword \
    --external-host-http-vserver=ose-vserver \
    --external-host-https-vserver=https-ose-vserver \
    --external-host-private-key=/path/to/key \
    --credentials='/etc/openshift/master/openshift-
router.kubeconfig' \(1)
    --service-account=router
```

(1) **`--credentials`** is the path to the CLI configuration file for the **openshift-router**. It is recommended using an **openshift-router** specific profile with appropriate permissions.

2.6.5.1. F5 Router Partition Paths

Partition paths allow you to store your OpenShift routing configuration in a custom **F5 BIG-IP®** administrative partition, instead of the default /**Common** partition. You can use custom administrative partitions to secure **F5 BIG-IP®** environments. This means that an OpenShift-specific configuration stored in **F5 BIG-IP®** system objects reside within a logical container, allowing administrators to define access control policies on that specific administrative partition.

See the **F5 BIG-IP®** documentation for more information about administrative partitions.

Use the **`--external-host-partition-path`** flag when deploying the F5 router to specify a partition path:

```
$ oadm router --external-host-partition-path=/OpenShift/zone1 ...
```

2.6.6. What's Next?

If you deployed an HAProxy router, you can learn more about monitoring the router.

If you have not yet done so, you can:

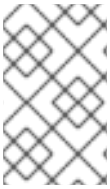
- Configure authentication; by default, authentication is set to Deny All.
- Deploy an integrated Docker registry.
- Populate your OpenShift installation with a useful set of Red Hat-provided image streams and templates.

2.7. FIRST STEPS

2.7.1. Overview

You can populate your OpenShift installation with a useful set of Red Hat-provided image streams and templates to make it easy for developers to create new applications. By default, the quick installation and advanced installation methods automatically create these sets in the **openshift** project, which is a default project to which all users have view access.

Use the following instructions to create the objects yourself. The files are installed on the file system of your master.



NOTE

This topic is only necessary if you installed OpenShift using a method other than the quick installation or the advanced installation. Image streams and templates will be automatically populated in the **`openshift`** project when using these methods.

2.7.2. Prerequisites

- The integrated Docker registry service must be deployed in your OpenShift installation.
- You must be able to run the following CLI commands with **cluster-admin** privileges, because they operate on the default **openshift**project.
- You must have cloned the repository that contains the supported imagestreams:

```
$ git clone https://github.com/openshift/openshift-ansible
```

-

2.7.3. Creating Image Streams for OpenShift Images

The core set of image streams provide images that can be used to build **Node.js**, **Perl**, **PHP**, **Python**, and **Ruby** applications. It also defines images for **MongoDB**, **MySQL**, and **PostgreSQL** to support data storage.

If your node hosts are subscribed using Red Hat Subscription Manager and you want to use the Red Hat Enterprise Linux (RHEL) 7 based images:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/image-
streams/image-streams-rhel7.json \
    -n openshift
```

Alternatively, to create the core set of image streams that use the CentOS 7 based images:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/image-
streams/image-streams-centos7.json \
    -n openshift
```

It is not possible to create both the CentOS and RHEL sets of image streams because they use the same names. If you desire to have both sets of image streams available to users, either create one set in a different project, or edit one of the files and modify the image stream names to make them unique.

2.7.4. Creating Image Streams for xPaaS Middleware Images

The xPaaS Middleware image streams provide images for **JBoss EAP**, **JBoss JWS**, and **JBoss A-MQ**. They can be used to build applications for those platforms using the provided templates.

To create the xPaaS Middleware set of image streams:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/xpaas-
streams/jboss-image-streams.json \
    -n openshift
```

NOTE

Access to the images referenced by these image streams requires the relevant xPaaS Middleware subscriptions.

2.7.5. Creating Database Service Templates

The database service templates make it easy to run a database image which can be utilized by other components. For each database (**MongoDB**, **MySQL**, and **PostgreSQL**), two templates are defined.

One template uses ephemeral storage in the container which means data stored will be lost if the container is restarted, for example if the pod moves. This template should be used for demonstration purposes only.

The other template defines a persistent volume for storage, however it requires your OpenShift installation to have persistent volumes configured.

To create the core set of database templates:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/db-templates
-n openshift
```

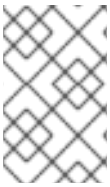
After creating the templates, users are able to easily instantiate the various templates, giving them quick access to a database deployment.

2.7.6. Creating InstantApp Templates

The InstantApp templates define a full set of objects for a running application. These include:

- Build configurations to build the application from source located in a GitHub public repository
- Deployment configurations to deploy the application image after it is built.
- Services to provide load balancing for the application pods.
- Routes to provide external access to the application.

Some of the templates also define a database deployment and service so the application can perform database operations.



NOTE

The templates which define a database use ephemeral storage for the database content. These templates should be used for demonstration purposes only as all database data will be lost if the database pod restarts for any reason.

After creating the templates, users are able to easily instantiate full applications using the various language images provided with OpenShift. They can also customize the template parameters during instantiation so that it builds source from their own repository rather than the sample repository, so this provides a simple starting point for building new applications.

To create the core InstantApp templates:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/quickstart-
templates -n openshift
```

There is also a set of templates for creating applications using various xPaaS Middleware products (**JBoss EAP**, **JBoss JWS**, and **JBoss A-MQ**), which can be registered by running:

```
$ oc create -f \
    openshift-ansible/roles/openshift_examples/files/examples/xpaas-
templates -n openshift
```

NOTE

The xPaaS Middleware templates require the xPaaS Middleware image streams, which in turn require the relevant xPaaS Middleware subscriptions.

NOTE

The templates which define a database use ephemeral storage for the database content. These templates should be used for demonstration purposes only as all database data will be lost if the database pod restarts for any reason.

2.7.7. What's Next?

With these artifacts created, developers can now log into the web console and follow the flow for creating from a template. Any of the database or application templates can be selected to create a running database service or application in the current project. Note that some of the application templates define their own database services as well.

The example applications are all built out of GitHub repositories which are referenced in the templates by default, as seen in the **SOURCE_REPOSITORY_URL** parameter value. Those repositories can be forked, and the fork can be provided as the **SOURCE_REPOSITORY_URL** parameter value when creating from the templates. This allows developers to experiment with creating their own applications.

You can direct your developers to the Using the InstantApp Templates section in the Developer Guide for these instructions.

CHAPTER 3. UPGRADING OPENSHIFT

3.1. OVERVIEW

When new versions of OpenShift are released, you can upgrade your cluster to apply the latest enhancements and bug fixes. See the OpenShift Enterprise 3.0 Release Notes to review the latest changes.

Unless noted otherwise, node and masters within a major version are forward and backward compatible, so upgrading your cluster should go smoothly. However, you should not run mismatched versions longer than necessary to upgrade the entire cluster.

Starting with OpenShift 3.0.2, if you installed using the advanced installation and the inventory file that was used is available, you can use the upgrade playbook to automate the upgrade process. Alternatively, you can upgrade OpenShift manually.

NOTE

This topic pertains to RPM-based installations only (i.e., the quick and advanced installation methods) and does not currently cover container-based installations.

3.2. USING THE AUTOMATED UPGRADE PLAYBOOK

Starting with OpenShift 3.0.2, if you installed using the advanced installation and the inventory file that was used is available, you can use the upgrade playbook to automate the upgrade process. This playbook performs the following steps for you:

- Applies the latest configuration by re-running the installation playbook.
- Upgrades and restart master services.
- Upgrades and restart node services.
- Applies the latest cluster policies.
- Updates the default router if one exists.
- Updates the default registry if one exists.
- Updates default image streams and InstantApp templates.

IMPORTANT

The upgrade playbook re-runs cluster configuration steps, therefore any settings that are not stored in your inventory file will be overwritten. The playbook creates a backup of any files that are changed, and you should carefully review the differences after the playbook finishes to ensure that your environment is configured as expected.

IMPORTANT

Running Ansible playbooks with the `--tags` or `--check` options is not supported by Red Hat.

Ensure that you have the latest **openshift-ansible** code checked out, then run the playbook utilizing the default **ansible-hosts** file located in */etc/ansible/hosts*. If your ***hosts*** file is located somewhere else, add the **-i** flag to specify the location:

```
# cd ~/openshift-ansible
# git pull https://github.com/openshift/openshift-ansible master
# ansible-playbook [-i /path/to/hosts/file]
playbooks/adhoc/upgrades/upgrade.yml
```

After the upgrade playbook finishes, verify that all nodes are marked as **Ready** and that you are running the expected versions of the **docker-registry** and **router** images:

```
# oc get nodes
NAME                 LABELS
STATUS
master.example.com
kubernetes.io/hostname=master.example.com,region=infra,zone=default
Ready
node1.example.com
kubernetes.io/hostname=node1.example.com,region=primary,zone=east
Ready

# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.0.2.0",
# oc get -n default dc/docker-registry -ojson | grep \"image\"
    "image": "openshift3/ose-docker-registry:v3.0.2.0",
```

After upgrading, you can use the experimental diagnostics tool to look for common issues:

```
# openshift ex diagnostics
...
[Note] Summary of diagnostics execution:
[Note] Completed with no errors or warnings seen.
```

3.3. UPGRADING MANUALLY

As an alternative to using the automated upgrade playbook, you can manually upgrade your OpenShift cluster. To manually upgrade without disruption, it is important to upgrade each component as documented in this topic. Before you begin your upgrade, familiarize yourself with the entire procedure. Specific releases may require additional steps to be performed at key points during the standard upgrade process.

3.3.1. Upgrading Masters

Upgrade your masters first. On each master host, upgrade the **openshift-master** package:

```
# yum upgrade openshift-master
```

Then, restart the **openshift-master** service and review its logs to ensure services have been restarted successfully:

```
# systemctl restart openshift-master
# journalctl -r -u openshift-master
```

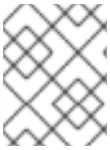
3.3.2. Updating Policy Definitions

After a cluster upgrade, the recommended default cluster roles may have been updated. To check if an update is recommended for your environment, you can run:

```
# oadm policy reconcile-cluster-roles
```

This command outputs a list of roles that are out of date and their new proposed values. For example:

```
# oadm policy reconcile-cluster-roles
apiVersion: v1
items:
- apiVersion: v1
  kind: ClusterRole
  metadata:
    creationTimestamp: null
    name: admin
  rules:
  - attributeRestrictions: null
    resources:
    - builds/custom
...
```



NOTE

Your output will vary based on the OpenShift version and any local customizations you have made. Review the proposed policy carefully.

You can either modify this output to re-apply any local policy changes you have made, or you can automatically apply the new policy by running:

```
# oadm policy reconcile-cluster-roles --confirm
```

3.3.3. Upgrading Nodes

After upgrading your masters, you can upgrade your nodes. When restarting the **openshift-node** service, there will be a brief disruption of outbound network connectivity from running pods to services while the service proxy is restarted. The length of this disruption should be very short and scales based on the number of services in the entire cluster.

On each node host, upgrade all **openshift** packages:

```
# yum upgrade openshift\*
```

Then, restart the **openshift-node** service:

```
# systemctl restart openshift-node
```

As a user with **cluster-admin** privileges, verify that all nodes are showing as **Ready**:

```
# oc get nodes
NAME                     LABELS
```

```
STATUS
master.example.com      kubernetes.io/hostname=master.example.com
Ready,SchedulingDisabled
node1.example.com       kubernetes.io/hostname=node1.example.com
Ready
node2.example.com       kubernetes.io/hostname=node2.example.com
Ready
```

3.3.4. Upgrading the Router

If you have previously deployed a router, the router deployment configuration must be upgraded to apply updates contained in the router image. To upgrade your router without disrupting services, you must have previously deployed a highly-available routing service.

IMPORTANT

If you are upgrading to OpenShift Enterprise 3.0.1.0 or 3.0.2.0, first see the Additional Manual Instructions per Release section for important steps specific to your upgrade, then continue with the router upgrade as described in this section.

Edit your router's deployment configuration. For example, if it has the default **router** name:

```
# oc edit dc/router
```

Apply the following changes:

```
...
spec:
 template:
    spec:
      containers:
      - env:
        ...
        image: registry.access.redhat.com/openshift3/ose-haproxy-
router:v3.0.2.0 (1)
        imagePullPolicy: IfNotPresent
        ...
```

(1) Adjust the image version to match the version you are upgrading to.

You should see one router pod updated and then the next.

3.3.5. Upgrading the Registry

The registry must also be upgraded for changes to take effect in the registry image. If you have used a **`PersistentVolumeClaim`** or a host mount point, you may restart the registry without losing the contents of your registry. The registry installation topic details how to configure persistent storage.

Edit your registry's deployment configuration:

```
# oc edit dc/docker-registry
```

Apply the following changes:

```
...
spec:
 template:
    spec:
      containers:
      - env:
        ...
        image: registry.access.redhat.com/openshift3/ose-docker-
registry:v3.0.2.0 1
        imagePullPolicy: IfNotPresent
        ...
```

1 Adjust the image version to match the version you are upgrading to.

IMPORTANT

Images that are being pushed or pulled from the internal registry at the time of upgrade will fail and should be restarted automatically. This will not disrupt pods that are already running.

3.3.6. Updating the Default Image Streams and Templates

By default, the quick installation and advanced installation methods automatically create default image streams, QuickStart templates, and database service templates in the **openshift** project, which is a default project to which all users have view access. These objects were created during installation from the JSON files located under ***/usr/share/openshift/examples***. Running the latest installer will copy newer files into place, but it does not currently update the **openshift** project.

You can update the **openshift** project by running the following commands. It is expected that you will receive warnings about items that already exist.

```
# oc create -n openshift -f /usr/share/openshift/examples/image-
streams/image-streams-rhel7.json
# oc create -n openshift -f /usr/share/openshift/examples/db-templates
# oc create -n openshift -f /usr/share/openshift/examples/quickstart-
templates
# oc create -n openshift -f /usr/share/openshift/examples/xpaas-streams
# oc create -n openshift -f /usr/share/openshift/examples/xpaas-templates
# oc replace -n openshift -f /usr/share/openshift/examples/image-
streams/image-streams-rhel7.json
# oc replace -n openshift -f /usr/share/openshift/examples/db-templates
# oc replace -n openshift -f /usr/share/openshift/examples/quickstart-
templates
# oc replace -n openshift -f /usr/share/openshift/examples/xpaas-streams
# oc replace -n openshift -f /usr/share/openshift/examples/xpaas-templates
```

3.3.7. Importing the Latest Images

After updating the default image streams, you may also want to ensure that the images within those streams are updated. For each image stream in the default **openshift** project, you can run:

```
# oc import-image -n openshift <imagestream>
```

For example, get the list of all image streams in the default **openshift** project:

```
# oc get is -n openshift
NAME     DOCKER REPO
TAGS                  UPDATED
mongodb  registry.access.redhat.com/openshift3/mongodb-24-rhel7
2.4,latest,v3.0.0.0    16 hours ago
mysql    registry.access.redhat.com/openshift3/mysql-55-rhel7
5.5,latest,v3.0.0.0    16 hours ago
nodejs   registry.access.redhat.com/openshift3/nodejs-010-rhel7
0.10,latest,v3.0.0.0   16 hours ago
...
```

Update each image stream one at a time:

```
# oc import-image -n openshift nodejs
Waiting for the import to complete, CTRL+C to stop waiting.
The import completed successfully.

Name:                   nodejs
Created:                16 hours ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2015-07-
21T13:17:00Z
Docker Pull Spec:       registry.access.redhat.com/openshift3/nodejs-010-
rhel7

Tag             Spec            Created         PullSpec
Image
0.10            latest          16 hours ago
registry.access.redhat.com/openshift3/nodejs-010-rhel7:latest
66d92cebc0e48e4e4be3a93d0f9bd54f21af7928ceaa384d20800f6e6fcf669f
latest                          16 hours ago
registry.access.redhat.com/openshift3/nodejs-010-rhel7:latest
66d92cebc0e48e4e4be3a93d0f9bd54f21af7928ceaa384d20800f6e6fcf669f
v3.0.0.0        <pushed>        16 hours ago
registry.access.redhat.com/openshift3/nodejs-010-rhel7:v3.0.0.0
66d92cebc0e48e4e4be3a93d0f9bd54f21af7928ceaa384d20800f6e6fcf669f
```

IMPORTANT

In order to update your S2I-based applications, you must manually trigger a new build of those applications after importing the new images using **`oc start-build <app-name>`**.

3.4. ADDITIONAL MANUAL STEPS PER RELEASE

Some OpenShift releases may have additional instructions specific to that release that must be performed to fully apply the updates across the cluster. Read through the following sections carefully depending on your upgrade path, as you may be required to perform certain steps and key points during the standard upgrade process described earlier in this topic.

See the OpenShift Enterprise 3.0 Release Notes to review the latest release notes.

3.4.1. OpenShift Enterprise 3.0.1.0

The following steps are required for the OpenShift Enterprise 3.0.1.0 release.

Creating a Service Account for the Router

The default HAProxy router was updated to utilize host ports and requires that a service account be created and made a member of the privileged security context constraint (SCC). Additionally, "down-then-up" rolling upgrades have been added and is now the preferred strategy for upgrading routers.

After upgrading your master and nodes but before updating to the newer router, you must create a service account for the router. As a cluster administrator, ensure you are operating on the **default** project:

```
# oc project default
```

Delete any existing **router** service account and create a new one:

```
# oc delete serviceaccount/router
serviceaccounts/router

# echo '{"kind":"ServiceAccount","apiVersion":"v1","metadata":
{"name":"router"}}' | oc create -f -
serviceaccounts/router
```

Edit the **privileged** SCC:

```
# oc edit scc privileged
```

Apply the following changes:

```
allowHostDirVolumePlugin: true
allowHostNetwork: true (1)
allowHostPorts: true (2)
allowPrivilegedContainer: true
...
users:
- system:serviceaccount:openshift-infra:build-controller
- system:serviceaccount:default:router (3)
```

1. Add or update **`allowHostNetwork: true`**.
2. Add or update **`allowHostPorts: true`**.
3. Add the service account you created to the **`users`** list at the end of the file.

Edit your router's deployment configuration:

```
# oc edit dc/router
```

Apply the following changes:

```
...
spec:
  replicas: 2
  selector:
    router: router
  strategy:
    resources: {}
    rollingParams:
      intervalSeconds: 1
      timeoutSeconds: 120
      updatePeriodSeconds: 1
      updatePercent: -10 (1)
    type: Rolling
    ...
  template:
    ...
    spec:
      ...
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      serviceAccount: router (2)
      serviceAccountName: router (3)
...
```

1. Add **`updatePercent: -10`** to allow down-then-up rolling upgrades.
2. Add **`serviceAccount: router`** to the template **`spec`**.
3. Add **`serviceAccountName: router`** to the template **`spec`**.

Now upgrade your router per the standard router upgrade steps.

3.4.2. OpenShift Enterprise 3.0.2.0

The following steps are required for the OpenShift Enterprise 3.0.2.0 release.

Switching the Router to Use the Host Network Stack

The default HAProxy router was updated to use the host networking stack by default instead of the former behavior of using the container network stack, which proxied traffic to the router, which in turn proxied the traffic to the target service and container. This new default behavior benefits performance because network traffic from remote clients no longer needs to take multiple hops through user space in order to reach the target service and container.

Additionally, the new default behavior enables the router to get the actual source IP address of the remote connection. This is useful for defining ingress rules based on the originating IP, supporting sticky sessions, and monitoring traffic, among other uses.

Existing router deployments will continue to use the container network stack unless modified to switch to using the host network stack.

To switch the router to use the host network stack, edit your router's deployment configuration:

```
# oc edit dc/router
```

Apply the following changes:

```
...
spec:
  replicas: 2
  selector:
    router: router
    ...
  template:
    ...
    spec:
      ...
      ports:
        - containerPort: 80 (1)
          hostPort: 80
          protocol: TCP
        - containerPort: 443 (2)
          hostPort: 443
          protocol: TCP
        - containerPort: 1936 (3)
          hostPort: 1936
          name: stats
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
      hostNetwork: true (4)
      restartPolicy: Always
...
```

(1) (2) (3) For host networking, ensure that the **containerPort** value matches the **hostPort** values for each of the ports.

(4) Add **hostNetwork: true** to the template **spec**.

Now upgrade your router per the standard router upgrade steps.

Configuring serviceNetworkCIDR for the SDN

Add the **serviceNetworkCIDR** parameter to the **networkConfig** section in */etc/openshift/master/master-config.yaml*. This value should match the **servicesSubnet** value in the **kubernetesMasterConfig** section:

```
kubernetesMasterConfig:
  servicesSubnet: 172.30.0.0/16
...
networkConfig:
  serviceNetworkCIDR: 172.30.0.0/16
```

Adding the Scheduler Configuration API Version

The scheduler configuration file incorrectly lacked **kind** and **apiVersion** fields when deployed using the quick or advanced installation methods. This will affect future upgrades, so it is important to add those values if they do not exist.

Modify the */etc/openshift/master/scheduler.json* file to add the **kind** and **apiVersion** fields:

```
{
  "kind": "Policy", 1
  "apiVersion": "v1", 2
  "predicates": [
  ...
}
```

1 Add **"kind": "Policy",**

2 Add **"apiVersin": "v1",**

CHAPTER 4. REVISION HISTORY: INSTALLATION AND CONFIGURATION

4.1. MON MAR 28 2016

Affected Topic	Description of Change
Prerequisites	Updated the topics to reflect that the Ansible installer playbooks now ship with OpenShift.
Quick Installation	
Advanced Installation	

4.2. MON MAR 21 2016

Affected Topic	Description of Change
Installing	Fixed broken links.

4.3. MON FEB 29 2016

Affected Topic	Description of Change
Prerequisites	Fixed the ***/etc/selinux/config*** file path in the SELinux section.

4.4. THU FEB 25 2016

Affected Topic	Description of Change
Installing → Advanced Installation	Added notes indicating that moving from a single master cluster to multiple masters after installation is not supported.

4.5. MON FEB 22 2016

Affected Topic	Description of Change
Prerequisites	Added an SELinux section to include guidance that SELinux must be enabled, or the installer will fail.

4.6. WED FEB 17 2016

Affected Topic	Description of Change
Installing → Advanced Installation	Added **`openshift_master_cluster_method=pacemaker`** as a requirement to the example inventory file in the Multiple Masters, Multiple etcd, and Multiple Nodes section.

4.7. MON FEB 15 2016

Affected Topic	Description of Change
Installing → Prerequisites	Updated to include guidance on how to check if Docker is running.

4.8. MON FEB 08 2016

Affected Topic	Description of Change
Installing → Deploying a Docker Registry	Replaced the deprecated **`oc log`** command with the new **`oc logs`** command in the TLS-mode example.
Installing → Prerequisites	Updated the System Requirements section to clarify that instances can be running on a private IaaS, not just a public one.

4.9. TUE JUN 23 2015

OpenShift Enterprise 3.0 release.